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

MANAGING SIMULATION WORKFLOW PATTERNS USING DYNAMIC SERVICE-ORIENTED COMPOSITIONS

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

Distributed simulation usage in industry has been limited due to its high cost in comparison to its returned benefits. A number of surveys of experts from different background suggested the need of distributed simulation features to overcome its challenges and cost. The RESTful Interoperability Simulation Environment (RISE) middleware, based on RESTful Web-services, deals with these issues. However, simulation assets also need to be part of a formal Business Process Management (BPM) to allow practical across-enterprise collaboration. The Workflow mechanism introduced here promises to help with this situation. Further, these workflows provide automation, repeatable and reusable simulation experiments. We present the design of a workflow component that is capable of managing and executing different workflow patterns across various simulation RISE servers. We further present in detail a number of simulation workflow patterns executed by the workflow component.

1 INTRODUCTION

Distributed simulation deals with executing simulation over geographically interconnected assets (Wainer et al. 2010). This offers many benefits such as allowing across-organization simulation collaboration without being in the same location, and the reuse of simulation assets. Other benefits also include reducing execution time, interoperating different vendor simulation toolkits, providing fault tolerance and information hiding – including the protection of intellectual property rights (Boer et al. 2008; Strassburger et al. 2008; Wainer et al. 2010). Distributed simulation is usually performed by (1) partitioning a single model among different logical processes (LP) to perform a single simulation run, or (2) by having a simulation run synchronized among different simulation models where each model is handled by a different LP. The latter approach is usually more attractive when simulation models are heterogeneous (for instance, when they were intended to run on a specific simulation environment). On the other hand, the first approach is usually better for LPs belonging to the same simulation environment.

Distributed simulation practice is now commonly used in the military sector, but is still limited in industry, mainly due to its high cost with respect to its return-of-investment (ROI), as indicated by a number of surveys (Boer et al. 2008; Strassburger et al. 2008; Wainer et al. 2010). Surveys such as (Strassburger et al. 2008) pointed out certain challenges that distributed simulation must overcome to make practical advancement in industry, in particular having plug-and-play or automatic middleware interoperability between heterogeneous simulation assets (in order to do it effortlessly and with rational cost). This leads to the concept of lightweight commercial-off-the-shelf (COTS) components, which can be assembled and interoperated with each other efficiently, effortlessly and quickly, with a business mentality of “Try-before-buy” way of thinking. The RESTful Interoperability Simulation Environment (RISE) middleware (Al-Zoubi et al. 2009; Wainer et al. 2010) is one of such plug-and-play interoperability middleware. It is based on RESTful Web-services and it is independent of any specific simulation formalism or software environment, as discussed in (Wainer et al. 2010).

Interoperability translates into collaboration bridges among organizations, resulting in less cost with rapid product development. Simulation assets also need to be interoperated, allowing collaboration in studying systems across organizations. To do so, three issues need to be resolved: (1) Plug-and-play interoperability between independent-developed simulation assets. In our case, this is done using the RISE middleware (Al-Zoubi et al. 2009; Wainer et al. 2010). (2) Simulation interoperability standards to synchronize different simulation environments in the same simulation run. A plug-and-play interoperability standard would avoid requiring legacy systems software implementation changes, fast and safe support. In our case, we proposed a lightweight standard based on RESTful Web-services for both conservative and optimistic distributed simulation. (3) Across-enterprise collaboration is needed to bridge simulation assets via distributed simulation middleware. We propose the use of workflows to accomplish this, plugging simulation into a formal Business Process Management (BPM) plan.

A product workflow defines the steps for developing a product until it gets into market (Weske 2007). The benefits of using workflows are said to be manifold: they serve as means for controlling processes, as information source for managers and quality control. They also provide features like an increased "truck rate", repeatability, integration of different experts sitting at remote locations, etc. Workflows are operation guidelines or process descriptions. The latter is the base for the process-oriented notion of quality, which makes sense for M&S as well. Taking the long history of process descriptions, using workflows for M&S is not a new idea. Meanwhile, there are now some technical solutions for workflow support and workflow management systems for scientific processes have been created, including software products like Kepler (Kepler 2010) or Trident (Trident 2010). In addition, the importance of the explicit workflows for concrete tasks is now evident for researchers, as repeatability is still among the major concerns if scientific results shall be reliable. Nevertheless, there is still no deep workflow integration in M&S software products currently in use. Users are still forced to combine a number of different software products using the more general workflow tools for scientific processes, to use scripting languages or to combine different functionalities manually (in a software or cross products).

We thus present the design of a workflow component that manages and executes workflow patterns (mainly simulation experiment patterns). It uses multiple repositories to allow the reuse of workflow patterns and simulation models, which in turn speeds up the construction of simulation models and experiments, while keeping the cost and error occurrences low. The workflow component follows the reference model recommended by the Workflow Management Coalition (WMC 2009). In this model, a workflow component is in the center and interacts with other surrounding repositories, servers and other workflow components. In our case, the workflow component uses, as a client, the RISE middleware (Al-Zoubi et al. 2009; Wainer et al. 2010) to create and manipulate simulation resources (URIs). RISE becomes a simulation repository manipulated by workflows, allowing users to access those resources from any Web client or workflow components. This is because the workflow component is a client application while simulation resources at the middleware side are URIs similar to any other URIs on the Web. Performing simulation through the RISE middleware prevent workflows of being specific to a citrating simulation environment (e.g. DCD++), as this functionality is provided at the RISE middleware.

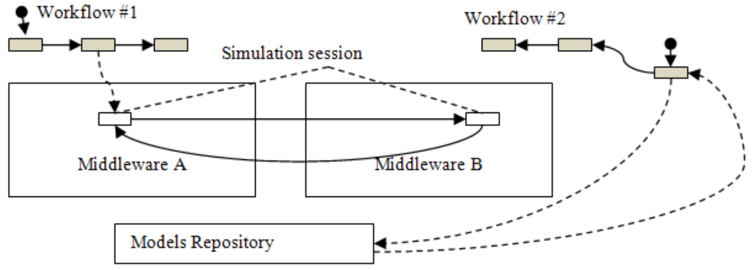


Figure 1: Overview Example for Simulation Workflows

Figure 1 shows an example of two workflows. The first one invokes a distributed simulation session between two middleware instances. In this case, Middleware A invokes Middleware B in this distributed simulation based on the modeler-partitioning scheme submitted during a workflow step (Al-Zoubi et al. 2009). However, if the workflow is using a different service, the workflow will then submit the necessary scripts accordingly. The second workflow in Figure 1, shows the workflow at a step of retrieving information from models repository. For example, it is bringing a simulation model from a remote location to be integrated with a local model. Note that both workflow instances could be the same, but being used separately.

Other research and commercial workflow applications target specific modeling and simulation environments (Isight 2009; Zacharewicz et al. 2008). In contrast, our component is intended to be used across simulation environments through the use of the plug-and-play RISE interoperability middleware. RISE can support different simulation engines (e.g. DCD++) or interoperate different heterogeneous simulation environments and it is the first existing workflow component to use RESTful Web-services, which allows to mashup Web 2.0 applications (O'Reilly 2009) as part of the workflow. Thus, workflows can interact with any device or application attached to the Web, hence making part of a simulation life cycle and loop. Furthermore, the workflow component can use several middleware instances simultaneously, allowing the component to replicate simulation resources on more than one location, which is helpful for fault-tolerance and workload distribution. This is done without much effort at the workflow component side (which perceives simulation resources at the middleware side as URIs that own and can manipulate while the middleware views the component as a client requesting services).

Our focus here is on the workflow component. We further show the component interaction with middleware to create simulation resources (URIs). We summarize the middleware API in the background section, since it is used by the workflow component. The CD++ simulation environment (Wainer 2009) is used in our examples; however, our discussion is not specific to CD++. CD++ partitions the model among different machines to perform a single simulation run, as in Figure 1 (Al-Zoubi et al. 2009; Wainer et al. 2010).

The rest of the paper is organized as follows. Section 2 presents an overview of YAWL notations (Van et al. 2005) used here, DCD++, and the RISE middleware API. Section 3 describes the workflow component architecture. Section 4 discusses identified simulation workflow patterns using YAWL notations. We further discuss converting notations into XML description so it can be stored and decupled from a specific notation language.

2 BACKGROUND

The main objective of the RESTful Interoperability Simulation Environment (RISE) is to provide a multipurpose online simulation interoperability and mashup middleware with a plug-and-play style. The RESTful-CD++ middleware (Al-Zoubi et al. 2009; Wainer et al. 2010) is decoupled from any supported service regardless of their underlying formalism, theory or implementation.

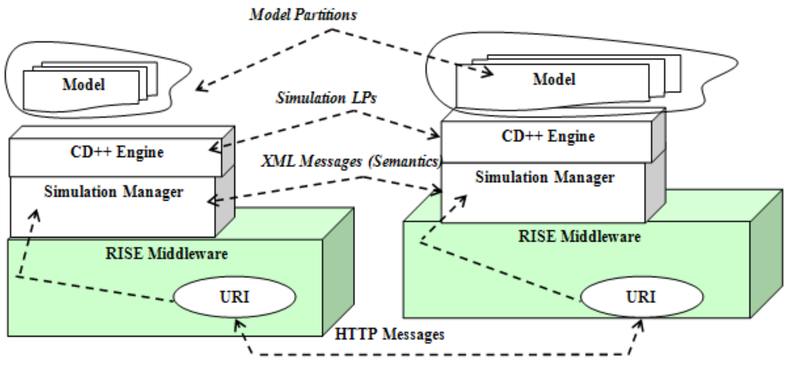
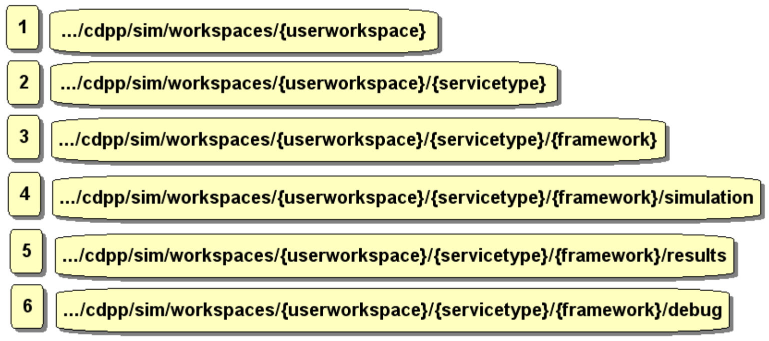


Figure 2: DCD++ Simulation Session via RISE Example

The middleware provides modelers with a flexible experimental framework (i.e. via client applications), allowing modelers to run any number of experiment instances. The experiment settings depend on the service used or services combination. For example, for CD++, the model can run on geographically distributed machines for a single simulation. The model partition is distributed according to an XML configuration document; simulation is synchronized using XML simulation messages, as shown in Figure 2. In this example, the LP is a Distributed CD++ engine (DCD++) responsible for a portion of the entire model. All XML messages are exchanged among logical processors (LP) using each other URIs. Experiment settings and resources (URIs) are persistent and repeatable, unless deliberately removed or updated.

Each experiment is wrapped and manipulated via a set of URIs (i.e. an experiment API), hence allowing their online access from anywhere. The URI API template can be created at runtime. The resource that best matches the request's URI will receive the request and it will become its responsibility to respond to the client. Clients connect to URIs via universal standardized virtual uniform channels. In case of RISE, channels are HTTP channels: GET (to read a resource entirely or partially), PUT (to create new resource or update an existing data), POST (to append new data to a resource), and DELETE (to remove a resource). These URIs are created and manipulated according to the middleware URI template (API), shown in Figure 3.



```
1 .../cdpp/sim/workspaces/{userworkspace}
2 .../cdpp/sim/workspaces/{userworkspace}/{servicetype}
3 .../cdpp/sim/workspaces/{userworkspace}/{servicetype}/{framework}
4 .../cdpp/sim/workspaces/{userworkspace}/{servicetype}/{framework}/simulation
5 .../cdpp/sim/workspaces/{userworkspace}/{servicetype}/{framework}/results
6 .../cdpp/sim/workspaces/{userworkspace}/{servicetype}/{framework}/debug
```

Figure 3: The RISE Middleware API

Line #1 shows the workspaces that hold all clients' simulation services (variables, written within braces { }, are assigned at runtime by clients before a request is sent to the server). Line #2 holds a specific client simulation service (i.e. a simulation engine like CD++) for a specific client. Modelers (clients) interact with a number of simulation-related resources during the simulation of a specific model, as shown in Lines 3-6 in Figure 3; hence, those resources form an experiment pattern from a user viewpoint as follows: (1) The framework resource (Line #3) holds an experiment input data (such as the model source code, simulation input variables, and sub-models interconnections). The POST channel is used to submit files to a framework. PUT is used to create a framework or update simulation configuration settings. DELETE is used to remove a framework. The GET channel is used to retrieve a framework state. (2) A simulation resource (Line #4) wraps an active simulation engine (e.g. CD++), which interacts with other remote simulation, if any. This resource exchanges synchronized messages with other simulation entities (in case of distributed simulation) via the POST channel, and POST can be used by modelers to input variables in order to manipulate simulation at runtime dynamically. The PUT channel is used to create this resource, hence to start simulation. The DELETE channel is used to abort simulation and remove this resource. (3) The results resource (Line #5) holds the simulation output files (if the simulation was completed successfully). The GET channel is used to retrieve results where the DELETE channel is used to remove those results. The PUT and POST channels are disabled for this resource. (4) The debug resource (Line #6) holds model-debugging files. For example, a modeler can print debugging information inside his model source code to be retrieved later via this resource. The GET channel is used to retrieve model-debugging files where the DELETE channel is used to remove those files. PUT and POST channels are disabled for this resource.

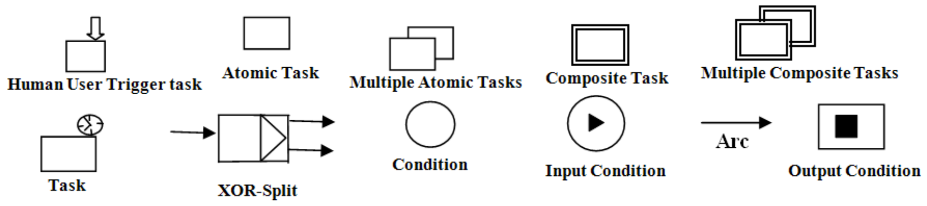


Figure 4: Excerpt of YAWL Notational elements

Figure 4 shows the YAWL notations: conditions are represented by circles and tasks are represented by rectangles. Input and output conditions specify the beginning and the end of a workflow. Atomic tasks are indivisible activities, where composite tasks (denoted by double border) enclose other workflow activities. Tasks are considered automatic, but placing an arrow on a top of a task indicates a user interaction while placing a clock indicates a timer trigger task. XOR-splits limit the flow output to exclusively one, based on a certain condition. Arcs show the token flow direction.

3 MANAGEMENT ARCHITECTURE

We distinguish here between simulation and workflow models. A *simulation model* is a model of a system to be studied by simulation; hence, it defines the necessary scripts or programming source code, which will be executed by a simulation engine. The simulation model may be expressed graphically using known workflow or self defined notations. However, the graphical notations still need to be translated into executable scripts by a simulation engine. The experimental framework at the simulation model level captures the set of circumstances under which a real system is observed and subjected to experimentation; see (Chreyh et al. 2009). On the other hand, a workflow model defines the blueprint of the set of activities that are executed and in which order, with the goal to achieve a single business goal. For example, tracking the evolving steps of a record in an organization is a workflow. For example, one must execute the "ship-product" task, if the condition "received payment" is satisfied. These workflow steps may be fully automated without human intervention. Thus, a simulation model execution might be a step plugged in a workflow model blueprint; hence, it is executed once the workflow token enters this task. Further, there are usually multiple instances of a workflow existing simultaneously to realize different situations. In our case, we fully automate the simulation experimental workflow blueprint (see Section 4.1) to execute different simulation experiments regardless of used simulation model or simulation engines. Figure 5 shows the overall design of the management architecture of the workflow component.

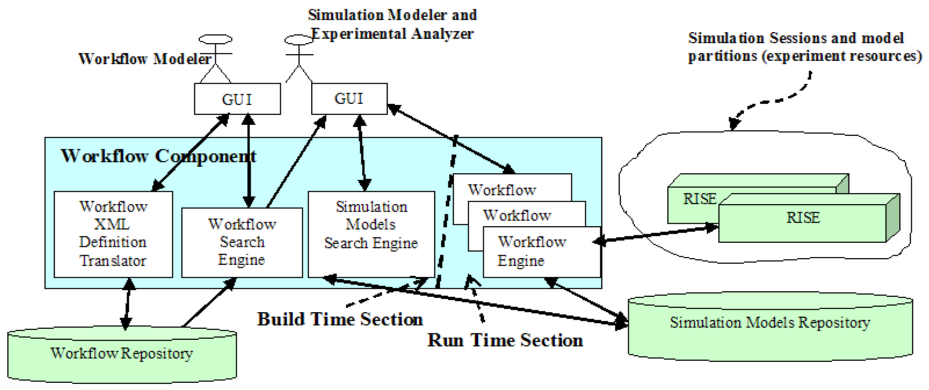

Simulation Modeler and Experimental Analyzer
Workflow Modeler
Simulation Sessions and model partitions (experiment resources)
GUI
GUI
Workflow Component
Workflow XML Definition Translator
Workflow Search Engine
Simulation Models Search Engine
Workflow
Workflow
Workflow Engine
RISE
RISE
Build Time Section
Run Time Section
Workflow Repository
Simulation Models Repository

Figure 5: Workflow Component Architecture

The workflow component architecture, shown in Figure 5, follows the reference model recommended by Workflow Management Coalition standard group (WMC 2009). In this model, a workflow component is in the center and interacts with other surrounding repositories, servers and perhaps other workflow components. A Graphical User Interface (GUI) is used to define the workflows graphically according to certain notations (in our case, YAWL). The workflow modeler's role is to design those workflow blue-prints. Once completed, they are compiled by the workflow component into an XML Definition script (see Section 4.1). The workflow components allow both workflow and simulation modelers to submit their search criteria to retrieve certain workflow patterns, as shown in Figure 5. The simulation modelers use workflow patterns, for example, in the context of simulation model analysis; and experiments, as in the case of the patterns presented in Section 4. On the other hand, workflow modelers use workflow pat-terns to construct other needed workflow patterns. For example, the CD++ models repository (Chreyh et al. 2009), an internet based searchable engine database for CD++ models.

In the *Run Time* phase, workflow engines execute the workflow patterns. Each workflow engine is an independent thread to process a workflow pattern instance. The workflow engine parses the workflow XML definition document and executes each step as applicable, manipulating simulation resources at the server(s) side as part of the simulation workflow process. The RISE servers are viewed as repositories of simulation resources (URIs) that are manipulated at run time. During workflow execution, the workflow engine moves the token through tasks, conditions and elements. The token time is spent inside tasks, since they represent the actual work in a workflow.

The execution semantics of each task in a workflow is expressed in a state transition diagram shown in Figure 6. The diagram consists of conditions (represented by circles) and transitions (represented by rectangles). Conditions are places to hold tokens (a token is an instance of a task), hence multiple in-stances reside in different condition of their task state transition diagram. Transitions are actions that cause a token to move from a condition to another. Transitions take time to complete. Figure 6 shows five possible conditions: (1) Waiting, which holds tokens that are not allowed to run. (2) Ready, which holds tokens that are active and waiting to execute. (3) Running, which holds tokens that are currently being ex-ecuted. (4) Completed, which holds tokens that have completed. (5) Cancellation, which holds tokens that have been canceled before completion.

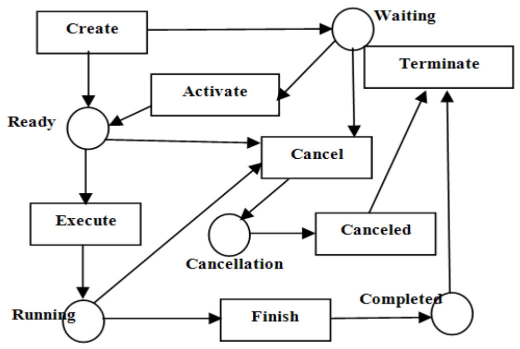


Figure 6: State Transition Diagram for a Workflow Task

The workflow component tracks token instances of a task via associating them to lists according to their defined conditions in the state transition diagram. In this case, based on the possible condition, there are four token lists: Waiting, Ready, Running and Completed lists, shown in Figure 7.

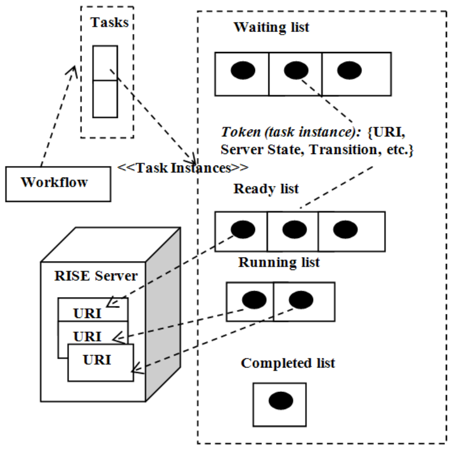


Figure 7: Lists for multiple instances (Tokens) of Tasks

4 EXPERIMENTAL PATTERNS

This section introduces a simulation workflow pattern, which provides automation and flexibility in the way simulation experiments are conducted. These patterns are typically stored as XML document (discussed in Section 4.1) in a workflow repository that can be executed by a workflow engine (Figure 5). On the other hand, this workflow component is independent of any specific workflow patterns, allowing more workflow patterns to be stored in the repository, to be retrieved and executed by the workflow component.

The Simulation-Workflow composite task, shown in Figure 8, encapsulates the entire simulation workflow pattern, allowing it to be plugged into another business process workflow. Multiple Simulation-Workflow task instances may exist simultaneously, allowing different simulation experiments to be executed simultaneously. In this case, each instance runs as an independent thread from other instances where tokens are managed according to the state transition diagram shown in Figure 6. The simulation-workflow task (shown in Figure 8) starts with executing the Simulation-Experimentation task (discussed in section 4.1) to run the defined experiment for one or more instances, provided a simulation model is present and validated. Atomic task Analysis handles the output results of all experiment instances as defined by the modeler. For example, the analysis may be performed via visualization or by comparing certain variables from experiment instances different simulation results. Thus, this task is put outside the Simulation-Experimentation task since it may require extra handling from modelers.

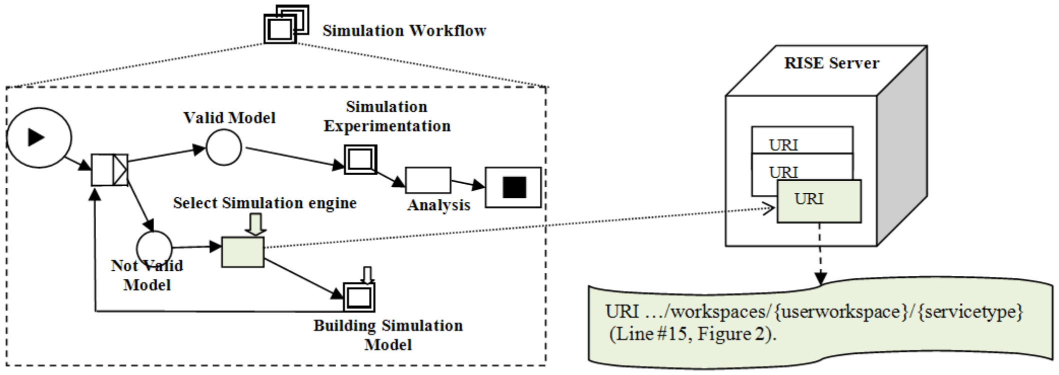

Simulation Workflow
Valid Model
Simulation Experimentation
Analysis
Select Simulation engine
Not Valid Model
Building Simulation Model
RISE Server
URI
URI
URI
URI …/workspaces/{userworkspace}/{servicetype} (Line #15, Figure 2).

Figure 8: Simulation Workflow Pattern Overview and Middleware Interactions

If a simulation model is present, the Simulation-engine-selection atomic task is initiated by the user to select a simulation engine that is capable of executing the simulation model under study. This "Execute" transition (Figure 6) sends the request to RISE to create the URI for the simulation engine, as shown in Figure 8. In this case, the simulation engine type is mapped to variable {servicetype} in the RISE API (Figure 3). For example, Figure 8 shows that the task has created URI *"…/Bob/DCDpp*", which indicates that DCD++ will be used by experiments belonging to workspace *"Bob"*. The workflow then starts the Building-Simulation-Model composite task (discussed in Section 4.2), which concerns about the control flow of building a simulation model in order to be later executed by a simulation engine. Therefore, model testing, validation and construction are part of this task. This task is partially automated since user involvement is needed at certain steps.

4.1 Simulation Experimentation Task

The composite task pattern provides an automotive method of conducting multiple experiment instances simultaneously. In other words, this pattern allows modelers to conduct the common steps for any number of instances or input variables automatically without the need of doing it manually one by one. Of course, performing steps manually is an error-prone and time consuming. The task workflow in YAWL notation is shown in Figure 11, which contains a multiple instances of the Experiment-Instances composite task. Each experiment instance corresponds to a URI at RISE side, which can be created, updated, read or deleted during the workflow execution. Figure 9 shows the interaction between the internal sub-tasks of an experiment instance (Figure 11) and RISE middleware. Figure 9 shows messages in terms of HTTP channel, URI and message format whereas RISE side is shown in terms of its URI template API (see Figure 3), hence as seen by the workflow component. RISE is discussed in (Al-Zoubi et al. 2009).

The Experiment-Instances task (Figure 11) starts with making the decision to reload experiment configuration or not to reload. If reloading configuration is required, the workflow follows the following path: (1) Get-ID atomic task is executed to assign a new ID for the experiment instance, if this is the first time execution. The ID needs to be unique within the same experiment scope; hence, it is implemented as counter (e.g. static variable in C++/Java class), which is incremented with every instance creation. This ID is needed to assign unique URI (name) for each experiment instance at the server side.

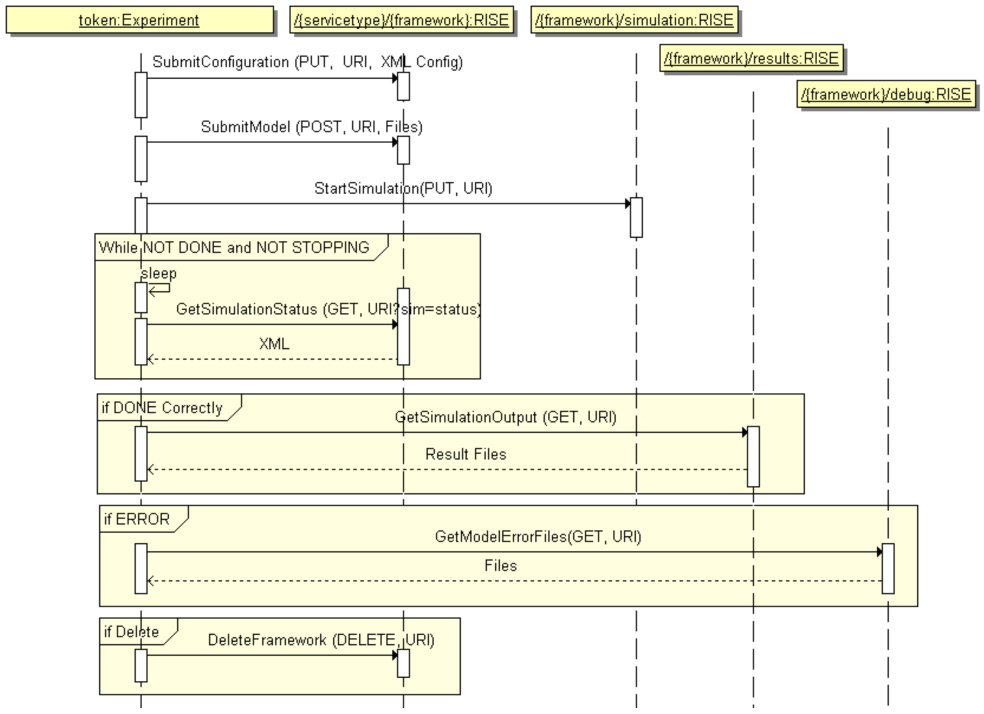

token:Experiment
/{servicetype}/{framework}:RISE
/{framework}/simulation:RISE
/{framework}/results:RISE
/{framework}/debug:RISE
SubmitConfiguration (PUT, URI, XML Config)
SubmitModel (POST, URI, Files)
StartSimulation(PUT, URI)
While NOT DONE and NOT STOPPING
sleep
GetSimulationStatus (GET, URI?sim=status)
XML
if DONE Correctly
GetSimulationOutput (GET, URI)
Result Files
if ERROR
GetModelErrorFiles(GET, URI)
Files
if Delete
DeleteFramework (DELETE, URI)

Figure 9: Experiment Instance and RISE Interaction

For example, URIs …/fireModel_1 and …/fireModel_2 are different instances URIs for the same experiment. This is similar to having different C++/Java object names for the same class. (2) Submit-Configuration and Submit-Model atomic tasks are executed. Note that when DCD++ is used (i.e. service-type = DCDpp), task configuration is an XML document that partitions a model among different servers. For example, assume a DEVS coupled model consists of producer and consumer atomic models. In this case, the modelers may place the producer model on a server while place the consumer on a different server. This XML document is sent via PUT channel to the appropriate framework (i.e. experiment) URI at RISE side, as shown in Figure 9. Distributed simulation is discussed in details in (Al-Zoubi et al. 2009), but our focus here is on the workflow component side due to the allowed space here.

The workflow executes task Start-Simulation, if there is no need to reconfigure the experiment instance token or if the token is exited the Submit-Model-and-Configuration task. The Start-Simulation task appends "/simulation" to the instance URI (e.g. …/friremodel_1/simulation) and sends the request to this URI via PUT channel. This requests creates the resource, hence starts the simulation. At this point, the main RISE middleware creates the necessary URIs on other servers (i.e. other RISE instances) and starts the simulation on those servers, as described in (Al-Zoubi et al. 2009). At the workflow component, the token enters a loop to keep checking on the simulation status. In this loop, the token executes the Sleep atomic task to sleep for some specified time, and then enters atomic task Get-Simulation-Status. This task appends query variable "sim=status" to the experiment instance URI and sends the request via GET channel (Figure 9). In this case, the middleware returns an XML message similar to the following: *<Simulation><Status>DONE</Status></Simulation>*. If the simulation is neither in the DONE nor in the STOPPING state, the token will then reenter the Sleep atomic task. Otherwise, it exits this loop on ERROR or DONE condition. Note that this loop will always exit even in case of server failure. In this case, a communication error is generated. The Get-Model-Error-Files atomic task is executed, if simulation exited on ERROR condition. On the other hand, in the normal circumstances the simulation completes successfully, which places the token in the Get-Simulation-Output atomic task. Finally, the workflow decides (as configured by the user) to remove the experiment instance or not remove it. The Delete-and-Release-ID atomic task removes the experiment instance at the server side by sending the request via channel DELETE to its URI. On the other hand, the workflow component maintains the token in a special queue, if the modeler plans to repeat the experiment instance; hence, avoiding going through initialization phase at the next run time.

The RISE URIs are persistent, hence they will always exist unless removed by their owners. Thus, experiment instances (tokens) URIs still exist at the server side even in case of RISE or workflow component shutdown. In this case, the workflow component can use the RISE help to retrieve these tokens. For example, a GET request to URI "…/workspaces/Bob/DCDpp" (see Lines #14 and #15 in Figure 3) will return a list of all of the experiment instances URIs of workspace "Bob" and uses DCD++ simulation engine, in a similar way to browsing a Web site. Thus, tokens can always be brought to life correctly.

It is common practice to represent graphical notations in scripts so that it can be stored locally in repositories and allowing it to be independent of specific graphical notations. This is the idea behind the XML Process Definition Language XPDL standardized by the Workflow Management Coalition (WMC) group (WMC 2009). In this case, the software needs to convert a graphical notation into the required definition. However, XPDL is complicated to meet our requirements such as defining middleware RESTful interactions and HTTP channels. Extending XPDL via profiling would add more complexity and defeat the main purpose of interoperating with other workflow components, since we would define new extended XPDL. Therefore, we propose a simple XML language to represent the workflows targeting RESTful interoperability principles, as shown in Figure 10.

```
<Task>
<Name>Simulation-Experimentation</Name><Type>Composite</Type>
<Input><Parm1>uri</Parm1><Parm2>ModelPath</Parm2></Input>
<Elements>
 <Splits><XOR><Instances><Instance>XOR-1</Instance>    </Instances></XOR></Split>
 <Conditions><Condition>COND-Configure</Condition>   </Conditions>
 <Tasks>

  <Task><Name>Submit-Configuration</Name>
    <Instances><Instance>
     <Name>Submit-Configuration-1</Name>
     <Input><Parm1>uri</Parm1><Parm2>ModelPath</Parm2></Input>
    </Instance></Instances></Task>
   <Task><Name>Start-Simulation-1</Name>
    <Instances><Instance>
    <Input><Parm1>uri+ /simulation </Parm1></Input>
   </Instance></Instances></Task>

 </Tasks>
</Elements>
<Transitions>

 <Transition><Id>xxx</Id><From>XOR-1</From><To>COND-Configure</To>
 <Transition><Id>xxx</Id><From>COND-Configure</From><To>Get-ID-1</To>
 <Transition><Id>xxx</Id><From>Get-ID-1</From><To>Submit-Configuration-1</To>
 <Transition><Id>xxx</Id><From>Submit-Configuration-1</From><To>Submit-Model-1</To>

</Transitions>

<!------------------------------------------->
<!--Task Submit Configuration -->
<!------------------------------------------->
<Task>
<Name>Submit-Configuration</Name>
<Type>Atomic</Type>
<Input><Parm1>uri</Parm1><Parm2>ModelPath</Parm2></Input>
<Implementation><Type>REST</REST>
 <Operation>
  <Channel>PUT</Channel>
  <Representation><Type>text/xml</Type><File>ModelPath</File></Representation>
 </Operation>
</Implementation>
</Task>

<!----------------------------------------->
<!--Task Start Simulation
```

Figure 10: Excerpt of XML Definition of Simulation-Experimentation Task Workflow (see Figure 11)

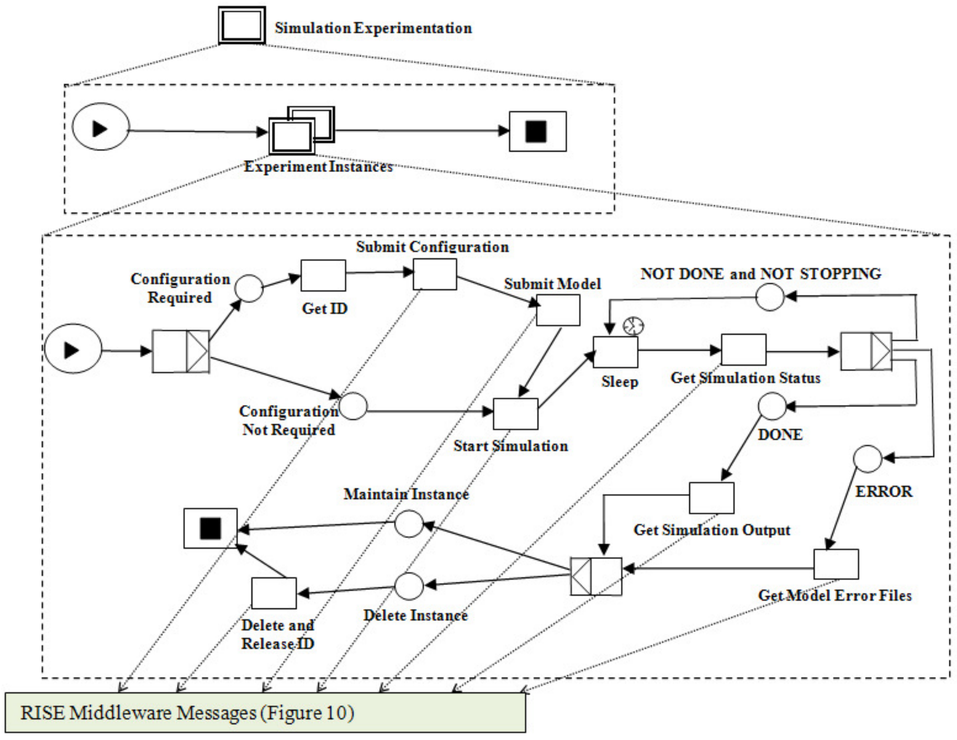


Figure 11: Simulation Experimentation Task Workflow

Figure 10 is the XML definition for YAWL graphical notations shown in Figure 11. Line #2 defines the task name and its type. Line #3 defines the input parameters of this task. In this case, an instance is initialized with the experiment instance URI and the path of the files that contain all of the simulation model required information, typically zipped up in a single file. Lines #4-20 define all enclosed element instances by this composite task. Line #5 defines a portion of split instances in the task (e.g. XOR-1). Line #6 defines the task-enclosed conditions. Lines 7-19 define internal task instances. In this case, Lines #9-13 creates an instance of atomic task Submit-Configuration with name Submit-Configuration-1. It further initializes it with the framework URI and simulation framework configuration (i.e. XML document). Line #14-17 creates an instance of task Start-Simulation, and initializes it with the experiment instance URI, appending to it "/simulation". Lines #21-28 define all internal transitions, hence how each internal element instance is connected to other instances. The rest of the XML document defines all internal atomic tasks to the Simulation-experimentation task. Lines #33-43 presents the Submit-Configuration task template. Line #36 defines the input parameters. Lines #37-42 defines the task implementation as of type of REST Web-service that uses channel PUT where only XML type of representation is supported. In the same way, the Start-Simulation atomic is defined in Lines #48-58.

4.2 Building-Simulation-Model Task

The Building-Simulation-Model composite task concerns of constructing a simulation model for studying a system behavior via simulation. Therefore, the experiment scope is the system understudy itself. On the other hand, the simulation-experimentation task discussed in Section 4.1 concerns about executing multiple experiment instances of different/same simulation experiments in certain framework. As a result, this task uses the simulation-experimentation task as part of model testing and validation. The workflow pattern offered by the Building-Simulation-Model task can be viewed as wizard workflow for modelers, since a number of tasks require human initiation to be executed. Further, multiple modelers can work on separate workflow instances to construct a portion of the final model where eventually several portions are integrated in one final product.

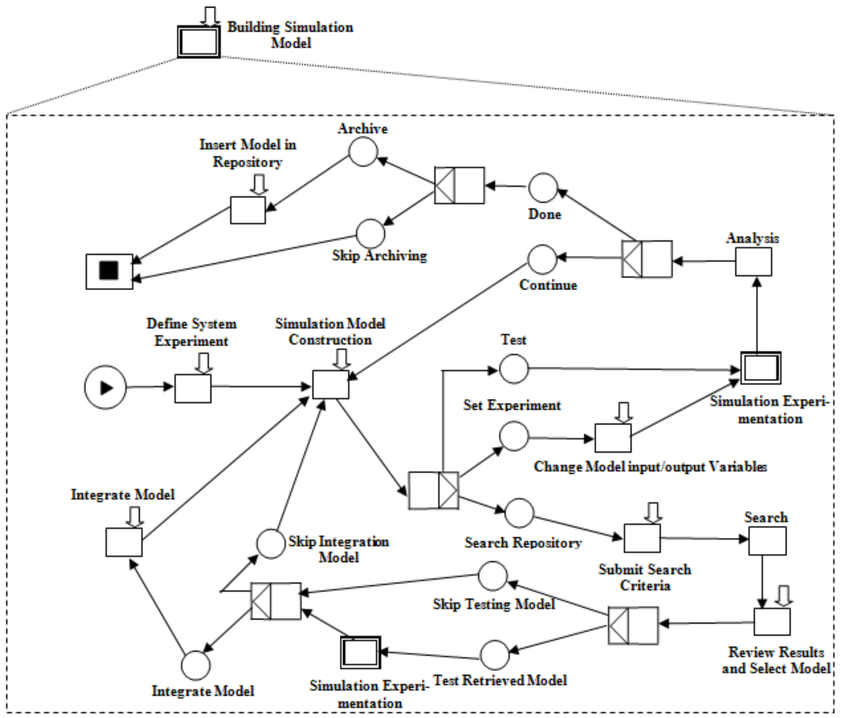


Figure 12: Building-Simulation-Model Composite Task Workflow

The Building-Simulation-Model (see Figure 12) task starts with task Define-System-Experiment to define the purpose of the simulation model with respect to the system understudy. Perhaps, at this step different modelers are assigned different portion of the expected final model. The workflow then moves to Simulation-Model-Construction task, which is the actual building of the model. The model might be built using graphic notations or by directly writing source code and scripts. In early stages, modelers are expected to search the simulation model repository so that they can re-use any existing models, speeding up the construction process. In this case, the workflow stems to enable the modeler to submit the search criteria and retrieve already existing models. The modeler can then choose to test a retrieved model by invoking the simulation-experimentation task, discussed in Section 4.1. At this point, the modeler chooses to integrate or not to integrate the retrieved model with the model under construction. Either choice, the workflow returns to the Simulation-Model-Construction task. The workflow allows the modeler to start testing the model under construction via invoking the simulation-experimentation task (Section 4.1). The modeler can update the model input/output parameters before executing the model. The modeler can choose to continue constructing the model upon completing model execution analysis and returns to the Simulation-Model-Construction task. Otherwise, the construction is terminated with the choice to upload the final model into the repository, allowing the model to be reused in future development.

5 CONCLUSIONS

Certain needed features have been identified as needed in the area of distributed simulation to overcome its challenges, high cost, and limited use in the industry, as identified by a number of surveys such as (Boer et al. 2008) (Strassburger et al. 2008). We have developed the RISE middleware (Al-Zoubi et al. 2009) to overcome some of those challenges such as plug-and-play interoperability and dynamicity. The middleware relays on the RESTful Web-service principles to achieve its interoperability comparing to other distributed simulation existing approaches (Wainer et al. 2010). The simulation framework URI template of the RISE API is a template for any type of experiment regardless of the simulation engine used, or the simulation model under study. Thus, the RISE already provides to the workflow component an extremely flexible way of creating different instances of different experiments (e.g., a DCD++ distributed simulation session). On the other hand, to interoperate heterogeneous simulation assets across-organization, we to include simulation assets in a formal BPM plan to allow practical across-enterprise collaboration. We showed here how the workflow component can execute workflows across geographical locations via the RISE middleware. Using RISE enables the workflow component to be independent of any specific simulation environment, and able to combine different heterogeneous simulation environments in the same workflow. Therefore, the RISE can provide workflow engines with (1) interoperability layer to compose different simulation workflows across organizations (regardless of specific simulation environments) or Web 2.0 services, (2) the ability to execute simulation workflows over multiple servers, which, in turn, is a way to provide simulation experiments with fault-tolerance and workload balance. Indeed, this allows running experiment instances already running on a server simultaneously on another, as a backup. Further, instances of an experiment can be spread over multiple servers when an experiment is required to be replicated a number of times, relieving a single server of handling all of load.

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.

Boer C., A. Bruin, and A. Verbraeck. 2009. “A survey on distributed simulation in industry”. Journal of Simulation. 3(1):3-16.

Chreyh, R., and G. Wainer. 2009. CD++ Repository: An Internet Based Searchable Database of DEVS Models and Their Experimental Frames. In *Proceedings of the 2009 Spring Simulation Conference*. San Diego, CA.

Kepler. <https://kepler-project.org/> [accessed March 15, 2010].

O'Reilly T. "What Is Web 2.0". < http://oreilly.com/web2/archive/what-is-web-20.html> [accessed May 01, 2009].

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.

Trident. <http://connect.microsoft.com/Trident> [accessed March 15, 2010].

Isight. <http://www.simulia.com/products/> [accessed October 10, 2009].

Van der Aalst W., ter Hofstede A. "YAWL: Yet Another Workflow Language". Information Systems. Vol. 30, No. 4, pp. 245-275. June 2005.

Wainer, G. and Al-Zoubi, K. 2010. *An Introduction to Distributed Simulation*. Chapter 11 in book *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains* ed. Catherine B., and J. Sokolowski. 2010. New Jersey: Wiley.

Wainer, G. 2009. *Discrete-Event Modeling and Simulation: A Practitioner's Approach*. Boca Raton, Florida: CRC press, Taylor & Francis Group.

Weske M. 2007. *Business Process Management: Concepts, Languages, Architectures*. New York: Springer.

WMC (Workflow Management Coalition). <http://www.wfmc.org/> [accessed October 10, 2009].

Zacharewicz G., Frydman C., Giambiasi N. "G-DEVS/HLA Environment for Distributed Simulations of Workflows". SIMULATION. Vol. 84, No. 5, pp. 197-213. 2008.

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, 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 . He is author of three books and over 220 research articles, edited four other books, and helped organizing over 100 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 Vice President Publications of SCS, Special Issues Editor of the Transactions of the SCS, and Associate Editor of Wireless Networks and 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 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>.