[Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains], Edited by [John A. Sokolowski, Catherine M. Banks].
ISBN 0-471-XXXXX-X Copyright © 2009 Wiley [Imprint], Inc.

Chapter xx

An Introduction to Distributed Simulation

Gabriel A. Wainer ***Khaldoon Al-Zoubi***

Introduction

Distributed simulation technologies were created to execute simulations on distributed computer systems (i.e., on multiple processors connected via communication networks) [23]. Distributed Simulation is a computer program that models real or imagined systems over time. On the other hand, distributed computer systems interconnect various computers (e.g. personal computers) across a communication network. Therefore, distributed simulation deals with executing simulation correctly over inter-connected multiple processors. Correctness means that the simulation should produce the same results as if it was executed sequentially using single processor. Fujimoto [23] distinguishes parallel from distributed simulation by their physical existence, used processors, communication network and latency. Parallel systems usually exist in a machine room, employ homogeneous processors and communication latency is measured with less than 100 microseconds. In contrast, distributed computers can expand from a single building to global networks, often employ heterogeneous processors (and software), and communication latency is measured with hundreds of microseconds to seconds. The simulation is divided spatially (or temporally) and mapped to participated processors. Our focus here is on distributed simulation, which employs multiple distributed computers to execute the same simulation run over a wide geographical area.

A focus of distributed simulation software has been on how to achieve model reuse via interoperation of heterogeneous simulation components. Other benefits include reducing execution time, connecting geographically distributed simulation components (without relocating people/equipment to other locations), interoperating different vendor simulation toolkits, providing fault tolerance and information hiding – including the protection of intellectual property rights - [6] [23].

Trends and Challenges of Distributed Simulation

The defense sector is currently one of the largest users of distributed simulation technology. On the other hand, the current adoption of distributed simulation in the industry is still limited. In recent years, there have been some studies (conducted in the form of surveys) to analyze these issues [5][6][56]. The surveys collected opinions, comments and interviews of experts from different background in the form of questionnaires, and showed that there is now an opportunity for distributed simulation in industry. It has been predicted that in the coming years, the sectors that will drive future advancement in distributed simulation are not only the defense sector, but also gaming industry, the high-tech industry (e.g. auto, manufacturing and working training), emergency and security management [56].

The High Level Architecture (HLA) is the preferred middleware standard in the defense sector [28]. However, its popularity in industry is limited. The HLA started as a large project mainly funded by the military, in order to provide the means for reusing legacy simulations in military training operations, so that exercised could be conducted between remote parties in different fields, reusing existing simulation assets. On the other hand, the adoption of these technologies in industry is based on return-of-investment policies. Therefore, most commercial-off-the-shelf (COTS) simulation packages do not usually support distributed simulation due to a cost/benefit issue. In [6], the authors suggested that, in order to make distributed simulation more attractive to the industrial community, we need a lightweight COTS-based architecture with higher cost/benefit ratio. The middleware should be easy to understand (e.g. programming interface, fast development and debugging), and interoperable with other vendor's simulation components. Distributed simulation might become a necessity when extending the product development beyond factory walls, particularly when such organizations prefer to hide detailed information [25]. New standards (for instance, Commercial Off-the-Shelf Simulation Package Interoperability, Core Manufacturing Simulation Data and DEVS) can contribute to achieve these goals [62].

Another recent study, carried out by Strassburger, Schulze and Fujimoto, focused on surveying experts from the area of distributed simulation and distributed virtual environment [56]. This study found out that the highest rated applications in future distributed simulation efforts include the integration of heterogeneous resources, and joining computer resources for complex simulations and training sessions. The study also identified some research challenges:

- Plug-and-Play capability: the middleware should be able to support coupling simulation models in such a way that the technical approach and standards gain acceptance in industry. In other words, interoperability should be achieved effortlessly.
- Automated semantic interoperability between domains: to achieve the plug-and-play challenge, interoperability must be achieved at the semantic level.

A Brief History of Distributed Simulation

Simulations have been used for war games by the U.S Department of Defense (DoD) since 1950s. However, until 1980s, simulators were developed as a standalone with single-task purpose (such as landing on the deck of an aircraft carrier). Those standalone simulators were extremely expensive comparing to the systems that they suppose to mimic. For example, the cost of a tank simulator in the 1970s was $18 million, while the cost of an advanced aircraft was around $18 million (and a tank was significantly less). By the 1980s, the need of performing cost-effective distributed simulation started to be used at the DoD to simulate war games [38].

The first large project in this area, the SIMulator NETworking program (SIMNET) was initiated in 1983 by the Defense Advanced Research Projects Agency (DARPA) in order to provide virtual world environment for military training [10][38][58][59]. SIMNET was different from previous simulators in a sense that many objects played together in the same virtual war-game. During a SIMNET exercise, a simulator sent/received messages to/from other simulators using a Local Area Network (LAN). This distributed simulation environment enabled various simulation components to interact with each other over the communication network. Cost played as a major factor for developing SIMNET. However, the ability of having different type of simulations interacting with each other was another major factor. For example, warships, tanks, and aircraft simulators worked together, enhancing the individual systems' ability to interact with others in a real-world scenario. Further, the design of SIMNET was different from previous simulators. The goal became to derive the simulation requirements, and only then decide the hardware needed for the simulation environment. This caused many required hardware in the actual systems to be rolled out from simulation training.

The success of SIMNET led to developing standards for Distributed Interactive Simulation (DIS) during the 1990s [22][27][53]. DIS is an open standard (realized in [31][32][33][34]) for conducting interactive and distributed simulations mainly within military organizations. DIS evolved from SIMNET and applied many of SIMNET's basic concepts. Therefore, DIS can be viewed as a standardized version of SIMNET. The DIS standards introduced the concept of interoperability in distributed simulation, meaning that one can interface a simulator with other compliant DIS simulators, if they follow the DIS standards. Interoperability via simulation standards was a major step forward provided by SIMNET, but it only permitted distributed simulations in homogeneous environments. DIS was aimed to provide consistency (from human observation and behavior) in an interactive simulation composed by different connected components. Consistency in these human-in-the-loop simulators was achieved via data exchange protocols and a common database. DIS exchanged data using standardized units called the Protocol Data Unit (PDU), which allowed DIS simulations to be independent of the network protocol used to transmit those PDUs [27]. DIS was successful in providing distributed simulation in LANs, but it only supported interactive simulations restricted to military training [56]. These simulations did not scale well in Wide Area Networks.

SIMNET and DIS use an approach in which a single virtual environment is created by a number of interacting simulations, each of which controls the local objects and communicates its state to other simulations. This approach led to new methods for integrating existing simulations into a single environment, and during the 1990s, the Aggregate Level Simulation Protocol (ALSP) was built. ALSP was designed to allow legacy military simulations to interact with each other over LANs and WANs. ALSP, for example, enabled Army, Air Force and Navy war game simulations to be integrated in single exercise [3][21] [63].

The next major progress in the defense simulation community occurred in 1996 with the development of the High Level Architecture (HLA) [28][29][30]. HLA was a major improvement because it combined both analytic simulations with virtual environment technologies in single framework [23]. The HLA replaced SIMNET and DIS, and all simulations in DoD are required to be HLA compliant since 1999 [23].

The distributed simulation success in the defense community along with the popularity of the Internet in the early 1990s led to the emergence of non-military distributed virtual environments, for instance, the Distributed Interactive Virtual Environment (DIVE) (which still in use since 1991). DIVE allows a number of users to interact with each other in a virtual world [19]. The central feature in DIVE is the shared, distributed database where all interactions occur through this common medium.

Another environment that became popular during 1990s was the Common Object Request Broker Architecture (CORBA) [26]. CORBA introduced new interoperability improvements since it was independent of the programming language used. On the other hand, CORBA use had sharply declined in new projects since 2000. Some reflect this for being very complicated to developers or by the process CORBA standard was created (e.g., the process did not require a reference implementation for a standard before being adopted). Further, Web Services became popular in the 2000's as an alternative approach to achieve interoperability among heterogeneous applications (which also contributed to CORBA's decline).

Web Services standards were fully finalized in 2000. However, TCP/IP, HTTP, and XML (which are the major Web Services standards), had matured since the 1990s. These standards have opened the way for Simple Object Access Protocol (SOAP) version 1.0 [39], which was developed in 1999 by Userland Software and Microsoft. SOAP provided a common language (based on XML) to interface different applications. A major breakthrough came when IBM backed up the SOAP proposal in early 2000 and joined the effort for producing the SOAP standard version 1.1 [8]. It was followed in the same year by the definition of the standards version 1.0 of the Web Services Description Language (WSDL), which is used to describe exposed services [14]. The final boost for making Web Services popular came when five major companies (Sun, IBM, Oracle, HP and Microsoft) announced their support for Web Services in their products in 2000. It did not take long for the distributed simulation community to take advantage of Web Services technology. Web Services are being now even used to wrap the HLA interfaces to overcome its interoperability shortcomings, or to perform pure distributed simulation across WAN/Internet. Web Services presented the Service Oriented Architecture (SOA) concept, which means services are deployed in interoperable units in order to be consumed by other applications. The US DoD Global Information Grid (GIG) is based on SOA to interoperate the DoD heterogeneous systems. At the time of this writing, Web Service is the technology of choice for interoperating heterogeneous systems across the Internet [67].

Synchronization Algorithms for Parallel and Distributed Simulation

Parallel/distributed simulations are typically composed of a number of sequential simulations where each is responsible of part of the entire model. In parallel and distributed simulations, the execution of a system is subdivided in smaller, simpler parts that run on different processors or nodes. Each of these subparts is a sequential simulation, which is usually referred to as a logical process (LP). Those LPs interact with each other using message passing to notify each other of a simulation event. In other words, LPs use messages to coordinate the entire simulation [23]. The main purpose of synchronization algorithms is to produce the same results as if the simulation were preformed sequentially in a single processor. The second purpose is to optimize the simulation speed by executing the simulation as fast as possible.

In order to reduce the execution times, the parallel simulator tries to execute events received on different LPs concurrently (in order to exploit parallelism). Nevertheless, this might cause errors in a simulation. Consider the scenario presented in Figure 1, in which two LPs are processing different events. Consider that the LPs receive two events: E_{200} is received by LP_2 (with timestamp 200), and event E_{300} is received by LP_1 (with timestamp 300). Suppose that LP_2 has no events before time 200, and LP_1 has no events before time 300. It thus seems reasonable to process E_{200} and E_{300}. Suppose that, when we execute E_{200}, it generates a new event, E_{250} (with timestamp 250), which must be sent to LP_1. When LP_1 receives the event E_{250}, it was already processing (or had processed) the event E_{300} with timestamp 300. As we can see, we receive an event from the past in the future (an event that requires immediate attention, and might affect the results of processing event E_{300}). This is called a *causality error*.

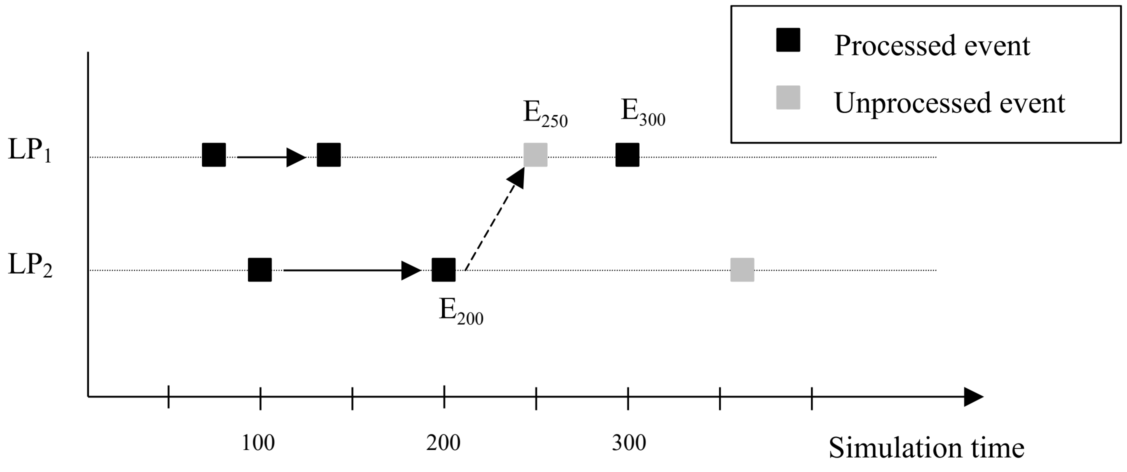


Figure 1. Causality error in a distributed simulation

The *local causality constraint* guarantees the conditions under which one cannot have causality errors [23]: if each LP processes events and messages in non-decreasing timestamp order, causality errors cannot occur. This brings us to a fundamental issue in synchronization algorithms: should we avoid or deal with local causality constraints? Based on these ideas, two kinds of algorithms were defined. *Conservative* (pessimistic) algorithms avoid local causality errors by carefully executing safe events, not permitting local causality errors. On the other hand, *Optimistic* algorithms allow causality errors to occur, but fix them when detected. It is difficult to decide which type is better than the other one, because simulation is application dependent. In fact, the support for both types of algorithms may exist within one system.

Conservative Algorithms

Conservative algorithms were introduced in late 1970s by Chandy-Misra [12] and Bryant [9]. This approach always satisfies local causality constraint via ensuring safe timestamp-ordered processing of simulation events within each LP [12][13]. Figure 2 shows the data structures used: input and output queues on each LP, and a Local Virtual Time (LVT) representing the time of the last processed event. For instance, LP-B uses two input queues: one from LP-A, and one from a different LP (not showed in the figure). At this point, it has processed an event at time 4, and has advanced the LVT=4. Its output queue is connected to LP-A.

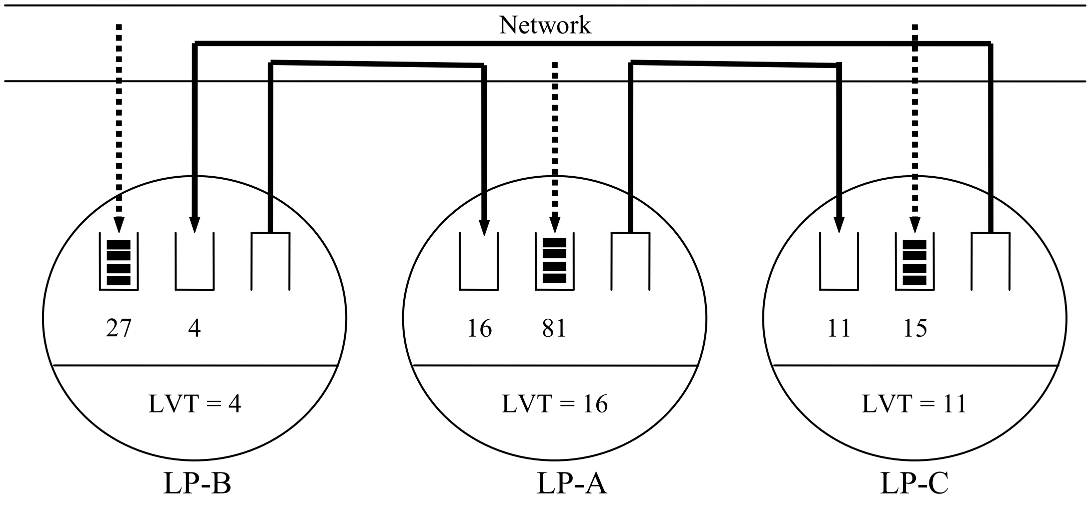


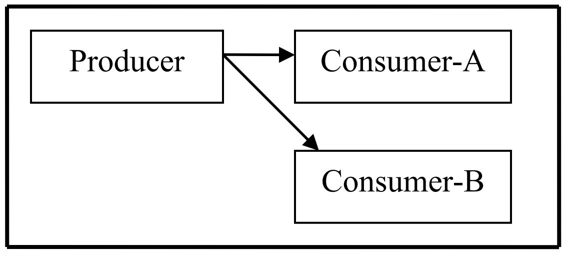
Figure 2: Deadlock situation. Each LP is waiting for an event from another LP

For instance, LP-B has received an event with timestamp = 27, thus we know that it will never receive an event with a smaller timestamp from the same LP. If at that point it receives an event with timestamp 4 from LP-C, LP-B can safely process it (in fact, it can process any event from LP-C earlier than time 27, as we know that we will not receive an event with earlier timestamp). However, LP-B must be blocked once all the unprocessed events from LP-C are processed. If one of the input queues is empty (as in the figure), the LP must be blocked. We cannot guarantee processing other events (for instance, although we have plenty of events in the second queue, as the first one is empty, and the associated timestamp is 4, the LP cannot continue: if we receive, for instance, a new event with timestamp 5 from LP-C, this will cause a causality error). As we can see, the simulation can enter into deadlock when a cycle of empty queues is developed where each process in the simulation is blocked (a shown in the figure).

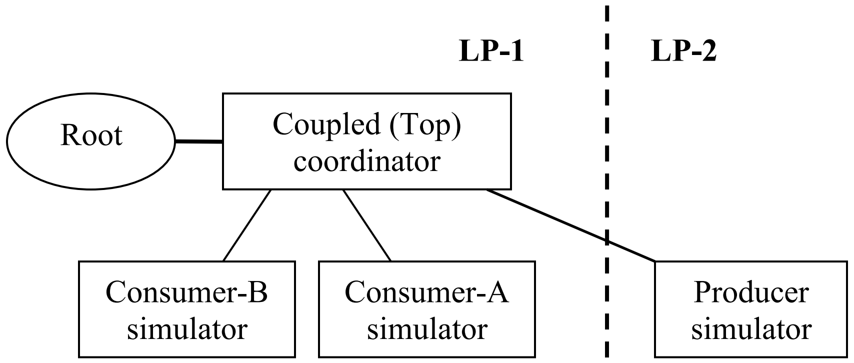
A solution to break the deadlock in Figure 2 is to have each LP broadcasting the lower bound on its time stamp to all other relevant LPs. This is can be accomplished by having LPs sending each other "null" messages with its timestamp. In this case, when an LP processes an event, it sends other LPs a null message, allowing other LPs decide the safe events to process. For instance, in our example, LP-A will inform LP-C that the earliest timestamp for a future event will be 16. Therefore, is it now safe for LP-C to process the next event in the input queue (with timestamp 15), which breaks the hold-and-wait cycle (thus preventing deadlock to occur). These are the basic ideas behind the Chandy/Misra/Bryant algorithm [9][12][23].

Further, runtime performance in conservative algorithms depends on an application property called *lookahead*, which is the time distance between two LPs. The lookahead value can ensure an LP to can process events in the future safely. Let us suppose that LP-A and LP-B in Figure 2represent the time taken to traverse two cities by car (which takes 3 units of simulation time). In this case, if LP-A is at simulation time 16, we know the smallest timestamp it will send to LP-B is 19, so LP-B can safely process events with that timestamp or lower. The lookahead is important value because it determines the degree of parallelism in the simulation and it affects the number of exchanged null messages. Naturally, the lookahead value is very difficult to extract in complex applications. Further, null messages could harshly degrade system performance [23]. Therefore, an LP can advance and process events safely once it realizes the lower timestamp bound and the lookahead information for all other relevant LPs. As a result, many algorithms were proposed during late 1980s and 1990s to arm each LP with this information as efficiently as possible. For example, the barrier algorithms execute the simulation by cycling between phases. Once all an LP reaches the barrier (i.e. a wallclock time), it is blocked until all other LPs get the chance to reach the barrier. In this case, an LP knows that all of its events are safe to process when it executes the barrier primitive (e.g. semaphore). Of course, the algorithms need to deal with the messages remaining in the network (called transient messages) before LPs cross the barrier. Examples of such algorithms are bounded lag [40], synchronous protocol [49] and a barrier technique [50], which deals with the transit messages problem. Different algorithms are discussed in detail in [23].

The above-described algorithms still form the basis of recent conservative distributed simulation. For example, the distributed CD++ (DCD++) [2] is using a conservative approach similar to the barrier algorithms, as shown in Figure 3. DCD++ is a distributed simulation extension of the CD++ toolkit [61], which is based on the DEVS formalism [66]. Figure 3 shows a DEVS coupled model that consists of three atomic models. An atomic model forms an indivisible block. A coupled model is a model that consists of one or more coupled/atomic models. The Producer model in Figure 3-A has one output port linked with the input port of two consumer models. Suppose that this model hierarchy is partitioned between two LPs, as shown in Figure 3-B.



A: Coupled model consists of three atomic models



B: Model hierarchy during simulation split between two LPs

Figure 3: DCD++ Conservative Simulation Example

In this case, each LP (which is a component running in DCD++) has its own unprocessed event queue and the simulation is cycling between phases. In this case, the Root Coordinator starts a phase by passing a simulation message to the topmost Coordinator in the hierarchy. This message is propagated downward in the hierarchy. In return, a DONE message is propagated upward in the hierarchy until it reaches the Root Coordinator. Each model processor uses this DONE message to insert the time of its next change (i.e. an output message to another model, or an internal event message) before passing it to its parent coordinator. A coordinator always passes to its parent the least time change received from its children. Once the Root coordinator receives a DONE message, it advances the clock and starts a new phase safely without worrying about any lingering transit messages in the network. Further, each coordinator in the hierarchy knows which child will participate in the next simulation phase. Furthermore, each LP can safely process any event exchanged within a phase since an event is generated at the time it suppose to be executed by the receiver model. In this approach, the barrier is represented by the arrival of the DONE message at the Root coordinator. However, the Root coordinator does not need to contact any of the LPs because they are already synchronized.

The lookahead value is the most important parameter in conservative algorithms. Therefore, lookahead extraction has been studied intensively by researchers. Recently, the effort has focused on determining the lookahead value dynamically at runtime (instead of static estimation). This is done by collecting lookahead information from the models as much as possible [16][64][42][45].

Optimistic Algorithms

Conservative algorithms avoid violating LPs local causality constraints while optimistic algorithms allow such violations to occur but provide techniques to undo any computation errors. Jefferson's Time Warp mechanism [35] remains the most well known optimistic algorithm. The simulation is executed via a number of Time Warp Processors (TWLP) interacting with each other via exchanging time-stamped event messages. Each TWLP maintains its Local Virtual time (LVT) and advances "optimistically" without explicit synchronization with other processors. On the other hand, a causality error is detected if a TWLP receives a message from another processor with a timestamp in the past (i.e. with a time-stamp less than the LVT), as shown in Figure 4. Such messages are called *straggler* messages.

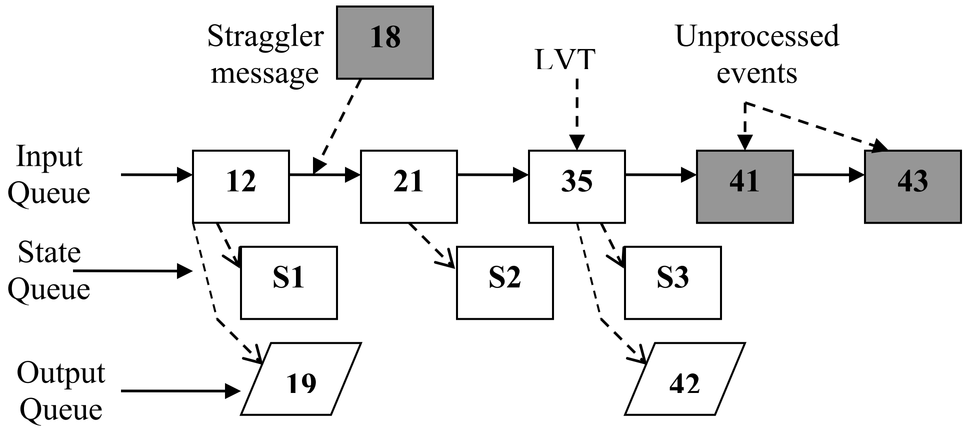


Figure 4: TWLP internal processing

To fix the detected error, the TWLP must rollback to the event before the straggler message timestamp; hence undo all performed computation. Therefore, three types of information are needed to be able to rollback computation:

- An input queue to hold all incoming events from other LPs. This is necessary because the TWLP will have to reprocess those events in case of rollback. The events in this queue are stored according to their received timestamp.
- A state queue to save the TWLP states that might rollback. This is necessary because the simulation state usually changes upon processing an event. Thus, to undo an event processing affect, the prior state of its processing must be restored. For example, as shown in Figure 4, the TWLP must rollback events 21 and 35, upon event 18 (i.e. with timestamp 18) arrival. Thus, the simulation must be restored to state S1, the state that resulted from processing event 12. Afterwards, the processor can process event 18 and reprocess events 21 and 35.
- An output queue to hold the output messages sent to other processors. These messages are sorted according to their sending timestamps. This is necessary because 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* and they may cause a rollback at its destination, triggering other anti-messages, resulting in a cascade of rollbacks in the simulation system. Upon event 18 arrival, in Figure 4, all anti-messages resulted from events 21 and 35 are triggered. In this example anti-message 42 is sent. When anti-message meets its counterpart positive message, they annihilate each other. Suppose the shown processor in Figure 4 receives an anti-message for event 43. In this case, unprocessed event 43 is destroyed without any further actions. On the other hand, if an anti-message is received for event 21, the simulation must be rollback to state S1, LVT is set to 12, and anti-message 42 must be sent to the appropriate processor.

The Time Warp computation requires a great deal memory throughout the simulation execution. Therefore, a TWLP must have a guarantee that rollback will not occur before a certain virtual time. In this case, a TWLP must not receive a positive/negative message before a specific virtual time, called Global Virtual time (GVT). This allows TWLP to reclaim memory via releasing unneeded data such as saved pervious simulation states, and events in the input/output queues with timestamp less than the GVT. Further, the GVT can be used to ensure committing certain operations that cannot be rolled back such as I/O operations. GVT serves as the lower floor for the simulation virtual time. Thus, as the GVT never decreases [20] and the simulation must not rollback below the GVT, all events processed before the GVT can be safely committed (and their memory can be reclaimed). Releasing memory for information older than GVT is performed via a mechanism called *fossil collection*. How often the

GVT is computed, is a trade-off. The more often the computation, it allows better space utilization, but it also imposes a higher communication overhead [23] [55] [43]. For example, the pGVT algorithm [17] allows users to set the frequency of GVT computation at compile time. The GVT computation algorithm described in [4] uses clock synchronization to have each processor start computation at the same time instant. Each processor should have a highly accurate clock to be able to use this algorithm.

The purpose of computing the GVT is to release memory, since the simulation is guaranteed to not rollback below it. The fossil collection manager cleans up all of the objects in the state/input/output queues with timestamp less than the GVT. Many techniques have been used to optimize this mechanism: the infrequent state saving technique (which avoids saving the modified state variables for each event), the one anti-message rollback technique (which avoids sending multiple anti-messages to same LP), or the anti-message with earliest timestamp (only sent to that LP since it suffices to cause the required rollback). Lazy cancellation is a technique that analyzes if the result of the new computed message is the same as the previous one. In this case, an antimessage is not sent [41].

Distributed Simulation Middleware

The main purpose of a distributed simulation middleware is to interoperate different simulation components and between different standards. Integrating new simulation components should be easy, fast and effortless. To achieve this, certain prerequisite conditions must be met [56]:

- the middleware Application Programming Interface (API) should be easy to understand
- it should follow widely accepted standards,
- it should be fast to integrate with new simulation software, and
- it should be interoperable with other middleware, and be independent of diverse platforms.

In the following sections, we will discuss some of the features of existing simulation middleware.

Common Object Request Broker Architecture (CORBA)

As discussed earlier, CORBA [26] is an open standard for distributed object computing defined by the Object Management Group (OMG) [51].

```
module BankAccount
{
    interface account {

      readonly attribute float balance;
      readonly attribute string name;

      void deposit (in float amount);
      void withdraw (in float amount);
    };

    interface accountManager {
      exception reject {string reason;};

      account createAccount (in string name) raises (reject);
      void deleteAccount (in account acc);
    };
};
```

Figure 5: CORBA IDL Example

CORBA objects services are defined using the Interface Definition Language (IDL), which can then be compiled into a programming language stubs such as C, C++ or Java (IDL syntax is similar to other programming languages, as shown in Figure 5). Clients in CORBA invoke methods in remote objects using a style similar to Remote Procedure Calls (RPC), using IDL stubs, as shown in Figure 6. The method call may return another CORBA handle where the client can invoke methods of the returned objects.

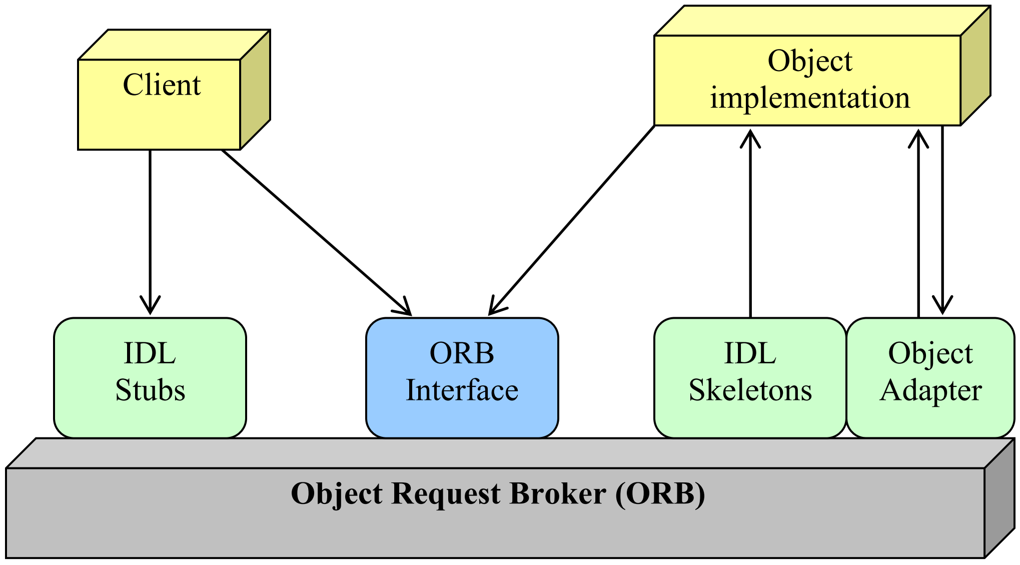


Figure 6: CORBA 2.x Reference Model

Figure 6 shows a picture of CORBA architecture: CORBA IDL stubs and skeletons glue operations between the client and server sides. The Object Request Broker (ORB) layer provides a communication mechanism for transferring client requests to target object implementations on the server side. The ORB interface provides a library of routines (such as translating strings to object references and vice versa). The ORB layer uses the object adapter with routing client requests to objects and with objects activation.

Building distributed simulations using CORBA is straightforward, since CORBA enables application objects to be distributed across a network. Therefore, the issue becomes identifying distributed object interfaces and defining them in IDL, hence a C++/Java local operation call becomes a remote procedure call (hidden by CORBA). Therefore, to support distributed simulation using CORBA, you just need to translate your existing C++/Java simulation interfaces into CORBA IDL definition.

The work in [65] is an example of implementing a distributed DEVS simulation using CORBA. For instance, a DEVS Simulator IDL interface (to the Coordinator, presented in Figure 3) could be defined as follows (tN is the global next event time):

```
Module Simulator{
      Interface toCoordinator
      {
            boolean start();
            double tN?();
            double set_Global_and_Sendoutput (in double tN);
            boolean appIyDeltaFunc(in message);
      };
};
```

The *Simulator* module above is initialized via the method *start*. The *Simulator* module receives its *tN* via the method *set_Global_and_Sendoutput*, and in response, it returns it output. The above IDL code can then be compiled into specific Java/C++ code. For instance, in [65], a DEVS coordinator IDL (to the simulator) was defined as follows:

```
Module Coordinator{
      Interface toSimulator
      {
            boolean register
                  (in Simu1ator::toCoordinator SimObjRef);
            boolean startSimulation();
            boolean stopSimulation();
      };
};
```

The above IDL description shows that the simulator is given an object reference for the *toCoordinator* interface (via register method). As a result, simulators and coordinators can now invoke each other methods (across the network).

High Level Architecture (HLA)

As discussed earlier, the HLA [36] was developed to provide a general architecture for simulation interoperability and reuse [28][29][30]. Table 1 shows the common terminology used in HLA.

Table 1: General HLA Terminology

Term	**Description**
Attribute	Data field of an object.
Federate	HLA simulation processor.
Federation	Multiple Federates interacting via RTI.
Interaction	Event (message) sent between Federates.
Object	Collection of data sent between Federates.
Parameter	Data field of an interaction.

Figure 7 shows the overall HLA simulation interaction architecture. The figure shows HLA simulation entities (called *Federates*). Multiple Federates (called a *Federation*) interact with each other using the *Run-Time Infrastructure* (RTI), which implements the HLA standards. Federates use the *RTIambassador* method to invoke RTI services while the RTI uses the *FederateAmbassador* method to pass information to a Federate in a callback function style. A callback function is a function passed to another function in the form of a reference (e.g., a C++ function pointer) to be invoked later via its reference. For example, in Figure 7, when the federate A sends an interaction (via

RTIambassador) to the federate B, the RTI invokes a function in federate B via that function reference.

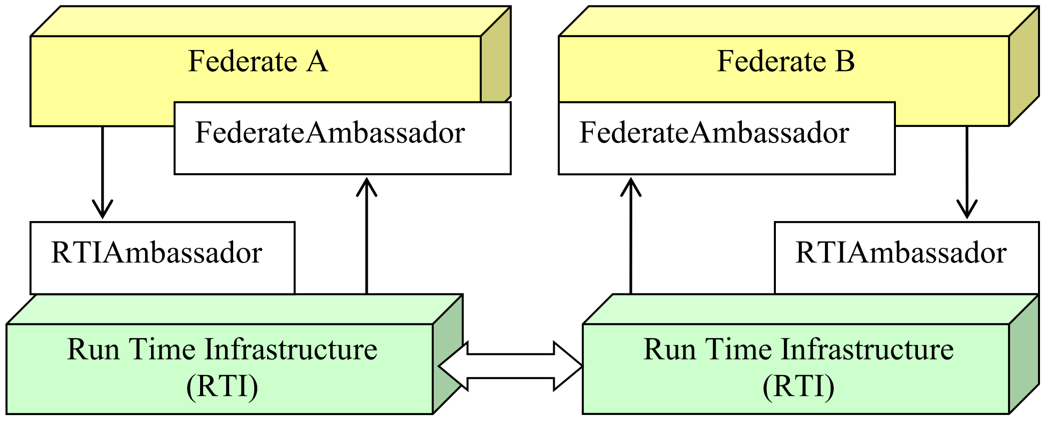


Figure 7: HLA Interaction Overview

The HLA consists of three parts: the *Object Model Template* (OMT) [29] (to document exchanged shared data), the *HLA Interface Specification* [30] (to define RTI/federates interfaces) and the *HLA Rules* [30] (to describe federates obligations and interactions with the RTI).

The *Object Model Template* (OMT) provides a standard for documenting HLA Object Model information. This ensures detailed documentation (in a common format) for all visible objects and interactions managed by federates. Therefore, the data transmitted can be interpreted correctly by receivers to achieve the Federation's objectives. The OMT consists of the following documents:

- The *Federation Object Model* (FOM), which describes the shared object's attributes and interactions for the whole federation (several federates connected via the RTI).

- The *Simulation Object Model* (SOM), which describes the shared object, attributes and interactions for a single federate. The SOM documents specific information for a single simulation.

The HLA interface specification [30] standardized the application programming interface (API) between federates and RTI services. The specification defines RTI services and the required callback functions that must be supported by the Federates. Many contemporary RTI implementations conform to the IEEE 1516 and HLA 1.3 API specifications such as Pitch pRTI™ (C++/Java), CAE RTI (C++), MÄK High Performance RTI (C++/Java) and poRTIco (C++). However, the RTI implementation itself is not part of the standards. Therefore, interoperability between different RTI implementations should not be assumed, since HLA standards do not define the RTI network protocol. In this sense, the standards assume homogeneous RTI implementations in a federation. However, federation should be able to replace RTI implementations since APIs are standardized (bur re-linking and compiling are required). Unfortunately, this is not always the case.

The RTI services are grouped as follows:

- *Federation Management*: services to create and destroy Federation Executions.
- *Declaration Management*: federates must declare exactly what objects (or object attributes) they can produce or consume. The RTI uses this information to tell the producing federates to continue/stop sending certain updates.
- *Object Management*: basic object functions, for instance, deletion/updates of objects.
- *Ownership Management*: services that allow Federates to exchange object attributes ownership between themselves.

- *Time Management*: these services are categorized in two groups:
    - *Transportation* services to ensure events delivery reliability and events ordering.
    - *Time advance* services to ensure logical time advancement correctly. For example, a conservative Federate uses the *Time Advance Request* service with a parameter *t* to request time to advance to *t*. The RTI then responds via invoking the *Time Granted* callback function.
    - Data Distribution Management: it controls filters for data transmission (data routing) and reception of data volume between federates.

The HLA has been widely used to connect HLA-compliant simulations via RTI middleware. However, it presents some shortcomings:

- No standards exist to interoperate different RTI implementations. Therefore, interoperability should not be assumed among RTIs provided by different vendors. Further, standards are too heavy and no load balancing as part of the standards [56].
- The system does not scale well when many simulations are connected to the same RTI. This is because RTI middleware acts as a bus that manages all activates related to connected simulations in a session.
- HLA only covers syntactic (not semantic) interoperability [56].
- Interfacing simulations with RTIs can vary from a standard to another. It is a strong selling point for commercial RTIs that you can use your old HLA 1.3 federates with HLA 1516 [28] RTI implementations.

- HLA API specifications are tied to programming languages. Some interoperability issues need to be resolved when federates are developed with different programming languages.
- Firewalls usually block RTI underlying communication when used on WAN/Internet networks.

A new WSDL API has been added to the HLA IEEE 1516-2007 standard, allowing HLA compliant simulation to be connected via the Internet using SOAP-based Web Services. Some examples of existing HLA-based simulation tools using Web Services include [7][48][68]. As shown in Figure 8, the Web Service Provider RTI component (WSPRC) is an RTI with one or more Web Service ports, allowing HLA to overcome some of its interoperability problems. Therefore, this solution uses Web Service interoperability in the WAN/Internet region while maintaining the standard HLA architecture locally. The WSPRC and WS federate APIs are described in WSDL where a standard federate and standard RTI API is described in actual programming languages. For instance, the Pitch pRTI™ version 4.0 supports Web Services.

WS-based solutions solved interoperability issues at the Federate level. However, this solution still does not solve interoperation of different WSPRC implementations, since the standard does not cover this part. Further, it does not provide a scalable solution, since many simulation components are still managed by a single component.

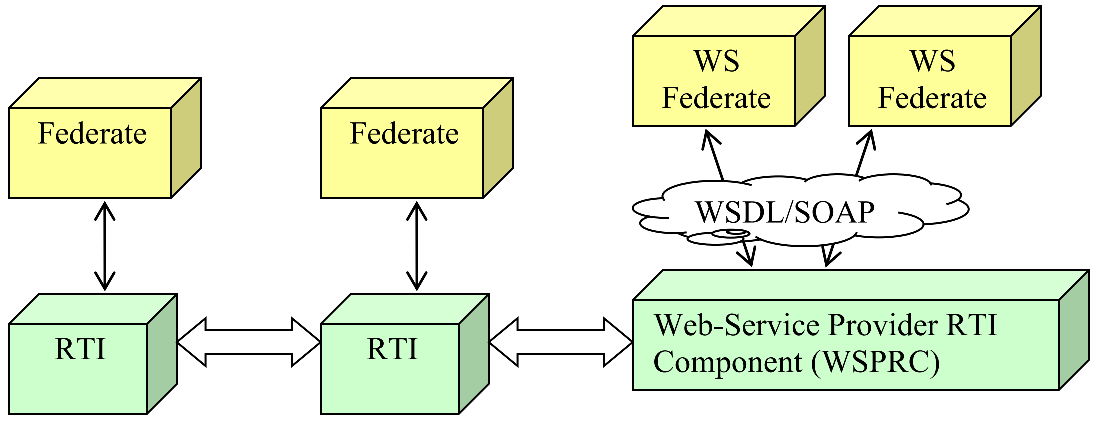


Figure 8: Interfacing RTI with Web Services

SOAP-Based Web Services Middleware

SOAP-based Web Services (or Big Web Services) provide a standard means of interoperating between different heterogeneous software applications, residing on a range of different platforms mainly for software reuse and sharing. At present, it is the leading technology for interoperating remote applications (including distributed simulations) across WAN/Internet networks. For example, a new WSDL API has been added to the HLA IEEE 1516-2007 standard, allowing HLA-compliant simulations to be connected via the Internet using SOAP-based Web Services. Efforts in [7][48][68] are examples of HLA-based simulation using Web Services. Further, the DEVS community is moving toward standardizing interoperability among different DEVS implementations using SOAP-based Web Services [1].

The SOAP-based WS programming style is similar to RPCs, as depicted in Figure 9. The Server exposes a group of *services* that are accessible via *ports*. Each service can be actually seen as an RPC, with semantics described via the procedure parameters. Ports can be viewed as a class exposing its functionality with a number of operations, forming an API accessible to the clients. On other hand, *clients* need to access those services, and they do so using procedure *stubs* at their end. The stubs are local, and allow the client to invoke services as if they were local procedure calls.

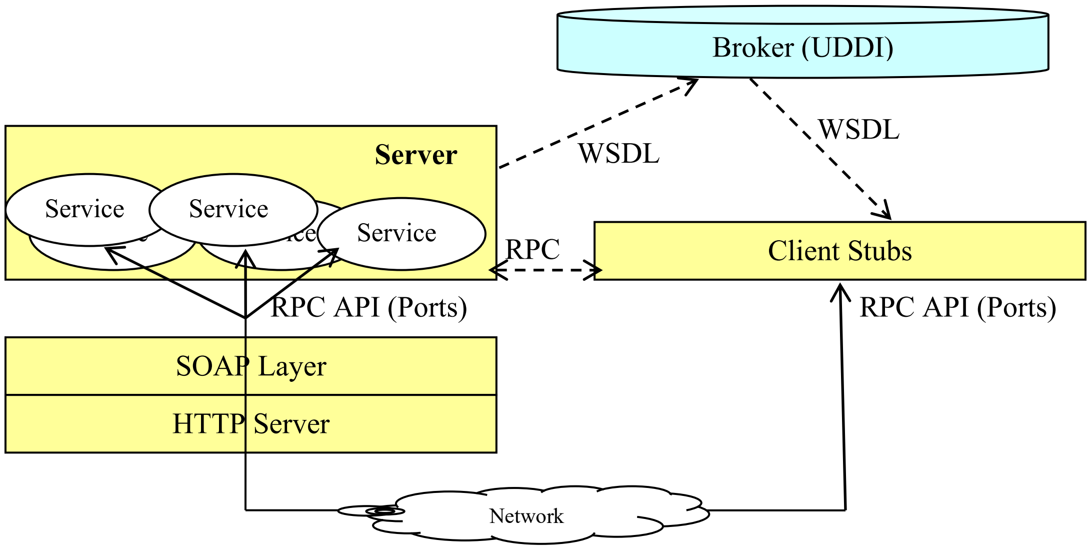


Figure 9: SOAP-based Web Service Architecture Overview

Client programmers need to construct service stubs with their software at compile time. The clients, consume a service at runtime, by invoking its stub. In a WS-based architecture, this invocation is in turn converted into an XML SOAP message (which describes the RPC call). This SOAP message is wrapped into an HTTP message, and sent to the server port, using an appropriate port URI. Once the message is received at the server, an HTTP server located into the same machine passes the message to the SOAP layer (also called SOAP engine; it usually runs inside the HTTP server as Java programs called *Servlets*). The SOAP layer parses the SOAP message and converts it into an RPC call, which is applied to the appropriate port (which activates the right service). In turn, the server returns results to the clients in the same way.

Service providers need to publish the services available (using WSDL documents), in order to enable clients to discover and use the services. One way of doing so, is via a broker called Universal Description, Discovery and Integration (UDDI). UDDI is a directory for storing information about web services and is based on the World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF) standards.

To achieve interoperability, services need to be described in WSDL [15] and published so that clients can construct their RPC stubs correctly. Further, XML SOAP messages ensure a common language between the client and the server regardless of their dissimilarities.

To demonstrate the role of SOAP and WSDL in an example, suppose that a simulation Web Service exposes a port that contains a number of simulation services. Suppose further that the *stopSimulation* service (which takes an integer parameter with the simulation session number, and returns true or false indicating the success or the failure of the operation) is used to abort a simulation, as shown below:

```
boolean stopSimulation(int in0);  // method prototype

...

result = stopSimulation(1000);    // method call
```

From the client viewpoint, the *stopSimulation* service is invoked similarly to any other procedure (using the SOAP-engine API). The responsibility of the SOAP engine (e.g. AXIS server) is to convert this procedure call into XML SOAP message as shown in Figure 10.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
4    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5  <SOAP-ENV:Body>
6    <ns1: stopSimulation xmlns:ns1="http://WS-Port-URI/">
7      <in0 xsi:type="xsd:int">1000</in0>
8    </ns1: stopSimulation>
9  </SOAP-ENV:Body>
10 </SOAP-ENV:Envelope>
```

Figure 10: SOAP Message Request Example

The SOAP message in Figure 10 will be then transmitted in the body of an HTTP message using the HTTP POST method. It is easy to see how an RPC is constructed in this SOAP message. The *stopSimulation* RPC call is mapped to lines 6-8 in Figure 10. Line #6 indicates invocation service *stopSimulation* on Web Service port with URI (*http://WS-Port-URI/*). URIs are WS port addresses (which correspond, for instance, to CORBA object references). Line #7 indicates that this service takes one integer parameter (i.e. simulation session number) with value 1000. Figure 11 shows a possible response to the client as a SOAP message, responding with the *stopSimulation* return value.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
4    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5  <SOAP-ENV:Body>
6    <ns1: stopSimulationResponse xmlns:ns1="http://WS-Port-URI/">
7 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
8                 <return xsi:type="xsd:boolean">true</return>
9    </ns1: stopSimulationResponse>
10  </SOAP-ENV:Body>
11 </SOAP-ENV:Envelope>
```

Figure 11: SOAP Message Response Example

The above-explained example shows how SOAP is used to achieve interoperability. Because the participant parties have agreed on a common standard to describe the RPC call (in this case SOAP), it becomes straightforward for software to convert an RPC (from any programming language) to a SOAP message (and vice versa).

Interoperability cannot only be achieved with SOAP messages because RPCs are programming procedures; hence, they need to be compiled with the clients' software. For example, a programmer writing a Java client needs to know that the *stopSimulation* service method looks exactly as *boolean stopSimulation (int in0)*. Here is where WSDL helps in achieving interoperability for SOAP-based WS. Web Services providers need to describe their services in a WSDL document (and publish them using UDDI) so that clients can use it to generate services stubs.

The major elements of any WSDL document are THE type, message, port Type, binding, port, and service elements. Some of these elements (type, message, and portType) are used to describe the functional behavior of the Web Service in terms of the functionality it offers. On the other hand, binding, port, and service define the operational aspects of the service, in terms of the protocol used to transport SOAP messages and the URL of the service. The former is referred to as abstract service definition, and the latter is known as concrete service definition.

To carry on with our previous example, the simulation service provider should describe the *stopSimulation* service (along with other provided services) in a WSDL document. Figure 12 shows an excerpt of the WSDL description for the *boolean stopSimulation (int in0)* service.

```
1  <wsdl:message name="stopSimulationRequest">
2    <wsdl:part name="in0" type="xsd:int"/>
3  </wsdl:message>
4
5  <wsdl:message name="stopSimulationResponse">
6    <wsdl:part name="stopSimulationReturn" type="xsd:boolean"/>
7  </wsdl:message>
8
9  <wsdl:portType name="CDppPortType">
10  <wsdl:operation name="stopSimulation" parameterOrder="in0">
11   <wsdl:input message="impl:stopSimulationRequest"
12             name="stopSimulationRequest"/>
13   <wsdl:output message="impl:stopSimulationResponse"
14             name="stopSimulationResponse"/>
15 </wsdl:operation>
16
17 </wsdl:portType>
18
19 <wsdl:binding name="CDppPortTypeSoapBinding"
20                     type="impl:CDppPortType">
21   <wsdlsoap:binding style="rpc"
22         transport="http://schemas.xmlsoap.org/soap/http"/>
23   <wsdl:operation name="stopSimulation">
24      <wsdlsoap:operation soapAction=""/>
25      <wsdl:input name="stopSimulationRequest">
26         <wsdlsoap:body encodingStyle="http://.../"
27                namespace="http://..." use="encoded"/>
28      </wsdl:input>
29
30      <wsdl:output name="stopSimulationResponse">
31         <wsdlsoap:body encodingStyle="http://.../"
32               namespace="http://..." use="encoded"/>
33      </wsdl:output>
34    </wsdl:operation>
35 </wsdl:binding>
```

Figure 12: Excerpt of WSDL Document Example

Lines 1-7 show the messages used by the Web Service to send the request and to handle the response. The *stopSimulation* operation uses an input message called *stopSimulationRequest* (which is an *int*eger), and an output message called *stopSimulationResponse* (a Boolean value). Lines 9-17 show the *portType* definition, which is used by operations accessing the Web Service. It defines *CDppPortType* as the name of the port, and *stopSimulation* as the name of the exposed operation by this port.

As discussed earlier, ports define connection points to a web service. If we want to relate this with a traditional program, *CDppPortType* defines a class where *stopSimulation* is a method with *stopSimulationRequest* as the input parameter, and *stopSimulationResponse* as the return parameter. Lines 19-35 show the *binding* of the Web Service, which defines the message format and ports protocol details. The <*wsdlsoap:binding*> element has two attributes: *style* and *transport*. In this example, the style attribute uses the RPC-style, and the transport attribute defines the SOAP protocol to apply. The <*wsdl:operation*> element defines each operation the port exposes. In this case, operation *stopSimulation* is the only one. The SOAP input/output encoding style for operation *stopSimulation* is defined in lines 25-33.

As we can see, it is a great deal of work to describe one RPC. However, mature tools are one of the main advantages of SOAP-based WS. The WSDL document is usually converted to programming language stubs and vice versa with a click of a button (or with a simple shell command).

Using SOAP and WSDL, interoperability is achieved at the machine level regardless of their differences such as programming languages and operating systems. However, interoperability at the human level is still needed. For example, a programmer still needs to know that the integer input parameter to service *stopSimulation* means the simulation session number (even if that programmer was able to compile and invoke the service). This gets worse when a service procedure is complex with many input parameters. Therefore, in practice a text description can be helpful for client programmers. It is possible to add comments to WSDL document like any other XML documents (and WSDL without comments is worse than programming code without them). However, WSDL documents are typically generated by tools (and they need to move comments between WSDL and programming code stubs).

In addition, we need a standardized protocol when using Web Services to interoperate various remote applications (such as interoperation of different simulations to perform distributed simulation). This is because Web Services provide interoperability to overcome differences between machines rather than to overcome the differences between various applications functionalities. Therefore, standards are still needed to accomplish simulations among different simulators successfully. As part of this effort, the DEVS simulation community is in the progress of developing standards to interface different DEVS (and non-DEVS) implementations using SOAP-based Web Services ([1] is an example of such proposals).

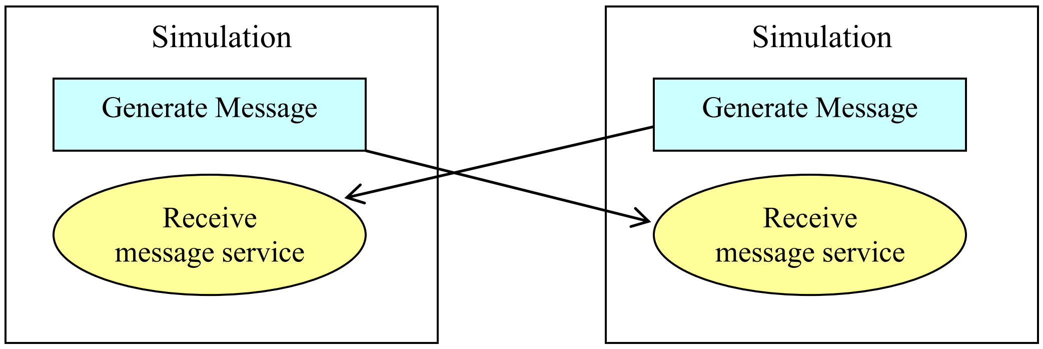


Figure 13: Distributed Simulation using SOAP-based WS

In contemporary WS-based distributed simulations (e.g. [60] and [47]), simulation components act as both client and server at the same time. In this case, a simulator becomes the client when it wants to send a simulation message to a remote simulator (which the later becomes the server), as shown in Figure 13.

SOAP-based distributed simulations share in common that synchronized messages are exchanged in RPC-style where contents are usually passed as input parameters to the RPC (so it becomes like invoking a local call procedure). Further, those RPCs are based on internal software implementation, which makes interfacing standards not easy to achieve among existing systems. This is because each system has already defined its RPC interfaces.

To summarize the major drawback points with SOAP-based Web Services:

- Heterogeneous interface by exposing few URIs (ports) with many operations. Building programming stubs correctly (i.e. compiled without errors) is not enough to interface two different simulators quickly and efficiently. One possible solution, one of the participant parties has to wrap their simulator API with the simulator API to be able to interact with it. Another possible solution is to combine both simulator APIs and expose new set of APIs, assuming this solution works. What happens if many vendor simulators need to interface with each other? It becomes a complex process. In fact, exposing heterogeneous programming procedures of a simulator and expecting it to interoperate with another simulator that is also exposes heterogeneous procedures quickly and efficiently is a naive assumption.
- It uses an RPC-style, which is suitable for closed communities that need to coordinate new changes among each other. In fact, those APIs (services) are programming procedures, which means that they reflect the internal implementation. Therefore, different vendors, for example, have to hold many meetings before they reach an agreement on defining those stubs, because they are tied into their internal implementation; hence it affects a great deal the internal design and implementation of

the simulation package. However, suppose that those different simulator vendors came to an agreement of standardizing the same exposed API, and assume that some changes are required during development or in the future. How easy is to change those standardized APIs? A new coordination among different teams becomes inevitable to redefine new services.

- To use SOAP-based services requires building services stubs at compile time. This can cause more complexity in future advancements if the simulation components can join/leave the simulation at runtime. For example, in [24] the authors present the Ad Hoc distribution simulation, where the number of logical processors (LP) is not known in advance and can be changed during runtime.

RESTful Web Services Middleware

The Representational State Transfer (REST) provides interoperability by imitating the World Wide Web (WWW) style and principles [18]. RESTful Web Services are gaining increased attention with the advent of Web 2.0 [52] and the concept of mashup (i.e. grouping various services from different providers and present them as a bundle) because of its simplicity. REST exposes services as “resources” (which are named with unique URIs similar to Web sites) and manipulated with uniform interface, usually HTTP methods. GET (to read a resource), PUT (to create/update a resource), POST (to append to a resource), and DELETE (to remove a resource). For example, a client applies the HTTP GET method to a resource’s URI in order to retrieve that resource representation (e.g., this is what happens when you browse a Web site). Further, a client can transfer data by applying HTTP methods PUT or POST to a URI. REST applications need to be designed as resource-oriented to get the benefits of this approach (see [54] for design guidelines). REST sometimes is confused with HTTP, since HTTP perfectly matches REST principles. However, REST is an approach that devotes principles such as standardized uniform-interface, universal addressing schemes, and resource-oriented design. REST has been used in many applications such as Yahoo, Flicker, and Amazon S3. It also used in distributed systems as NASA SensorWeb [11] (which uses REST to support interoperability across Sensor Web systems that can be used for disaster management). Another example of using REST to achieve plug-and-play interoperability heterogeneous sensor and actuator Networks is described in [57]. Example of REST usage in Business Process Management (BPM) is described in [37], which focuses on different methods and tools to automate, manage and optimize business processes. REST has also been used for modeling and managing mobile commerce spaces [44].

REST architecture separates the software interface from internal implementation; hence, services can be exposed while software internal implementation is hidden form consumers and providers need to conform to the service agreement, which comes in the form of messages (e.g. XML). This type of design is a recipe for a plug-and-play (or at least semi-automatic) interoperability, as a consumer may search, locate and consume a service at runtime (this is why Web 2.0 applications have expanded beyond regular computer machines to cell phones or any other device connected to the Internet). In contrast, other RPC-style form of interfacing require a programmer to build the interface stubs and recompile the application software before being able to use the intended service. This is clearly not the way to reach a plug-and-play interoperability. Distributed simulation can benefit of this capability toward future challenges (see [56] study) such as having middleware that have a plug-and-play (semi-automatic) interoperability, and accessed by any device from anywhere. Indeed, interoperating two independent developed simulators where each one of them exposes heterogeneous defined set of RPCs is not a trivial task to do. In fact, RPCs are often tied to internal implementation and semantics are described in programming parameters. To add to the situation complexity, many simulators expose many RPCs of many objects (or ports). One has to question if this task worth the cost, particularly if we need to add more independent developed simulators and models. The bottom line is that those simulators are software packages; hence, they interface with their APIs. Therefore, the API design matters when connecting diverse software together. To achieve plug-and-play interoperability, simulators need to have uniform interface and semantics need to be described in form of messages such as XML.

Based on these ideas, we designed RESTful-CD++ [2] the first existing distributed simulation middleware based on REST. The RESTful-CD++ main purpose is to expose services as URIs. Therefore, RESTful-CD++ routes a received request to its appropriate destination resource and apply the required HTTP method on that resource. This makes the RESTful-CD++ independent of a simulation formalism or a simulation engine. CD++ is selected to be the first simulation engine to be supported by the RESTful-CD++ middleware.

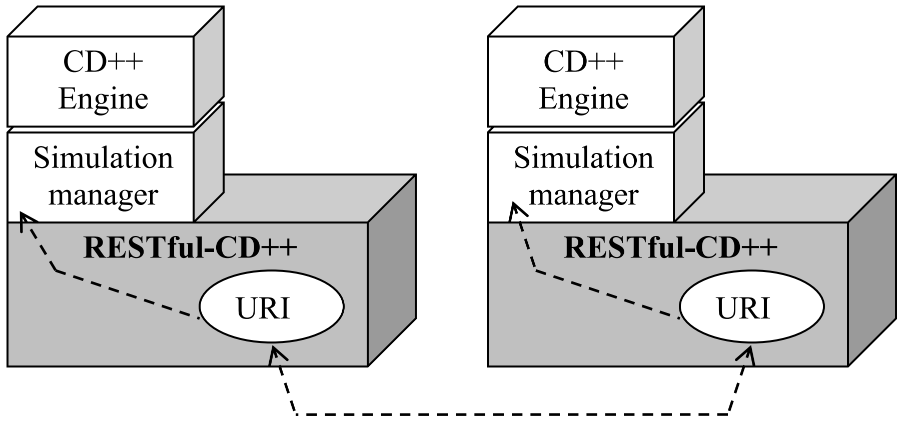


Figure 14: RESTful-CD++ Distributed Simulation Session

In this case, as shown in Figure 14, the simulation manager component is constructed to manage the CD++ distributed simulation such as the geographic existence of model partitions, as shown in Figure 3. The simulation manager is seen externally as a URI (e.g. similar to web site URIs). On the other hand, is a component that manages a distributed simulation logical processor (LP) instance, in our case an LP is a CD++ simulation engine. Therefore, LPs exchange XML simulation messages among each other according to their wrapped URIs (using the HTTP POST method). The RESTful-CD++ exposes its APIs as a regular Web-site URIs that can be mashed up with other Web 2.0 applications (e.g. to introduce real systems in the simulation loop). In addition, it is capable of consuming services from SOAP-based Web Services.

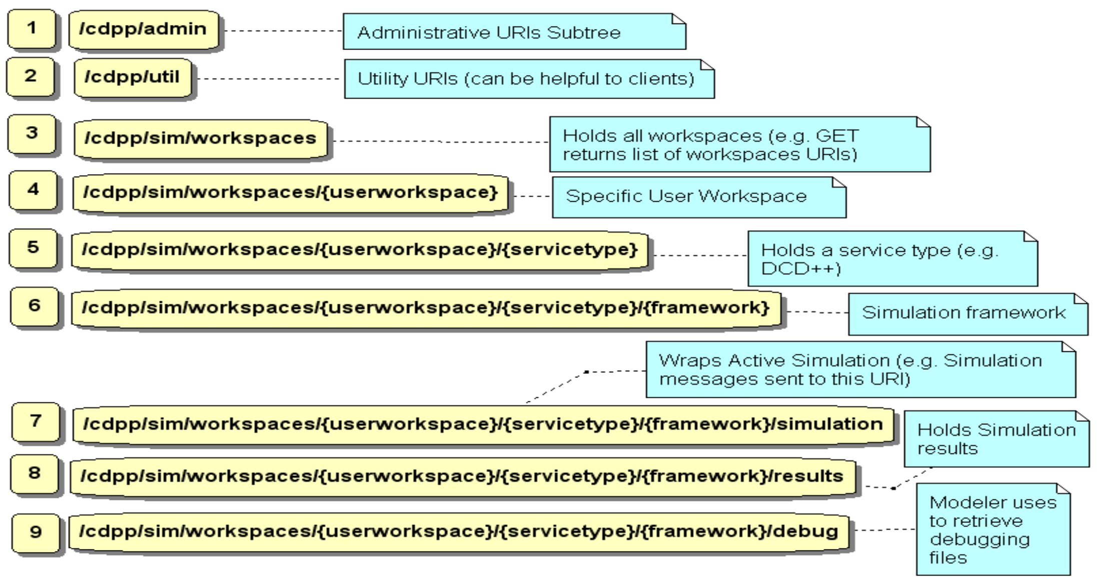

1
/cdpp/admin
Administrative URIs Subtree
2
/cdpp/util
Utility URIs (can be helpful to clients)
3
/cdpp/sim/workspaces
Holds all workspaces (e.g. GET returns list of workspaces URIs)
4
/cdpp/sim/workspaces/{userworkspace}
Specific User Workspace
5
/cdpp/sim/workspaces/{userworkspace}/{servicetype}
Holds a service type (e.g. DCD++)
6
/cdpp/sim/workspaces/{userworkspace}/{servicetype}/{framework}
Simulation framework
Wraps Active Simulation (e.g. Simulation messages sent to this URI)
7
/cdpp/sim/workspaces/{userworkspace}/{servicetype}/{framework}/simulation
Holds Simulation results
8
/cdpp/sim/workspaces/{userworkspace}/{servicetype}/{framework}/results
Modeler uses to retrieve debugging files
9
/cdpp/sim/workspaces/{userworkspace}/{servicetype}/{framework}/debug

Figure 15: RESTful-CD++ URIs Template (APIs)

Summary

Distributed simulation deals with executing simulations on multiple processors connected via communication networks. Distributed simulation can be used to achieve model reuse via interoperation of heterogeneous simulation components, reducing execution time, connecting geographically distributed simulation components, avoiding people/equipment relocations and information hiding – including the protection of intellectual property rights. These simulations are typically composed of a number of sequential simulations where each is responsible of part of the entire model.

The main purpose of a distributed simulation middleware is to interoperate different simulation components and between different standards. Integrating new simulation components should be easy, fast and effortless. A number of middlewares have been used to achieve interoperability among different simulation components such as CORBA, HLA and SOAP-based/REST-based Web-services. HLA is the used

middleware in the military sector where various simulation components are plugged into the RTI, which manages the entire simulation activities. On the other hand, SOAP-based and CORBA expose services as RPC-style via ports/objects where semantic is described in the parameters of those RPCs. REST-based WS, instead, separate interface from internal implementation via exposing standardized uniform interface and describing semantics in form of messages (e.g. XML). REST can provide a new means of achieving a plug-and-play (automatic/semi-automatic) distributed simulation interoperability over the Internet and introducing real systems in the simulation loop (e.g. Web 2.0 mashup applications). This approach has the potential of highly influencing the field, as it would make the use of distributed simulation software more attractive for industry (as one can reuse existing applications and integrate them with a wide variety of e-commerce and business software applications already existing on the Web).

References

[1] Al-Zoubi K.; Wainer, G. "Interfacing and Coordination for a DEVS Simulation Protocol Standard". Proceedings of Distributed Simulation and Real-Time Applications (DS-RT 2008). Vancouver, BC, Canada. 2008.

[2] Al-Zoubi K.; Wainer, G. "Using REST Web Services Architecture for Distributed Simulation". Proceedings of Principles of Advanced and Distributed Simulation PADS 2009, Lake Placid, New York, USA. 2009.

[3] Anita A., Gordon M., David S. "Aggregate Level Simulation Protocol (ALSP) 1993 Confederation Annual Report". The MITRE Corporation. 1993.

http://ms.ie.org/alsp/biblio/93_annual_report/93_annual_report_pr.html. [Accessed March 2009].

[4] Bauer, D.; Yaun, G.; Carothers, C.D.; Yuksel, M.; Kalyanaraman, S. "Seven-O'Clock: A New Distributed GVT Algorithm Using Network Atomic Operations". Proceedings of Principles of Advanced and Distributed Simulation (PADS 2005), Monterey, CA, USA. 2005.

[5] Boer C., Bruin A., Verbraeck A. "Distributed simulation in industry -- a survey, part 3 -- the HLA standard in industry". Proceedings of Winter Simulation Conference (WSC 2008). Miami, FL, USA. 2008.

[6] Boer C., Bruin A. and Verbraeck A. "A survey on distributed simulation in industry". Journal of Simulation. Vol. 3, No. 1, pp. 3–16. March 2009.

[7] Boukerche, A.; Iwasaki, F.M.; Araujo, R.; Pizzolato ,E.B. "Web-Based Distributed Simulations Visualization and Control with HLA and Web Services". Proceedings of Distributed Simulation and Real-Time Applications (DS-RT 2008). Vancouver, BC, Canada. 2008.

[8] Box D., Ehnebuske D., Kakivaya G., Layman A., Mendelsohn N., Nielsen H., Thatte S., Winer D. "Simple Object Access Protocol (SOAP) 1.1". May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. [Accessed March 2009].

[9] Bryant, R. E. "Simulation of packet communication architecture computer systems". Technical Report LCS, TR-188. Massachusetts Institute of Technology. Cambridge, MA, USA. 1977.

[10] Calvin, J.; Dickens, A.; Gaines, B.; Metzger, P.; Miller, D.; Owen, D.; "The SIMNET virtual world architecture". Proceedings of Virtual Reality Annual International Symposium (IEEE VRAIS 1993). Seatlle, WA. 1993.

[11] Cappelaere, P.; Frye, S.; Mandl, D. "Flow-enablement of the NASA SensorWeb using RESTful (and secure) workflows". 2009 IEEE Aerospace conference. Big Sky, Montana, USA. March 2009.

[12] Chandy, K. M. and J. Misra. "Distributed Simulation: A Case Study in Design and Verification of Distributed. Programs". IEEE Transactions on Software Engineering. Vol. SE-5, No. 5, pp. 440-452. 1979.

[13] Chandy, K. M. and J. Misra. "Asynchronous Distributed Simulation via a Sequence of Parallel Computations". Communications of the ACM. Vol. 24, No. 4, pp. 198-205. April 1981.

[14] Christensen E., Curbera F., Meredith G., Weerawarana S. "Web Services Description Language (WSDL) 1.0". http://xml.coverpages.org/wsdl20000929.html. [Accessed March 2009].

[15] Christensen, E; Curbera, F.; Meredith, G.; Weerawarana, S "Web Service Desctiption Language (WSDL) 1.1". March, 2001. http://www.w3.org/TR/wsdl. [Accessed March 2009].

[16] Chung M., Kyung C. "Improving Lookahead in Parallel Multiprocessor Simulation Using Dynamic Execution Path Prediction". Proceedings of Principles of Advanced and Distributed Simulation (PADS 2006). Singapore. May 2006.

[17] D'Souza L.M; Fan, X.; Wisey, P.A. "pGVT: An algorithm for accurate GVT estimation". Proceedings of Principles of Advanced and Distributed Simulation (PADS 1994). Edinburgh, Scotland. July 1994.

[18] Fielding, R. T. "Architectural Styles and the Design of Network-based Software Architectures". Ph.D. Thesis. University of California, Irvine, 2000.

[19] Frécon E.; Stenius M. "DIVE: A scalable network architecture for distributed virtual environments", Distributed Systems Engineering Journal. Vol. 5, No. 3, pp. 91-100. September 1998.

[20] Frey P.; Radhakrishnan, R.; Carter, H.W.; Wilsey, P.A.; Alexander, P. "A formal specification and verification framework for Time Warp based parallel simulation". IEEE Transactions on Software Engineering. Vol. 28, No. 1, pp. 58-78. January 2002.

[21] Fischer M. "Aggregate Level Simulation Protocol (ALSP) - Future Training with Distributed Interactive Simulations", U. S. Army Simulation, Training and Instrumentation Command. International Training Equipment Conference. The Hague, Netherlands. 1995.

[22] Fitzsimmons, E.A.; Fletcher, J.D.; "Beyond DoD: non-defense training and education applications of DIS". Proceedings of the IEEE. Vol. 83, No. 8, pp. 1179 – 1187. August 1995.

[23] Fujimoto, R. M. "Parallel and distribution simulation systems". John Wiley & Sons. New York. 2000.

[24] Fujimoto, R.; Hunter, M.; Sirichoke, J.; Palekar, M.; Kim, H.; Suh Wonho. “Ad Hoc Distributed Simulations”. Proceedings of Principles of Advanced and Distributed Simulation (PADS 2007). San Diego, California, USA. June 2007.

[25] Gan, B. P.; Liu, L.; Jain, S.; Turner, S. J.; Cai, W. T. and Hsu, W.J. “Distributed Supply Chain Simulation Across the Enterprise Boundaries”. Proceedings of Winter Simulation Conference (WSC 2000). Orlando, FL, USA. December 2000.

[26] Henning, M., and S. Vinoski. “Advanced CORBA programming with C++”. Addison–Wesley. Reading, MA. 1999.

[27] Hofer, R.C.; Loper, M.L. “DIS today”. Proceedings of the IEEE. Vol. 83, No. 8, pp. 1124 – 1137. August 1995.

[28] IEEE-1516-2000. “Standard for modeling and simulation (M&S) High Level Architecture (HLA) - frameworks and rules”. 2000.

[29] IEEE-1516.1-2000. “Standard for modeling and simulation (M&S) High Level Architecture (HLA) - federate interface specification”. 2000.

[30] IEEE-1516.2-2000. Standard for modeling and simulation (M&S) High Level Architecture (HLA) - object model template (OMT) specification. 2000.

[31] IEEE-1278.1-1995 - Standard for Distributed Interactive Simulation - Application protocols. 1995.

[32] IEEE-1278.2-1995 - Standard for Distributed Interactive Simulation - Communication Services and Profiles. 1995.

[33] IEEE 1278.3-1996 - Recommended Practice for Distributed Interactive Simulation - Exercise Management and Feedback. 1996.

[34] IEEE 1278.4-1997 - Recommended Practice for Distributed Interactive - Verification Validation & Accreditation. 1997.

[35] Jefferson, D. R. "Virtual time". ACM Transactions on Programming Languages and systems. Vol. 7. No. 3: pp. 405-425. 1985.

[36] Khul F., Weatherly R., Dahmann J.: "Creating Computer Simulation Systems: An Introduction to High Level Architecture". Prentice Hall. 1999.

[37] Kumaran, S.; Rong Liu; Dhoolia, P.; Heath, T.; Nandi, P.; Pinel, F. "A RESTful Architecture for Service-Oriented Business Process Execution". IEEE International Conference on e-Business Engineering (ICEBE '08). Xi'an, China. October 2008.

[38] Lenoir, T. and Lowood, H. "Theaters of wars: the military – entertainment complex". http://www.stanford.edu/class/sts145/Library/Lenoir-Lowood_TheatersOfWar.pdf . [Accessed March 2009].

[39] Kakivaya G., Layman A., S. Thatte S., Winer D. "SOAP: Simple Object Access Protocol". Version 1.0. 1999. http://www.scripting.com/misc/soap1.txt. [Accessed March 2009].

[40] Lubachevsky B. "Efficient distributed event-driven simulations of multiple-loop networks". Proceedings of the 1988 ACM SIGMETRICS conference on Measurement and modeling of computer systems. Santa Fe, NM. 1988.

[41] Lubachevsky, B.; Weiss, A.; Schwartz, A. “An analysis of rollback-based simulation”. ACM Transactions on Modeling and Computer Simulation (TOMACS). Vol. 1, No. 2, pp. 154-193. 1991.

[42] Liu, J.; Nicol, D.M.“Lookahead revisited in wireless network simulations”. Proceedings of Principles of Advanced and Distributed Simulation (PADS 2002). Washington, DC. 2002.

[43] Mattern, F. “Efficient algorithms for distributed snapshots and global virtual time approximation”. Journal of parallel and distributed computing. Vol. 18, No. 4. pp. 423-434. August 1993.

[44] McFaddin, S.; Coffman, D.; Han, J.H.; Jang, H.K.; Kim, J.H.; Lee, J.K.; Lee, M.C.; Moon, Y.S.; Narayanaswami, C.; Paik, Y.S.; Park, J.W.; Soroker, D. “Modeling and Managing Mobile Commerce Spaces Using RESTful Data Services”. 9th IEEE International Conference on Mobile Data Management (MDM'08). Beijing, China. 2008.

[45] Meyer R., Bagrodia L. “Path Lookahead: a Data Flow View of PDES Models”. Proceedings of Principles of Advanced and Distributed Simulation (PADS 1999). Atlanta, GA, USA. 1999.

[46] Misra J. “Distributed discrete-event simulation”. Computing Surveys. Vol. 18 No. 1, pp. 39 – 65. March 1986.

[47] Mittal S., Risco-Martín J.L., and Zeigler B.P. “DEVS-based simulation web services for net-centric T\&E,” in Proceedings of the 2007 summer computer simulation conference, San Diego, California, USA. 2007.

[48] Möller, B. and Dahlin, C. “A First Look at the HLA Evolved Web Service API”. Proceedings of 2006 Euro Simulation Interoperability Workshop, Simulation Interoperability Standards Organization. Stockholm, Sweden. 2006.

[49] Nicol D. “The cost of conservative synchronization in parallel discrete event simulation”. Journal of the ACM. Vol. 40, No. 2, pp. 304 – 333. April 1993.

[50] Nicol D. “Noncommittal barrier synchronization”. Parallel Computing. Vol. 21, No. 4, pp. 529 – 549. April 1995.

[51] Object Management Group (OMG). http://www.omg.org/. [Accessed February 2009].

[52] O'Reilly T. “What Is Web 2.0”. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html. [Accessed May 2009].

[53] Pullen, J.M.; Wood, D.C.; “Networking technology and DIS”. Proceedings of the IEEE. Vol. 83, No. 8, pp. 1156 – 1167. August 1995.

[54] Richardson L., Ruby S. “RESTful Web Services”, 1st edition. O’Reilly Media, Inc., Sebastopol, California. 2007.

[55] Samadi, B. “Distributed simulation, algorithms and performance analysis”. PhD Thesis, Computer science department, University of California, Los Angeles. 1985.

[56] Strassburger S., Schulze T., Fujimoto R. “Future trends in distributed simulation and distributed virtual environments: results of a peer study”. Proceedings of Winter Simulation Conference (WSC 2008). Miami, FL, USA. 2008.

[57] Stirbu, V. “Towards a RESTful Plug and Play Experience in the Web of Things”. IEEE International Conference on Semantic Computing (ICSC 2008). Santa Clara, CA, USA. August 2008.

[58] Taha, H.A. “Simulation with SIMNET II”. Proceedings of Winter Simulation Conference (WSC 1991). Phoenix, Arizona, USA. December 1991.

[59] Taha, H.A. “Introduction to SIMNET v2.0”. Proceedings of Winter Simulation Conference (WSC 1988). San Diego, California, USA. December 1988.

[60] Wainer, G.; Madhoun, R.; Al-Zoubi, K. “Distributed Simulation of DEVS and Cell-DEVS Models in CD++ using Web Services”. Simulation Modelling Practice and Theory. Vol. 16, No. 9, pp. 1266-1292. October 2008.

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

[62] Wainer G., Zeigler B., Nutaro J., Kim T. “DEVS standardization study group Final report”. http://www.sce.carleton.ca/faculty/wainer/standard. [Accessed March 2009].

[63] William E. Babineau, Philip S. Barry, C. Zachary Furness, "Automated Testing within the Joint Training confederation (JTC)", Proceedings of the Fall Simulation Interoperability Workshop, Orlando, FL. September 1998.

[64] Zacharewicz, G. "Giambiasi, N.; Frydman, C.; Improving the lookahead computation in G-DEVS/HLA environment". Proceedings of Distributed Simulation and Real-Time Applications (DS-RT 2005). Montreal, QC. 2005.

[65] Zeigler, B.P.; Doohwan Kim. "Distributed supply chain simulation in a DEVS/CORBA execution environment". Proceedings of Winter Simulation Conference (WSC 1999). Phoenix, AZ. 1999.

[66] Zeigler, B.; Kim, T.; Praehofer. "H. Theory of Modeling and Simulation". 2nd Edition. Academic Press. 2000.

[67] Zeigler, B.; Hammods, P. "Modeling & Simulation-Based Data Engineering: Pragmatics into Ontologies for Net-Centric Information Exchange". Academic Press. 2007.

[68] Zhu H.; Li G.; Zheng L. "Introducing Web Services in HLA-based simulation application". Proceedings of IEEE 7th World Congress on Intelligent Control and Automation (WCICA 2008). Chongqing, China. June 2008.