Agile Net-centric Systems Using DEVS Unified Process

Saurabh Mittal

DUNIP Technologies
PO Box 26218, Tempe AZ 85285 USA
saurabh.mittal@duniptechnologies.com
http://www.duniptechnologies.com

Abstract. Industry and government are spending extensively to transition their business processes and governance to Service Oriented Architecture (SOA) implementations for efficient information reuse, integration, collaboration and cost-sharing. SOA enables orchestrating web services to execute such processes using Business Process Execution Language (BPEL). Business Process Modeling Notation (BPMN) is another method that outputs BPEL for deployment. As an example, the Department of Defenses (DoD) grand vision is the Global Information Grid that is founded on SOA infrastructure. The SOA infrastructure is to be based on a small set of capabilities known as Core Enterprise Services (CES) whose use is mandated to enable interoperability and increased information sharing within and across Mission Areas, such as the Warfighter domain, Business processes, Defense Intelligence, and so on. Net-Centric Enterprise Services (NCES) is DoDs implementation of its Data Strategy over the GIG. However, composing/orchestrating web services in a process workflow (a.k.a Mission thread in the DoD domain) is currently bounded by the BPMN/BPEL technologies. With so much resting on SOA, their reliability and analysis must be rigorously considered. The BPMN/BPEL combination neither has any grounding in system theoretical principles nor can it be used in designing net-centric systems based on SOA in its current state. In this work we present a system theoretical framework using the DEVS Unified Process (DUNIP) that allows bifurcated model-continuity based life cycle process for simultaneous development of the executable system using web-services (including the model) and the automated generation of Test-suite for Verification and Validation. The entire net-centric system, which includes artifacts like the model, the simulation and the real system, is deployed on SOA. The simulation system is made possible on a recently developed DEVS-based service framework called DEVS/SOA. We will show the design of DEVS-agents based on WSDLs and how they are composed towards the systems specification. We will demonstrate how agility is an inherent characteristic of such a system founded on DUNIP. We will also present the case of Department of Defense Architecture Framework (DoDAF) and how agility can be applied to the design and evaluation process.

Keywords: DEVS, DUNIP, DoDAF, SOA, WSWF, NCES, GIG

1 Introduction

Industry and government are spending extensively to transition their business processes and governance to Service Oriented Architecture implementations for efficient information reuse, integration, collaboration and cost-sharing. Service Oriented Architecture (SOA) enables orchestrating web services to execute such processes using Business Process Execution Language (BPEL). Business Process Modeling Notation (BPMN) is another method that outputs BPEL for deployment. As an example, the Department of Defense's (DoD grand vision is the Global Information Grid that is founded on SOA infrastructure. As illustrated in Figure 1, the SOA infrastructure is to be based on a small set of capabilities known as Core Enterprise Services (CES) whose use is mandated to enable interoperability and increased information sharing within and across Mission Areas, such as the Warfighter domain, Business processes, Defense Intelligence, and so on) [16]. Net-Centric Enterprise Services (NCES) [30] is DoD's implementation of its Data Strategy over the GIG. NCES provide SOA infrastructure capabilities such as service and metadata registries, service discovery, user authentication, machine-to-machine messaging, service management, orchestration, and service governance.

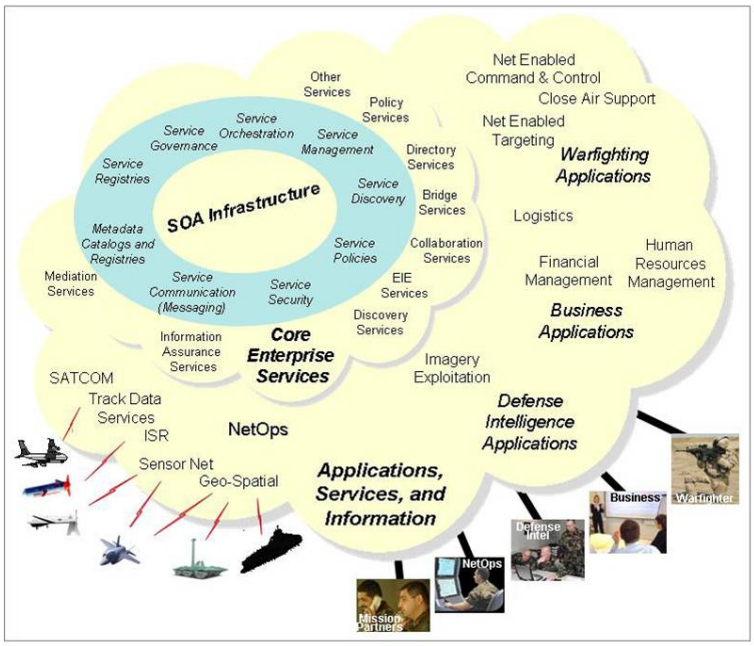

Fig. 1. Core enterprise services in Global Information Grid [16]

However, composing/orchestrating web services in a process workflow (a.k.a Mission thread in the DoD domain) is currently bounded by the BPMN/BPEL technologies. Moreover, there are few methodologies to support such composition/orchestration. Further, BPMN and BPEL are not integrated in a robust manner and different proprietary BPMN diagrams from commercial tools fail to deliver the same BPEL translations. Today, these two technologies are by far the only viable means whereby executives and managers can devise process flows without touching the technological aspects. With so much resting on SOA, their reliability and analysis must be rigorously considered. The BPMN/BPEL combination neither has any grounding in system theoretical principles nor can it be used in designing net-centric systems based on SOA in its current state.

In this research work we provide a proof of concept of how Discrete Event System Specification (DEVS) Formalism can deliver another process work flow mechanism to compose web services in a SOA. A DEVS System is composed of events and components/systems that produce and consume those events. An event is any change in state that merits attention from self/other systems. These systems can be either a simple atomic black box that perform a single task only or they may be a complex system of systems that receive the event and delegate it to one of its sub-components. We will employ DEVS Formalism to a net-centric system deployed using Web Services. Such an architecture where events work along with web services is aptly termed as Service Oriented Architecture (SOA). During this process of designing the net-centric system, we will propose Web Service Work Flow (WSWF) formalism and show how it is executed on the recently developed DEVS/SOA [28] distributed modeling and simulation framework.

In addition to supporting SOA application development, the framework enables verification and validation testing of application. We will also describe how WSWF can be mapped to high level system descriptive frameworks like Department of Defense Architecture Framework (DoDAF) [9],[10],[11], and System Entity Structure (SES). We will demonstrate the execution of WSWF in a complete case-study in which a workflow is composed and executed using DEVS/SOA framework.

Finally, this paper will establish that the DEVS Unified Process inherently is agile and that when deployed on SOA makes it a truly interoperable and testable framework. The paper is organized as follows. Section 2 presents the related technologies. Section 3 describes the underlying technologies that include DEVS, DUNIP, Web Services, Abstract DEVS Service Agent, and DEVS/SOA framework. Section 4 presents layered architecture of Agent-based Test Instrumentation System on/using Global Information Grid using SOA (GIG/SOA). Section 5 deals with Abstract DEVS Service wrapper in detail and also discusses how statistics gathering is integrated with the wrapper design. Section 6 presents the workflow composition and how high-level specifications, as specified by frameworks like DoDAF, can be reduced to WSWF formalism. It is discussed using ontology based System Entity Structure (SES) framework that is targeted to modeling, simulation, systems design and engineering. Section 7 presents a

complete case study demonstrating the usage of WSWF. Section 8 presents some ideas on agility inherent in the DEVS Unified Process. Finally, Section 9 lists conclusions and future work.

2 Related Technologies

In 2003 there were more than 10 recognized groups defining standards for BPM related activities. 7 of these bodies were working on modeling definitions so its no wonder that the whole picture got very confused [31]. Fortunately there has been a lot of consolidation, and currently only 3 key standards to really take notice:

1. BPMN
2. XPDL
3. BPEL

The Business Process Modeling Notation (BPMN) is a standardized graphical notation for graphically representing business processes workflows. BPMNs primary goal is to provide a standard notation that is readily understandable by all business stakeholders. Stakeholders in this definition include business analysts, technical developers and business managers. BPEL is an "execution language" the goal of which is to enable definition of web service orchestrations. Ultimately, BPEL is all about bits and bytes being moved from place to place and manipulated. XPDL is described not an executable programming language like BPEL, but specifically a process design format that literally represents the "drawing" of the process definition.

XPDL is effectively the file format or "serialization" of BPMN. More generally, it can also support any design method or process model that uses the XPDL meta-model. XPDL is a proven format for process design interchange, and it is the most practical standard for establishing a Process Design Ecosystem. Summarizing, currently there is no popular means other than BPMN/BPEL to design a web service workflow orchestration.

3 DEVS Unified Process with DEVS/SOA

3.1 DEVS

Discrete Event System Specification (DEVS) [39] is a formalism, which provides a means of specifying the components of a system in a discrete event simulation. In DEVS formalism, one must specify *Basic Models* and how these models are connected together. These basic models are called *Atomic Models* (Figure 2) and larger models which are obtained by connecting these atomic blocks in meaningful fashion are called *Coupled Models* (Figure 3). Each of these atomic models has *inports* (to receive external events), *outports* (to send events), set of *state*

variables, *internal transition*, *external transition*, and *time advance functions*. Mathematically it is represented as 8-tuple system:

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

where

X is the set of input values
S is the set of states
Y is the set of output values
$\delta_{int} : S \to S$ is the internal transition function
$\delta_{ext} : Q \times X_b \to S$ is the external transition function,
where $X_b$is a set of bags over elements in X, Q is the total state set
$\delta_{con} : S \times X_b \to S$ is the confluent transition function,
subject to $\delta_{con}(s, \phi) = \delta_{int}(s)$
$\lambda : S \to Y_b$ is the output function
$ta : S \to R_{(0+,inf)}$ is the time advance function

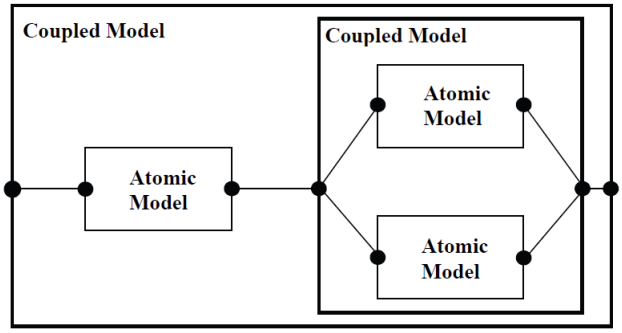

Fig. 2. Hierarchical components at two levels

The models description (implementation) uses (or discards) the message in the event to do the computation and delivers an output message on the outport and makes a state transition. A DEVS-coupled model designates how atomic models can be coupled together and how they interact with each other to form a complex model. The coupled model can be employed as a component in a larger coupled model and can construct complex models in a hierarchical way. The specification provides component and coupling information. The coupled DEVS model is defined as follows.

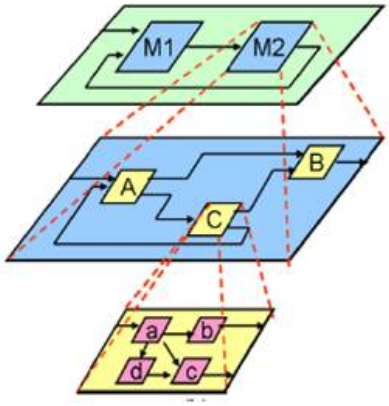

Fig. 3. Hierarchical components for multi-level systems

$$M =< X, Y, D, M_{ij}, I_j, Z_{ij} >$$

where

X is a set of inputs
Y is a set of outputs
D is a set of DEVS component names
for each $i \in D$,
M_i is a DEVS component model
I_i is the set of influences for I
for each $j \in I_i$,
Z_{ij} is the *i-to-j* output translation function.

A Java-based implementation of DEVS formalism, DEVSJAVA [40], can be used to implement these atomic or coupled models. DEVS formalism consists of models, the simulator and the Experimental Frame as show in Figure 4. We will focus our attention to these two types of models i.e. atomic and coupled.

3.2 Web Services and Interoperability using XML

The Service oriented Architecture (SOA) framework is a framework consisting of various W3C standards, in which various computational components are made available as services that interact in an automated manner achieving machine-to-machine interoperable interaction over the network. The interface is specified using Web Service Description language (WSDL) [38] that contains information

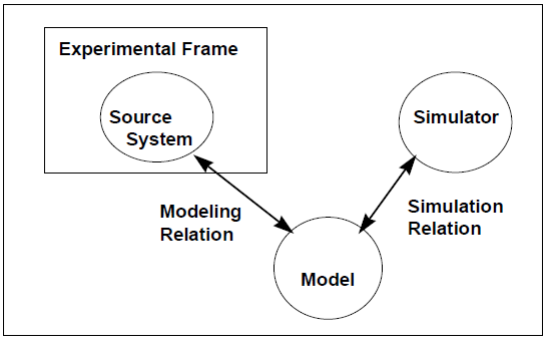

Fig. 4. DEVS separation of the model, the simulation and the Experimental Frame

about ports, message types, port types, and other relating information for binding two interactions. It is essentially a client server framework, wherein client requests a service using a SOAP message that is transmitted via HTTP protocol in the XML format. A Web service is published by any commercial vendor at a specific URL is consumed/requested by another commercial application on the Internet. It is designed specifically for machine-to-machine interaction. Both the client and the server encapsulate their messages in SOAP wrappers.

The fundamental concept of web services is to integrate software application as services. Web services allow the applications to communicate with other applications using open standards. To offer DEVS-based simulators as web services, they must have the following standard technologies: communication protocol (Simple Object Access Protocol, SOAP [35]), service description (Web Service Description Language, WSDL), and service discovery (Universal Description Discovery and Integration, UDDI).

3.3 An abstract DEVS Service Agent

As a crucial part of our workflow, we have designed an Abstract DEVS Service Agent to link DEVS models with Web Services and to generate statistics regarding remote method calls and response times.

Figure 5 depicts an illustrative example. Our proposed model consists of two DEVS atomic models. The DEVS Web Service Consumer invokes the remote operation provided by means of an external transition. When the operation is processed, this atomic model calculates the round-trip-time (RTT) taken by such operation and directs both the RTT and the received response from the Web Service to the DEVS Logger atomic model. At the end of the simulation, the DEVS Logger provides statistics such as operations executed successfully, the RTT consumed per operation, etc.

The DEVS Web Service Consumer needs to be configured by means of: (a) the URL of the Web Service, (b) name of the operations offered by the web service, and (c) the parameters needed by these operations. This information

is specified in the WSDL document. In order to avoid to the user to extract this information by hand, we have implemented a wrapper which automatically generates the DEVS Web Service Consumer for a Web Service. Thus, given a WSDL address, our framework is able to generate the corresponding DEVS Service Agent. Details on how this wrapper is built are given in Section 5.

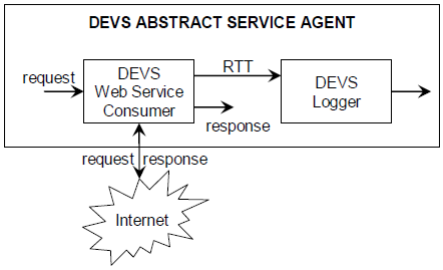

Fig. 5. Schematic showing the architecture of DEVS Agent Service Model

3.4 DEVS/SOA framework for Net-centric Modeling and Simulation

DEVS Modeling Language (DEVSML) is a way of representing DEVS models in the XML language [22]. The DEVSML is built on JAVAML [3], which is an XML representation of JAVA. DEVSML takes its power from the underlying JAVAML that is needed to specify the behavior logic of atomic and coupled models. The DEVSML models are transformable to JAVA in both forward and reverse directions. It is an attempt to provide interoperability between various models and create dynamic scenarios. The layered architecture of this capability is shown in Figure 6. At the top is the application layer that contains model in DEVSJAVA or DEVSML. The second layer is the DEVSML layer itself that provides seamless integration, composition and dynamic scenario construction resulting in portable models in DEVSML that are complete in every respect. These DEVSML models can be ported to any remote location using the net-centric infrastructure and be executed at any remote location. Another major advantage of such capability is total simulator transparency. The simulation engine is totally transparent to model execution over the net-centric infrastructure. The DEVSML model description files in XML contains meta-data information about its compliance with various simulation builds or versions to provide true interoperability between various simulator engine implementations. This has been achieved for at least two independent simulation engines as they have an underlying DEVS protocol to adhere to. This has been made possible with the implementation of a single atomic DTD and a single coupled DTD that validates the DEVSML descriptions generated from these two implementations. Such run-time interoperability provides great advantage when models from different repositories are

used to compose bigger coupled models using DEVSML seamless integration capabilities. More details about the implementation can be seen at [22].

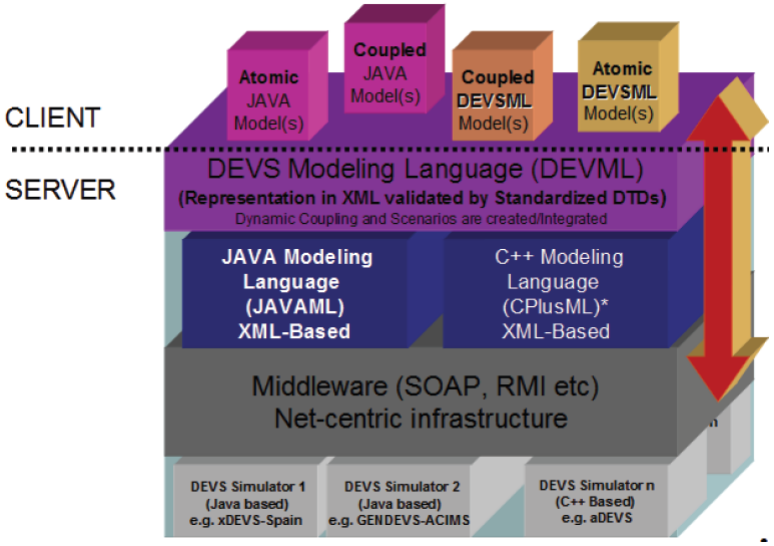

Fig. 6. Layered architecture of DEVSML towards transparent simulators in net-centric domain

The DEVS/SOA framework [28] is analogous to other DEVS distributed simulation frameworks like DEVS/HLA, DEVS/RMI and DEVS/CORBA [13],[36], [7],[17],[41]. The distinguishing mark of DEVS/SOA is that it uses SOA as the network communication platform and XML as the middleware and thus acts as a basis of interoperablity using XML [27]. Furthermore, it uses web-services as the underlying technology to implement the DEVS simulation protocol.

The complete setup requires one or more servers that are capable of running DEVS Simulation Service, as shown in Figure 7. The capability to run the simulation service is provided by the server side design of DEVS Simulation protocol supported by the latest DEVSJAVA Version 3.1 [1].

The numerous modes of DEVS model generation are beyond the scope of this paper (the interested reader is referred to [24]. Once a DEVS model package is developed, the next step is simulation as illustrated in Figure 7.The DEVS/SOA client (Figure 8) takes the DEVS models package and through the dedicated servers hosting DEVS simulation services, it performs the following operations:

- Upload the models to specific IP locations i.e. partitioning (Figure 9)
- Run-time compile at respective sites
- Simulate the coupled-model
- Receive the simulation output at clients end

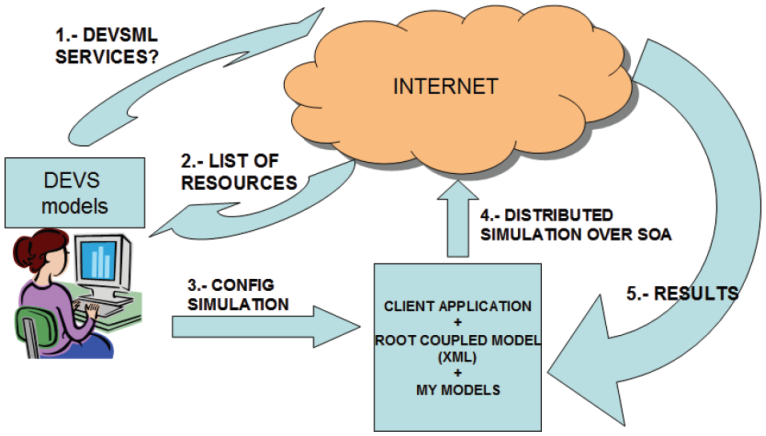

Fig. 7. Execution of DEVS models using DEVS/SOA framework

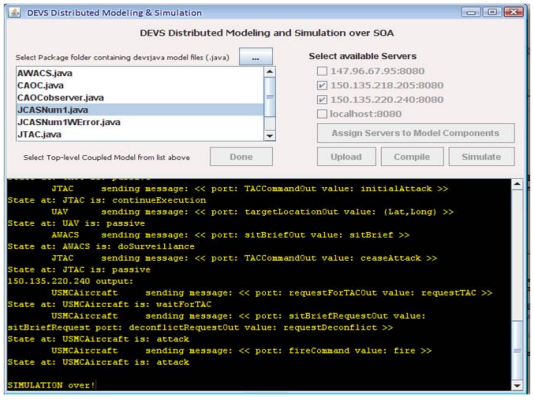

Fig. 8. DEVS/SOA client hosting a distributed simulation

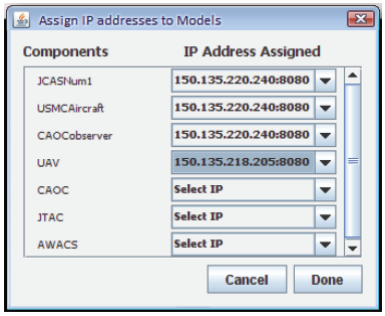

Fig. 9. Server assigned to models using manual model partitioning

This section has laid the foundation of net-centric DEVS framework called DEVS/SOA that allows deployment of DEVS models to specific IP addresses and allows interoperability between DEVS models using DEVSML. It provides a layered framework in which the models are transparent to their simulators. In the next section we will see how the net-centric DEVS is applicable to testing of Global Information Grid based on Service Oriented Architecture (GIG/SOA). A sample movie of DEVS/SOA in action is available at [34].

3.5 DEVS Unified Process a.k.a DUNIP

This section describes the bifurcated Model-Continuity process [24] and how various elements like automated DEVS model generation, automated test-model generation (and net-centric simulation over SOA are put together in the process, resulting in DEVS Unified Process (DUNIP) [24],[12]. The DEVS Unified Process (DUNIP) is built on the bifurcated Model-continuity based life-cycle methodology. The design of simulation-test framework occurs in parallel with the simulation-model of the system under design. The DUNIP process consists of the following elements:

1. Automated DEVS Model Generation from various requirement specification formats
2. Collaborative model development using DEVS Modeling Language (DEVSML)
3. Automated Generation of Test-suite from DEVS simulation model
4. Net-centric execution of model as well as test-suite over SOA

Considerable amount of effort has been spent in analyzing various forms of requirement specifications, viz, state-based, Natural Language based, UML-based, Rule-based, BPMN/BPEL-based and DoDAF-based, and the automated processes which each one should employ to deliver DEVS hierarchical models and DEVS state machines [24],[15]. Simulation execution today is more than just model execution on a single machine. With Grid applications and collaborative computing the norm in industry as well as in scientific community, a net-centric platform using XML as middleware results in an infrastructure that supports distributed collaboration and model reuse. The infrastructure provides for a platform-free specification language DEVS Modeling Language (DEVSML) [22] and its net-centric execution using Service-Oriented Architecture called DEVS/SOA [23]. Both the DEVSML and DEVS/SOA provide novel approaches to integrate, collaborate and remotely execute models on SOA. This infrastructure supports automated procedures for test-case generation leading to test models.

Using XML as the system specifications in rule-based format, a tool known as Automated Test Case Generator (ATC-Gen) was developed which facilitated the automated development of test models [18],[42],[19]. DUNIP (Figure 10) can be summarized as the sequence of the following steps:

1. Develop the requirement specifications in one of the chosen formats such as BPMN, DoDAF, Natural Language Processing (NLP) based, UML based or simply DEVS-based for those who understand the DEVS formalism.
2. Using the DEVS-based automated model generation process, generate the DEVS atomic and coupled models from the requirement specifications using XML
3. Validate the generated models using DEVS W3C atomic and coupled schemas to make them net-ready capable for collaborative development, if needed. This step is optional but must be executed if distributed model development is needed. The validated models which are Platform Independent Models (PIMs) in XML can participate in collaborative development using DEVSML.
4. From step 2, either the coupled model can be simulated using DEVS/SOA or a test-suite can be generated based on the DEVS models.
5. The simulation can be executed on an isolated machine or in distributed manner (using SOA middleware if the focus is net-centric execution). The simulation can be executed in real-time as well as in logical time.
6. The test-suite generated from DEVS models can be executed in the same manner as laid out in Step 5.
7. The results from Step 5 and Step 6 can be compared for verification and validation process.

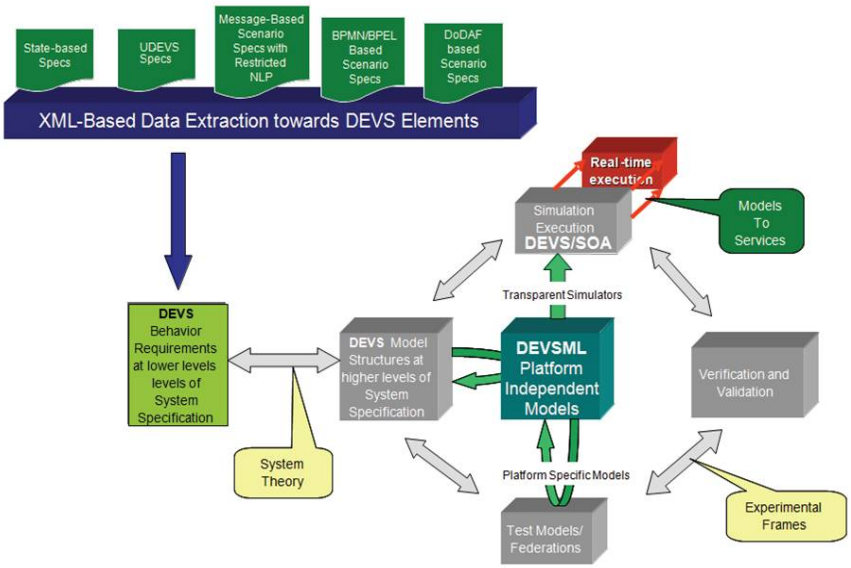

Fig. 10. The complete DEVS Unified Process

4 Multi-layered Agent-based Test Instrumentation System using GIG/SOA

A DEVS distributed federation is a DEVS coupled model whose components reside on different network nodes and whose coupling is implemented through middleware connectivity characteristic of the environment, e.g., SOAP for GIG/SOA, The federation models are executed by DEVS simulator nodes that provide the time and data exchange coordination as specified in the DEVS abstract simulator protocol. The DEVS Agent Monitoring System or Test Instrumentation System (TIS) is a DEVS coupled system that observes and evaluates the operation of the DEVS coupled system model. The DEVS models used to observe other participants are the DEVS test-agents. The TIS should provide a minimally intrusive test capability to support rigorous, ongoing, repeatable and consistent testing and evaluation (T&E). Requirements for such a test implementation system include ability to

1. deploy agents to interface with SoS component systems in specified assignments
2. enable agents to exchange information and coordinate their behaviors to achieve specified experimental frame data processing
3. respond in real-time to queries for test results while testing is still in progress
4. provide real-time alerts when conditions are detected that would invalidate results or otherwise indicate that intervention is required
5. centrally collect and process test results on demand, periodically, and/or at termination of testing.
6. support consistent transfer and reuse of test cases/configurations from past test events to future test events, enabling life-cycle tracking of SoS performance.
7. enable rapid development of new test cases and configurations to keep up with the reduced SoS development times expected to characterize the reusable web
8. service-based development supported on the GIG/SOA.

Many of these requirements are not achievable with current manually-based data collection and testing. Instrumentation and automation are needed to meet these requirements. Net-centric Service Oriented Architecture (SOA) provides a currently relevant technologically feasible realization of the concept. As discussed earlier, the DEVS/SOA infrastructure enables DEVS models, and test agents in particular, to be deployed to the network nodes of interest. [26],[43] provides complete detail on how such observers can be autogenerated and be executed using DEVS/SOA.

4.1 Deploying Test Agents over the GIG/SOA

Figure 11 depicts a logical formulation test federation that can observe a System Under Test (SUT) to verify the message flow among components as derived from

information exchange requirements. In this context, a mission thread is a series of activities executed by operational nodes. In playing out this thread, DEVS test models are informed of the current activities (or see to detect their onset) as well as the operational nodes that execute these messages. These test models watch messages sent and received by the components that host the participating operational nodes. The test models check whether such messages are the ones that should be sent or received under the current function.

The test-agents are contained in DEVS Experimental Frames (EF) are implemented as DEVS models, and distributed EFs are implemented as DEVS models, or agents as we have called them, reside on network nodes. Such a federation, illustrated in Figure 12, consists of DEVS simulators executing on web servers on the nodes exchanging messages and obeying time relationships under the rules contained within their hosted DEVS models. This DEVS Agent Monitoring System that contains DEVS models interacts with real world web services, as we shall in Section 7 case study.

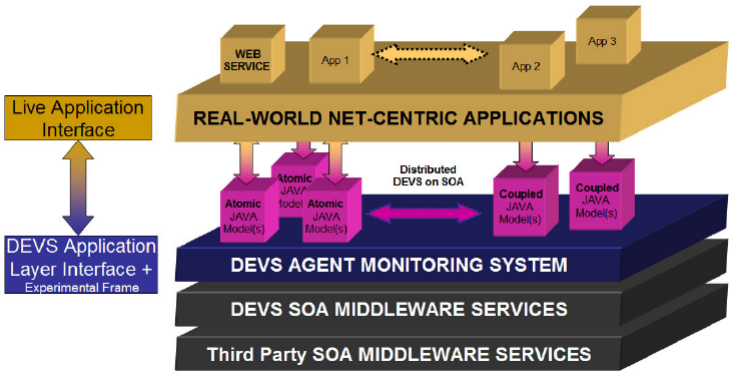

Fig. 11. Multi-layered Agent-based Test Instrumentation Framework

4.2 Implementation of Test Federations

A test federation observes an orchestration of web-services to verify the message flow among participants adheres to information exchange requirements. A good way to specify these requirements is through Department of Defense Architecture Framework (DoDAF) that have specific documents (OV-3 and SV-6) to localize these information exchanges [9]. These documents very well define the input and output messages for the constituent system and operational components. As derived from DoDAF inputs, a mission thread is a series of activities executed by operational nodes and employing the information processing functions of web-services. As discussed in [26],[43], test agents watch messages sent and received by the services that host the participating operational nodes. Depending on

the mode of testing, the test architecture may, or may not, have knowledge of the driving mission thread under test. If thread knowledge is available, DEVS test agents can be aware of the current activity of the operational nodes it is observing. This enables it to focus more efficiently on a smaller set of messages that are likely to provide test opportunities. A DEVS distributed federation is a DEVS coupled model whose components reside on different network nodes and whose coupling is implemented through middleware connectivity characteristic of the environment, e.g., SOAP for GIG/SOA, The federation models are executed by DEVS simulator nodes that provide the time and data exchange coordination as specified in the DEVS abstract simulator protocol.

To help automate set-up of the test we use a capability to inter-covert between DEVS and XML. DEVSML allows distributing DEVS models in the form of XML documents to remote nodes where they can be coupled with local service components to compose a federation [23],[24]. The layered middleware architecture capability is shown earlier in Figure 6. Such run-time interoperability provides great advantage when models from different repositories are used to compose models using DEVSML seamless integration capabilities. Finally, the test federation is illustrated in Figure 12 where different models (federates) in DEVSML collaborate for a simulation exercise over GIG/SOA.

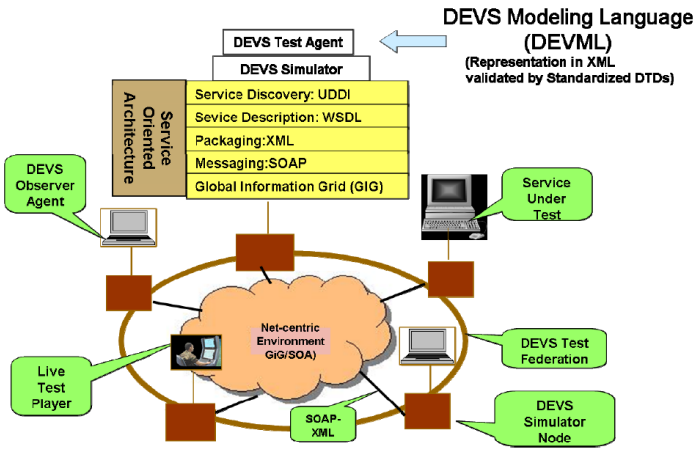

Fig. 12. Protypical DEVS Test Federation

This section has laid out the framework on the creation and execution of a DEVS-based test instrumentation system. More details on the TIS design aspects can be seen in [26]. In the next section we will demonstrate how it can be applied to web services framework.

5 Abstract DEVS Service Wrapper

This section will provide details about the role of DEVS interface with a live web service. This is the most crucial step as it links a live web service with a modeling and simulation framework. It is the seat of model-continuity [52] where a DEVS atomic model performs the dual role of a model as well as a wrapper for a real software application utilizing web services.

Web services are utilized using web service clients that are created by various open source and commercially available tools such as Eclipse Web Service Toolkit (WST), Netbeans IDE, Websphere etc.. All of them use the WSDL as the input to generate the web service client. In our implementation we utilize the Axis2 framework to generate clients. Our choice of Axis2 plugin is driven by the implementation platform of DEVS/framework which is Axis/Java. However, it doesnt matter which method is used to generate the client.

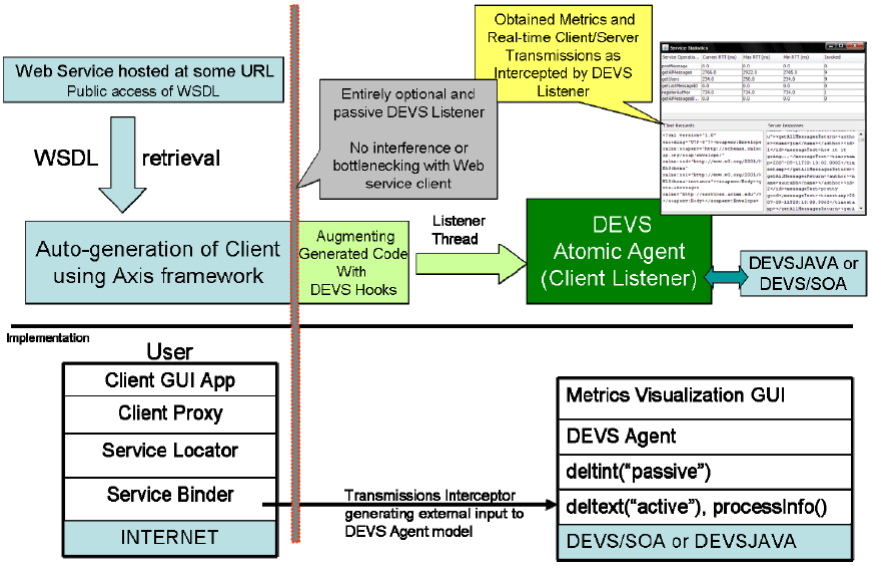

Fig. 13. DEVS Wrapper implementation over an Axis Web Service client

A DEVS model has two modes of operation: an internal behavior representation and an external behavior representation. In developing a DEVS wrapper, which would be effectively a DEVS web service client, we will implement the external behavior. The concept is shown in the top half of Figure 13. The detail is shown in the lower half of the same Figure 13. It shows the mapping between the Axis layers, specifically the Axis binding layer and the DEVS elements. It describes the external event that is triggered whenever there is message exchange through the Axis client. This triggered event informs the DEVS atomic model

that wraps this Axis client. Such an arrangement does not create any bottleneck or any pipe between the actual Axis client and the network. The DEVS wrapper is informed of the round-trip-time (RTT) when the actual service has been executed its completion. Consequently, it is a passive observer and offers no interference to the true communication between the client and the live web service. By inserting a specific set of code in any Axis generated client, we can create a DEVS wrapper that is ready to become a part of a test-agent federation coupled system, as described in the previous section.

Having described the basic DEVS Web service wrapper, the next task in line is the creation of a coupled model, a web service workflow to be more specific to actually utilize the DEVS modeling and simulation capabilities.

6 Workflow composition and DoDAF-based Mission Threads

Web service workflows and orchestration is generally done using BPEL or BPMN or hard-coded in a language specific platform implementation such as Java or .NET. However, to create a DEVS coupled model there are numerous ways [24]. For example the most recent XML-Based Finite Deterministic DEVS (XFD-DEVS) [25] uses XML as the preferred means to develop a Platform Independent Model for both atomic and coupled models. Providing another method to create a web service workflow is beneficial for both the communities. Not only does it provide modeling and simulation capabilities to the existing Web Service architecture, it also establishes DEVS as a production environment that can effectively create application level code using system theoretical concepts.

Another mode of system level design is made possible by System Entity Structure (SES) [44]. The SES is a high level ontology framework targeted to modeling, simulation, systems design and engineering. Its expressive power, both in strength and limitation, derive from that domain of discourse. An SES is a formal structure governed by a small number of axioms that provide clarity and rigor to its models. The structure supports hierarchical and modular compositions allowing large complex structures to be built in stepwise fashion from smaller, simpler ones. Tools have been developed to transform SESs back and forth to XML allowing many operations to be specified in either SES directly or in its XML guise. The axioms and functionality based semantics of the SES promote pragmatic design and are easily understandable by data modelers. Together with the availability of appropriate tool support, it makes development of XML Schema transparent to the modeler. Finally, SES structures are compact relative to equivalent Schema and automatically generate associated executable simulation models.

The most recent Department of Defense Architecture Framework (DoDAF) application to GIG/SOA is another contender to compose web service workflows for mission-thread design and evaluation. DoDAF, as applicable to mission-thread testing, consists of three views: Operational View (OV), Systems View (SV) and Technical View (TV). It comprises of 26 documents1 to describe a

mission thread. Wrapping head around such documents require sufficient level of understanding and experience with C4ISR frameworks. The main documents are listed in Table 1.

Table 1. Relevant DoDAF products

Description	**DoDAF Type**
Overview and Summary Information	AV-1
High-Level Operational Concept Description	OV-1
Operational Node Connectivity Description	OV-2
Operational Information Exchange Matrix	OV-3
Organizational Relationships	OV-4
Operational Activity Model	OV-5
Operational Event Trace Description	OV-6b,c
Systems Interface Description	SV-1
Communication Description	SV-2
Systems to Systems Matrix	SV-3
Functionality Description	SV-4
Operational Activity to Function Traceability Matrix	SV-5
Data Exchange Matrix	SV-6
Technical Standards Profile	TV-1

For more detailed analysis of DoDAF, refer [20],[21]. Figure 14 shows the various DoDAF views map into the SES framework.

Operational and System perspectives are considered two different decompositions of the system under consideration. They are represented by corresponding nodes called aspects labeled by the names, Operational View and System View, respectively. The Operational View aspect has entities labeled opNodes (operational nodes) and activities. The various operational views of DoD AF (other than OV-4) are easily interpreted as describing the entities and their interactions. Likewise, the System View aspect has entities labeled functions with DoD AF views that are associated with the functions and their interactions. The one exception is SV-5 which is a relation between the functions of the System View and the activities of the Operational View. This view describes how the activities are implemented via executable functions supplied by the system.

Although the current DoDAF specification provides an extensive methodology for system architectural development, it is deficient in several related dimensions absence of integrated modeling and simulation support, especially for model-continuity throughout the development process, and lack of associated testing support [20]. To overcome these deficiencies, we described an approach to support specification of DoDAF architectures within a development environment based on DEVS-based modeling and simulation. The authors [20],[45] enhanced the DoDAF specification to incorporate M&S as a means to develop executable architecture [2] from DoDAF specifications and provided detailed DoDAF to DEVS mapping leading to simulation, and feasibility analysis.

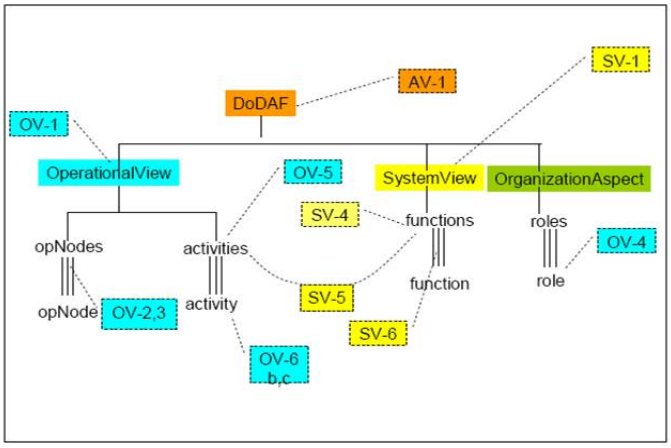

Fig. 14. Mapping of DoDAF documents to System Entity Structure (SES)

6.1 Web Service Work Flow Formalism

So, after providing an overview of various frameworks that can compose a web service workflow, or simply a process workflow based on certain goals, objectives or requirements, we can deduce the information we need to compose a workflow and develop an automated procedure towards DEVS based design and analysis. The information set for a Web Service workflow formalism can be described in a four element tuple as:

$$WSWF :< W, M, F, X >$$

where,

W: Set of Web service definitions (WSDLs) or Agents each with a valid URL

M : Set of web service methods

F : defined as $< C, L, D >$

where,

C is a set of W-M pairs with each pair as a source or destination

L is a set of partner links with each link containing a
src and dest pair defined in C

D is a type of workflow mode which can either be a
sequence, while, holdSend or concurrent type which are
corresponding to the BPEL specifications

X is a Set of messages,

where,

each Message contains Data and is defined by time of entry in system,

rate, whether it is periodic or stochastic and can be either an Input message or an Output message

The WSWF is expressed in SES as shown in Figure 15 and Figure 16:

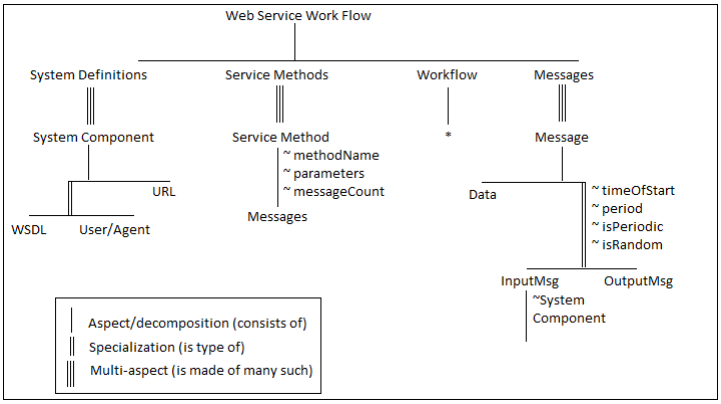

Fig. 15. SES representation of Web Service Work Flow Formalism

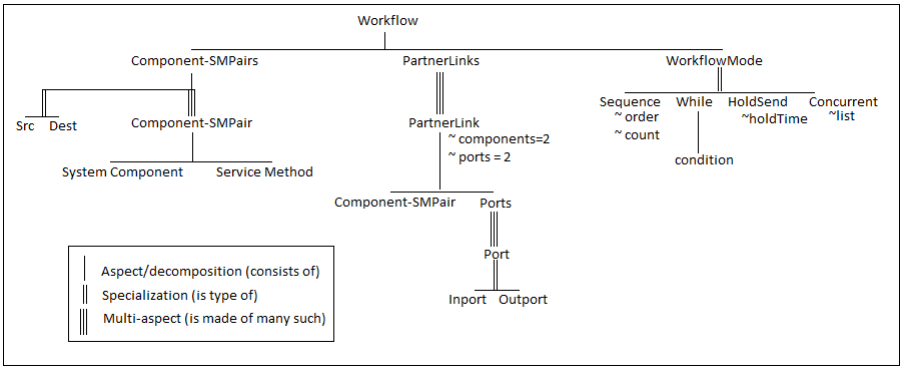

Fig. 16. SES representation of Workflow entity from Figure 15

The WSWF is represented using natural language as shown in Figure 17.

By expressing the SES for WSWF formalism in restricted natural language, it is made executable using SES-DEVS methodology as elaborated in Zeiglers recent book [43]. Using the SES builder [33], we can very well extract the DTD and/or schema for WSWF. The generated DTD for WSWF is provided below.

```
<?xml version='1.0' encoding='us-ascii'?>
```

```
From FORMALISM perspective, WSWF is made of SystemDefinitions, ServiceMethods, and Workflow!
From INFO perspective, WSWF is made of Messages!
From SystemContainer perspective, SystemDefinitions is made of more than one SystemComponent!
From MethodContainer perspective, ServiceMethods is made of more than one ServiceMethod!
From MessageContainer perspective, Messages is made of more than one Message!

From SystemStructure perspective, SystemComponent is made of URL!
From MessageStructure perspective, Message is made of DATA!
From WorkflowStructure perspective, Workflow is made of WorkflowMode,
ComponentServiceMethodPairs, and PartnerLinks!
From InfoStructure perspective, ServiceMethod is made of InfoExchanges!

From INFO perspective, InfoStructure is like MessageContainer!

SystemComponent can be WSDL, or USERAGENT in SystemType!
Message can be InputMsg, or OutputMsg in MessageType!
Message has timeOfStart, period, is_Periodic, and is_Random!
InputMsg has SystemComponent!

ServiceMethod has methodName, parameters, and messageCount!

WorkflowMode can be Sequence, While, HoldSend, or Concurrent in WorkflowType!
From WhileStructure perspective, While is made of Condition!
Sequence has order, and count!
HoldSend has holdTime!
Concurrent has List!

From ComponentSMPairContainer perspective, ComponentServiceMethodPairs is made of more than
one ComponentServiceMethodPair!
From PartnerLinkContainer perspective, PartnerLinks is made of more than one PartnerLink!
PartnerLinks has ComponentCount and PortCount!
the range of PartnerLinks's ComponentCount is RANGE with values (2,2)!
the range of PartnerLinks's PortCount is RANGE with values (2,2)!

From PartnerLinkStructure perspective PartnerLink is made of ComponentServiceMethodPair, and
Ports!
ComponentServiceMethodPairs can be Src, or Dest in ComponentType!
From PortContainer perspective, Ports is made of more than one Port!
Port can be Inport, or Outport in PortType!
```

Fig. 17. WSWF SES expressed in natural language for automated schema and DEVS code generation

```
<!-- DTD for a WSWF -->
<!ELEMENT WSWF (aspectsOfWSWF)>
<!ELEMENT aspectsOfWSWF (WSWF-FORMALISMDec | WSWF-INFODec )>
<!ELEMENT WSWF-FORMALISMDec ( Workflow , ServiceMethods ,
 SystemDefinitions )>
<!ELEMENT Workflow (aspectsOfWorkflow)>
<!ELEMENT aspectsOfWorkflow (Workflow-WorkflowStructureDec )>
<!ELEMENT Workflow-WorkflowStructureDec
  ( PartnerLinks , WorkflowMode ,  ComponentServiceMethodPairs )>
<!ELEMENT PartnerLinks (aspectsOfPartnerLinks)>
<!ELEMENT aspectsOfPartnerLinks
 (PartnerLinks-PartnerLinkContainerMultiAsp )>
<!ELEMENT PartnerLinks-PartnerLinkContainerMultiAsp (PartnerLink*)>
<!ELEMENT PartnerLink (aspectsOfPartnerLink)>
<!ELEMENT aspectsOfPartnerLink (PartnerLink-PartnerLinkStructureDec )>
<!ELEMENT PartnerLink-PartnerLinkStructureDec
 ( ComponentServiceMethodPair ,  Ports )>
<!ELEMENT ComponentServiceMethodPair ( #PCDATA)>
<!ELEMENT Ports (aspectsOfPorts)>
<!ELEMENT aspectsOfPorts (Ports-PortContainerMultiAsp )>
<!ELEMENT Ports-PortContainerMultiAsp (Port*)>
<!ELEMENT Port (Port-PortTypeSpec)>
<!ELEMENT Port-PortTypeSpec (Inport | Outport )>
<!ELEMENT Inport ( #PCDATA)>
<!ELEMENT Outport ( #PCDATA)>
<!ATTLIST Ports-PortContainerMultiAsp
numContainedInPorts CDATA #IMPLIED>
<!ATTLIST PartnerLinks-PartnerLinkContainerMultiAsp
numContainedInPartnerLinks CDATA #IMPLIED>
<!ATTLIST PartnerLinks
PortCount CDATA "unknown"
ComponentCount CDATA "unknown">
<!ELEMENT WorkflowMode (WorkflowMode-WorkflowTypeSpec)>
<!ELEMENT WorkflowMode-WorkflowTypeSpec
  (While | HoldSend | Sequence | Concurrent )>
<!ELEMENT While (aspectsOfWhile)>
<!ELEMENT aspectsOfWhile (While-WhileStructureDec )>
<!ELEMENT While-WhileStructureDec ( Condition )>
<!ELEMENT Condition ( #PCDATA)>
<!ELEMENT HoldSend ( #PCDATA)>
<!ATTLIST HoldSend
holdTime CDATA "unknown">
<!ELEMENT Sequence ( #PCDATA)>
<!ATTLIST Sequence
count CDATA "unknown"
```

```
order CDATA "unknown">
<!ELEMENT Concurrent ( #PCDATA)>
<!ATTLIST Concurrent
List CDATA "unknown">
<!ELEMENT ComponentServiceMethodPairs
 (ComponentServiceMethodPairs-ComponentTypeSpec,
 aspectsOfComponentServiceMethodPairs)>
<!ELEMENT ComponentServiceMethodPairs-ComponentTypeSpec (Src | Dest )>
<!ELEMENT Src ( #PCDATA)>
<!ELEMENT Dest ( #PCDATA)>
<!ELEMENT aspectsOfComponentServiceMethodPairs
  (ComponentServiceMethodPairs-
ComponentSMPairContainerMultiAsp )>
<!ELEMENT ComponentServiceMethodPairs-ComponentSMPairContainerMultiAsp
 (ComponentServiceMethodPair*)>
<!ATTLIST ComponentServiceMethodPairs-ComponentSMPairContainerMultiAsp
numContainedInComponentServiceMethodPairs CDATA #IMPLIED>
<!ELEMENT ServiceMethods (aspectsOfServiceMethods)>
<!ELEMENT aspectsOfServiceMethods
  (ServiceMethods-MethodContainerMultiAsp )>
<!ELEMENT ServiceMethods-MethodContainerMultiAsp (ServiceMethod*)>
<!ELEMENT ServiceMethod (aspectsOfServiceMethod)>
<!ELEMENT aspectsOfServiceMethod (ServiceMethod-InfoStructureDec )>
<!ELEMENT ServiceMethod-InfoStructureDec ( InfoExchanges )>
<!ELEMENT InfoExchanges ( #PCDATA)>
<!ATTLIST ServiceMethod
messageCount CDATA "unknown"
parameters CDATA "unknown"
methodName CDATA "unknown">
<!ATTLIST ServiceMethods-MethodContainerMultiAsp
numContainedInServiceMethods CDATA #IMPLIED>
<!ELEMENT SystemDefinitions (aspectsOfSystemDefinitions)>
<!ELEMENT aspectsOfSystemDefinitions
  (SystemDefinitions-SystemContainerMultiAsp )>
<!ELEMENT SystemDefinitions-SystemContainerMultiAsp (SystemComponent*)>
<!ELEMENT SystemComponent
  (SystemComponent-SystemTypeSpec,aspectsOfSystemComponent)>
<!ELEMENT SystemComponent-SystemTypeSpec (USERAGENT | WSDL )>
<!ELEMENT USERAGENT ( #PCDATA)>
<!ELEMENT WSDL ( #PCDATA)>
<!ELEMENT aspectsOfSystemComponent
  (SystemComponent-SystemStructureDec )>
<!ELEMENT SystemComponent-SystemStructureDec ( URL )>
<!ELEMENT URL ( #PCDATA)>
<!ATTLIST SystemDefinitions-SystemContainerMultiAsp
```

```
numContainedInSystemDefinitions CDATA #IMPLIED>
<!ELEMENT WSWF-INFODec ( Messages )>
<!ELEMENT Messages (aspectsOfMessages)>
<!ELEMENT aspectsOfMessages (Messages-MessageContainerMultiAsp )>
<!ELEMENT Messages-MessageContainerMultiAsp (Message*)>
<!ELEMENT Message (Message-MessageTypeSpec,aspectsOfMessage)>
<!ELEMENT Message-MessageTypeSpec (InputMsg | OutputMsg )>
<!ELEMENT InputMsg ( #PCDATA)>
<!ATTLIST InputMsg
SystemComponent CDATA "unknown">
<!ELEMENT OutputMsg ( #PCDATA)>
<!ELEMENT aspectsOfMessage (Message-MessageStructureDec )>
<!ELEMENT Message-MessageStructureDec ( DATA )>
<!ELEMENT DATA ( #PCDATA)>
<!ATTLIST Message
timeOfStart CDATA "unknown"
is_Random CDATA "unknown"
is_Periodic CDATA "unknown"
period CDATA "unknown">
<!ATTLIST Messages-MessageContainerMultiAsp
numContainedInMessages CDATA #IMPLIED>
```

6.2 Mapping of DEVS, BPEL and DoDAF artifacts with WSWF Formalism

The WSWF information set can very well be extracted from the DoDAF information set. WSWF formalism has also been mapped to XML-Based Finite Deterministic DEVS (XFDDEVS) [25],[46] atomic and coupled models. XFD-DEVS is defined by the following tuple:

$$\begin{aligned}\text{Atomic } XFDDEVS = \; & < incomingMessageSet, \\ & outgoingMessageSet, \\ & StateSet, \\ & TimeAdvanceTable, \\ & InternalTransitionTable, \\ & ExternalTransitionTable, \\ & OutputTable >\end{aligned}$$

$$\begin{aligned}\text{Coupled } XFDDEVS = \; & < ModelSet, \\ & CouplingSet >\end{aligned}$$

The table below shows the mapping with BPEL as well. Although mapping to WSWF to BPEL is in early stages, WSWF does have the information set that is required to generate a BPEL file and the associated WSDL file as well. The code to DEVS models has been autogenerated using technologies like JAXB and XSLT. The autogenerated code provides us the DEVS skeleton in platform independent implementation in XML which could be transformed to platform specific implementation in Java, C++ or C#. More information on platform independent DEVS model generation can be seen at [25]. This skeleton can be easily augmented for any run-time capabilities. Providing detailed code implementations have been retained for brevity.

Table 2. WSWF Mapping with DoDAF, FDDEVS, and BPEL

WSWF	**DoDAF**	**XFDDEVS**	**BPEL**
W	OV-2, OV-4, SV-4	ModelSet	Process
M	OV-5, OV-6	StateSet, ExternalTransitionTable	Basic Activities, PartnerLink-PortType definitions
F			
C	W, M, OV-2, OV-8	ExternalTransitionTable paramss, InternalTransitionTable params	PartnerLink params, source and target specs in both basic and structured activities
L	SV-2	CouplingSet	PartnerLinks
D	SV-5, OV-5, OV-6	ExternalTransitionTable, InternalTransitionTable	StructuredActivities
X	SV-6, OV-3	ExternalTransitionTable, OutputTable	Messages in accompanying WSDL

The WSWF formalism is a new way to compose web service workflows that is expressed in SES-XML methodology. Since it is expressed in XML, it can be mapped easily to XPDL and possibly BPEL too. Since it is largely textual, it can retrieve information from static DoDAF documents as per Table 1. This detailed mapping, however, is not the focus of the current research and will be reported in our forthcoming publication. Going further in our development and execution of this workflow, the following sequential process provide the needed steps in order to do performance evaluation using DEVS test models [26],[24] or execution using DEVS/SOA framework as a real application. In terms of net-ready capability testing, what is required is the communication of live web services with those of test-models designed specifically for them. The approach is:

1. Specify the scenario using WSWF
2. Develop the DEVS model (or generate DEVS workflow)
3. Auto-generate the test-models from DEVS models (using DUNIP as described in [26])

4. Run the model and test-model over SOA (as per DUNIP)
5. Execute as a real-time simulation with live web services embedded in DEVS
6. Execute the test-models with real-world web services (live)
7. Compare the results of steps 5 and 6 to perform verification and validation.

7 Case Study

This case study is divided into two parts:

1. The first study demonstrates the execution of a web service encapsulated in a DEVS wrapper Agent and the associated obtained statistics.
2. The second study extends the first study by developing a workflow that utilizes more than one web services in a workflow manner. It demonstrates the following:
   - Observe user activity with DEVS Agent via WSDL-based access to collaborative service
   - Deploy DEVS virtual user models to simulate humans in collaboration scenario with human user in the loop
   - Show how DEVS agent observers communicate with other DEVS agent via DEVS/SOA infrastructure

7.1 DEVS Wrapper Agent

In this most basic demonstration, we use only one web service. This web service executes a chat session between two users. The schematic is shown in Figure 17. In our example, we execute the session with a live person and a DEVS agent. The live person here is Jim Client that connects to the CHAT service via an Internet browser at [6]. The chat session is executed using the GUI as shown in Figure 18.

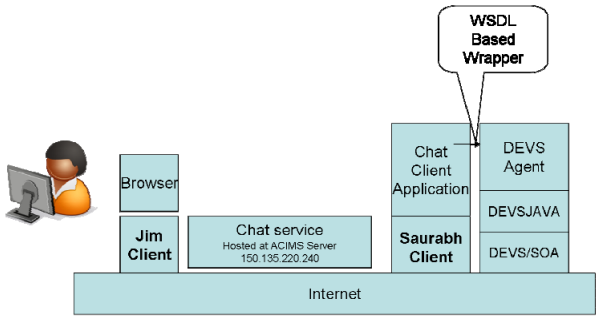

Fig. 18. Schematic showing basic execution of DEVS Wrapper Agent

The DEVS agent is defined according to the WSWF formalism as follows:

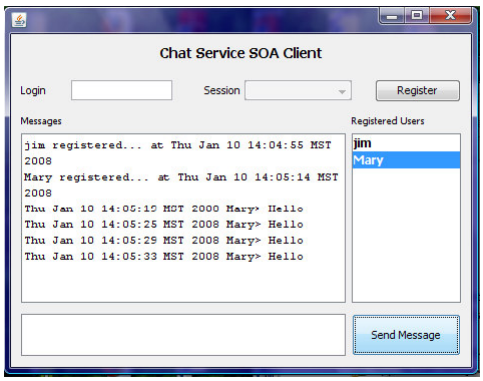

Fig. 19. Chat service client engaged with another chat particpant

```
<W>: CHAT:
<W1:CHAT>:http://150.135.220.240:8080/ChatServiceCollaboration
/services/ChatService?wsdl
<A1:Jim>: Jim:localhost:8080
<M>: Methods:
<M1> postMessage()
<M2> getAllMessages()
<M3> getLastMessageId()
<M4> registerAuthor()
<M5> getUsers()
<M6> getAllMessagesForAuthor()
<F>:"Flow specifications"
<C>
<C1:Src>A1-M1
<C2:Src>A1-M2
<C3:Src>A1-M4
<C4:Src>A1-M5
<C5:Dest>W1-M1
<C6:Dest>W1-M2
<C7:Dest>W1-M4
<C8:Dest>W1-M5
<L>
<L1>C1-C5
<L2>C2-C6
<L3>C3-C7
<L4>C4-C8
<D>
<D1>M1-HoldSend
<D2>M2-While-infinity
```

```
<D3>M4-HoldSend
<D4>M5-While-infinity
<X>: Set of Messages
<InputMsg>
<I1>W1-M1{string:T1:0:false:false}
<I2>W1-M4{string:T0:0.1:true:false}
<OutputMsg>
<O1>M2{string:T2:1:true:false}
<O2>M5{string:T2:1:true:false}
```

<W> tag contains description of the Chat Web Service as W1 and the agent description as A1 along with their URL. <M> contains the list of service methods that may be used in the process flow. <F> contains the flow description categorized into <C,L,D> as per the WSWF. <C> provides the source and destination specification for a W/A defined in <W> with <M>. <L> specifies the coupling between the sources and destinations as defined in <C>. <D> contains the execution of methods in <M> in logic implementation. For example, <D1>M1-HoldSend implies that the method M1 is to executed in HoldSend manner. Similarly, <D2>M2-While-infinity implies that M2 will be executed indefinitely when invoked or bounded by any condition. <X> specifies the input and output message structures in <InputMsg, OutputMsg> tags. The structure of <InputMsg> as specified in WSWF SES is <SystemComponent-Method{Data: time of Start: R+: isPeriodic: isRandom>. For example, the specification <I1>W1-M1string:T1:0:false:false implies that the message I1 is an input to W1, method M1 with data as string. It starts at T1 with period 0. Any non-zero value means that the message will be incoming at a periodic rate. The next boolean variable false implies that it is not periodic. The last variable false implies that it is not random either. Similary, <I2>W1-M4string:T0:0.1:true:false implies that M4 at W1 is to be invoked by string data message with a periodic rate of 0.1. The <OutputMsg> has a similar structure except the fact that it does not contain any information about the system component. It only contains information about the method in <M> as it is just an output message. Whenever method <Mx> is invoked, it returns with the parametric details as in <O1>M2string:T2:1:true:false.

It is worth stressing here that the messages flow through the linkages as specified in <L>. This acts as a coupling for the DEVS models. There are two DEVS models in the WSWF instance described above, viz. W1 and A1. Based on the coupling information for ex. <L4>C4-C8 implies that the source is Agent <C4:Src>A1-M5 and the destination is Web service <C8:Dest>W1-M5. The source sends a message invoking method M5 at the destination. If there is a specification on how M5 should be invoked in <InputMsg> listing, then the source has to ensure that it conforms to that specification. In this example there is no specification for M5. This implies that there are no parameters to be passed, but just the invocation. At the destination side, M5 has a specification <O2>M5string:T2:1:true:false, which implies that whenever M5 returns a value, it will according to this <OutputMsg> specification.

The statistics for each of the methods in <M> is gathered according to the autogenerated agent GUI monitor at the agents end. The statistics are largely the round trip time (RTT) for each of <M>. The GUI in Figure 20 also shows the SOAP messages that are exchanged between the pairs as specified in <W>.

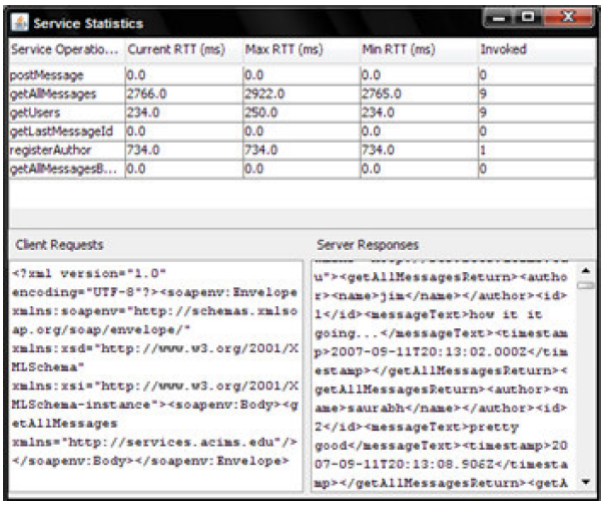

Fig. 20. Associated statistics GUI for an encapsulated Web Service in DEVS Atomic Model

7.2 Workflow design, Analysis and Execution

The previous demonstration has established that we can encapsulate a live web service within a DEVS atomic model using an XML based formalism such as WSWF. It also establishes that we can create virtual users as DEVS agents that input and communicate with live users. Having such capability allows us to build upon the advances of DEVS hierarchical component based modeling and simulation. In the next demonstration, we will build a workflow with two live web services and all the clients as virtual users.

DEVSJAVA execution on a single machine The first service is the same CHAT service and the second service is a weather service [37]. In this demonstration, we will show that virtual users are engaged in chat session and one user requests weather from another user. The second user (Jim Client) shown in Figure 21 requests the weather from the Weather web service and reports it back to the first user using the CHAT service. We will then also execute the entire scenario as a self-contained coupled model on DEVS/SOA with these virtual agents deployed at different IP addresses. The schematic is shown in Figure 21.

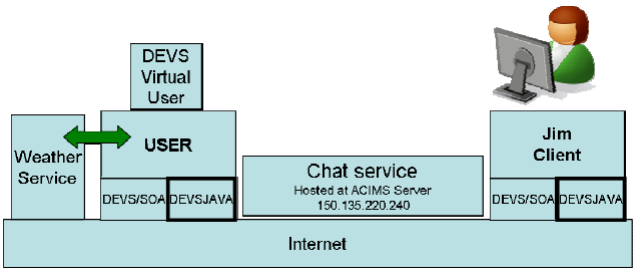

Fig. 21. Schematic of Workflow scenario with two virtual DEVS agents

The workflow according to WSWF formalism is defined as follows:

```
<W>: CHAT-and-WEATHER:
<W1:CHAT>:<http://150.135.220.240:8080/ChatService Collaboration/
services/ChatService?wsdl>
<W2:WEATHER>: http://www.webservicex.net/WeatherForecast.asmx.
<A1:JIM>: Jim:localhost:8080
<A2:USER>: User:localhost:8080
<M>: Methods:
<M1> postMessage()
<M2> getAllMessages()
<M3> getLastMessageId()
<M4> registerAuthor()
<M5> getUsers()
<M6> getWeather()
<F>:"Flow specifications"
<C>
<C1:Src>{A1,A2}-M1
<C2:Src>A1-M2
<C3:Src>{A1,A2}-M4
<C4:Src>A1-M5
<C5:Src>A2-M6
<C6:Dest>W1-M1
<C7:Dest>W1-M2
<C8:Dest>W1-M4
<C9:Dest>W1-M5
<C10:Dest>W2-M6
<L>
<L1>C1-C6 //notice that both A1 and A2 are coupled to W1-M1
<L2>C2-C7
<L3>C3-C8 //notice that both A1 and A2 are coupled to W1-M4
<L4>C4-C9
<L5>C5-C10
<D>
```

```
<D1>M1-HoldSend:5
<D2>M2-While-infinity
<D3>M4-HoldSend
<D4>M5-While-infinity
<D5>M6-HoldSend
<X>: Set of Messages
<InputMsg>
<I1>W1-M1{string:T1:5:true:false}
<I2>W1-M4{string:T0:0.1:true:false}
<I3>W2-M6{string:T3:0:false:false}
<I4>A2-M1(string:T4:0:false:false)
<OutputMsg>
<O1>M2{string:T2:1:true:false}
<O2>M5{string:T2:1:true:false}
<O3>M6{string:T3:0:false:false}
where, T0 > 0, T1&T2 > 0, and T3,T4>T1,T2.
```

The description of WSWF instance above is on the same lines as of previous example. However, instead of just one, there are two services in this instance as specified by `<W1>` and `<W2>`. The two services are: the `Chat Service` and the publically available `Weather service`. There is an addition method `<M6>` that invokes the `Weather service`. There are two agents viz., `Jim` and `USER`. The `USER` is a virtual user and is modeled as a DEVS Agent and `Jim` is a live person. A DEVS agent is a computer program implemented as a DEVS model. It is engaged in chat session with `Jim` and reports back the results of `Weather service` when the request to invoke comes from `Jim`, the real user.

The demonstration has been structured in a manner that it be executed in a single machine. To execute it on remote machines we will be using DEVS/SOA which is described in the next sub-section. To execute it on a single machine, DEVSJAVA platform is sufficient. Figure 22 shows the virtual user USER in black console and the `Jim` real user in the Chat window. Notice that the Jim client also has the monitor running that invokes method `<M4>` and `<M5>` at the `Chat Web service`. The GUI also shows the DEVSJAVA simulation viewer which shows that DEVSJAVA is being used to run the scenario. The Jim client requests weather from USER client. The `USER` invokes `<W2>` web service, and reports back the result by method `<M1>` to the `Chat Service`.

To provide complete performance analysis of the workflow as per the GUI in Figure 22 is outside the scope of the paper and has been retained.

Execution on DEVS/SOA Framework The scenario remains the same as in preview sub-section. However, the execution is made on DEVS/SOA platform (Figure 23). The real user Jim has now been replaced by another virtual client. The only modification in the WSWF instance is the following:

```
<W>: CHAT-and-WEATHER:
```

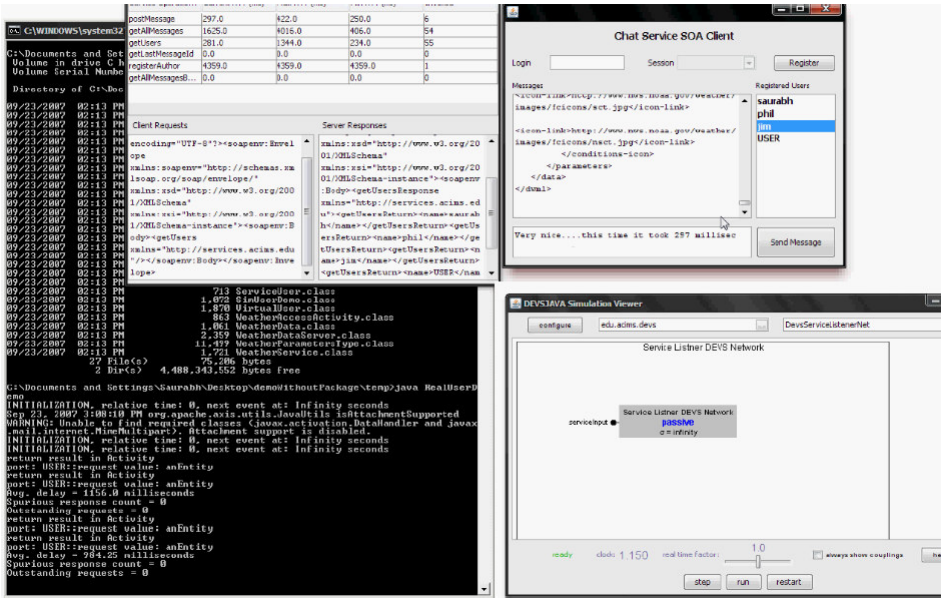

Fig. 22. Snapshot of execution of workflow case study as depicted in Figure 21

```
<W1:CHAT>:<http://150.135.220.240:8080/ChatService Collaboration/
services/ChatService?wsdl>
<W2:WEATHER>: http://www.webservicex.net/WeatherForecast.asmx.
<A1:USER1>: User1:150.135.218.205:8080
<A2:USER2>: User2:150.135.220.240:8080
```

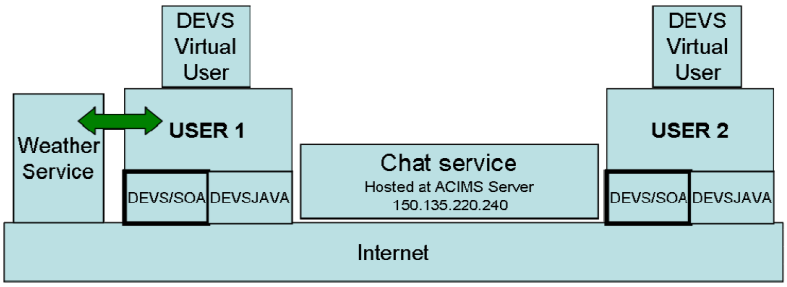

Fig. 23. Execution of Workflow scenario with DEVS/SOA framework

The generated Java code is fed to the DEVS/SOA client GUI as reproduced again in Figure 24. `USER2` in the generated code is given the Class name

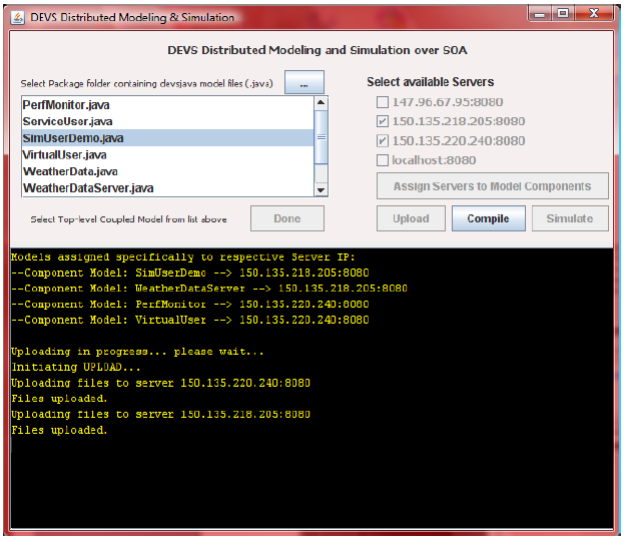

Fig. 24. Models package being executed using DEVS/SOA client

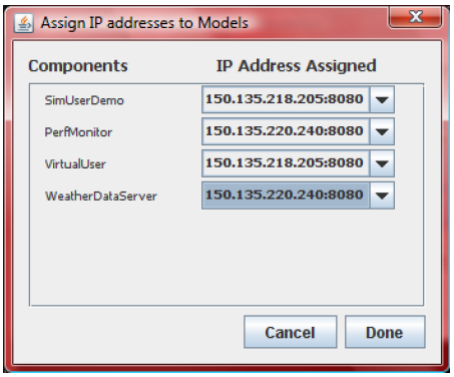

Fig. 25. IP assignment of models for Workflow scenario

PerfMonitor for differentiation. The Class VirtualUser is USER1. The USER1 is assigned an IP address 150.135.218.205:8080 and USER2 at 150.135.220.240:8080. These virtual users are then sent to these respective IP addresses. These IP addresses provide the DEVS Simulation service and Apache Tomcat servers are used as containers at these IPs. The other dependent files are also uploaded at corresponding IPs. The assignment can be done manually as shown in Figure 25. Once uploaded, the files are compiled at run-time at the servers end and a distributed simulation is executed between these remote machines. Once the simulation is over, the result is communicated back into the console window as shown in Figure 24. The detailed output of the simulation run is shown below. As can be seen from the output below, VirtualUser sent three requests and got responses with different delays. The responses are communicated by the other USER2 after invoking the Weather service. The result is also sent back to VirtualUser, as,

```
Place Name:HILLSIDE MANOR;MinTempC:-4;MaxTempC:5.

Models assigned specifically to respective Server IP:
--Component Model: SimUserDemo --> 150.135.218.205:8080
--Component Model: VirtualUser --> 150.135.218.205:8080
--Component Model: PerfMonitor --> 150.135.220.240:8080
--Component Model: WeatherDataServer --> 150.135.220.240:8080
Uploading in progress... please wait...
Initiating UPLOAD...
Uploading files to server 150.135.220.240:8080
Files uploaded.
Uploading files to server 150.135.218.205:8080
Files uploaded.
Compilation in progress....please wait....
Starting compilation at remote servers.....
Compiling project at 150.135.220.240:8080...
Success: true
Project compiled.

Compiling project at 150.135.218.205:8080...
Success: true
Project compiled.
Waiting to start SIMULATION....
Simulation in Progress....please wait...
Running simulation ...
21 iterations.
Simulators output:
150.135.220.240 output:
VirtualUser: sent a request
Avg. delay = 375.0 milliseconds
Spurious response count = 0
```

```
Outstanding requests = 0
VirtualUser: response length 48
Place Name:HILLSIDE MANOR;MinTempC:-4;MaxTempC:5
VirtualUser: sent a request
Avg. delay = 355.25 milliseconds
Spurious response count = 0
Outstanding requests = 0
VirtualUser: response length 48
Place Name:HILLSIDE MANOR;MinTempC:-4;MaxTempC:5
SIMULATION over!
```

Hybrid execution using DEVS/SOA and DEVSJAVA Once we have such DEVS coupled workflow system, we can extend this system by replacing any virtual user with a live user. Figure 26 below shows the schematic of such operation and a demonstration is made available as an .avi file at [8]. In the schematic below the DEVS coupled system is augmented with other DEVS agents for reporting statistics etc, basically the idea being, such DEVS enabled workflows can now participate in live modeling and simulation exercises in real-time.

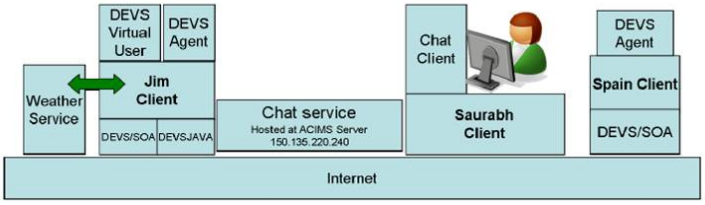

Fig. 26. WSWF formalism based workflow using DEVS as middleware for live modeling and simulation exercises

8 Agility in DEVS Unified Process

Agile software methodologies have taken quite a notice these recent years primarily due to the factors such as volatile ever-changing requirements, dynamic technological landscape, high employee turnover, and most importantly, satisfying the business needs. [50] summarizes it as:

> Agile development is not defined by a small set of pratices and techniques but defines a strategic capability, a capability to create and respond to change, a capability to balance flexibility and structure, a capability to draw creativity and innovation out of a development team, and a capability to lead organizations through turbulence and certainty.

There is a fundamental shift in the approach of delivering the product by hard-line requirements specifications supported by methodologies like Capability Maturity Model (CMM) and CMM Integration (CMMI) and the Agile practices. While the former delineates defined repeatable processes so that the performance can be measured within very close tolerances, the latter is more geared towards employing the latest advancement in technologies, to explore, and to deliver the product as soon as possible. The key point of agile practices is the inclusion of software engineering life cycle in each iteration so that the features delivered are production ready at the end of each iteration. While the vision of most projects are clear, what remains fuzzy are the exact requirement specifications that the developers are faced with. With agile practices, a constant dialogue with the customer, repeatable testing procedures, incremental development and using the latest technology, the requested feature can be delivered in the next iteration without changing the entire project vision. The DEVS Unified Process, similarly is based on agile methodology. Table 3 lists the similarities with each phase of agile development methodology [51].

Table 3 establishes that DEVS Unified Process has all the needed phases of being agile and the model continuity [52] enables any DEVS artifiact to be a real software. With DUNIP's SOA edge, we have any DEVS model that is available as a web service. Modeling and Simulation in today's world is more than just a software. It is an enabling technology that has far reaching impact on any nations' progress and advance the forefront of various technologies in many domains such as biology, chemistry, phyiscs, space science, etc. While there are customized M&S software for different problem sets and different domains, an agile methodology is another ace that when employed could incorporate the latest advancements in software engineering discipline and apply it to the M&S solution at hand.

9 Conclusions and Future Work

Service Oriented Architecture (SOA) have come a long way and many of the businesses are seriously considering migration of their IT systems towards SOAs. DoDs initiative towards migration of GIG/SOA and NCES requires reliability and robustness, not only in the execution but in the design and analysis phase as well. Web service orchestration is not just a research issue but a more practical issue for which there is dire need. Further, Service Oriented Architecture must be taken as another instance of system engineering for which there must be a laid out engineering process. Modeling and Simulation provides the needed edge. Lack of methodologies to support design and analysis of such orchestration (except BPEL related efforts) cost millions in failure. This research has proposed that Discrete Event Formalism can be used to compose and analyze Web service workflows. The DEVS theory, which is based on system theoretic concepts, gives solid grounding in the modeling and simulation domain.

The prime motivation of applying DEVS system theoretical principles to these emerging net-centric systems comes from an editorial by Carstairs [53]

Table 3. Agile Methodology and DEVS Unified Process

Phase	Agile Methodology	DEVS Unified Process (DUNIP)
Model	Identify the domain and business use-case requirements and specify in domain specific languages such as UML, etc.	DUNIP begins by taking requirements in different formats like DoDAF, UML, State-based, NLP and transform them into platform independent XML models
Implementation	Transform your models into executable code with running unit-tests	From PIMs, the DUNIP engine generates code in platfrom specific models (PSMs) such as Java, C++, C# etc. With strong DEVS theory underlying each of the atomic models, the models can be verified mathematically, as well as graphical with various DEVS toolsets such as DEVSJAVA. Unit-testing for each transition or an event is inherent in DEVS.
Test	Identify defects, ensure quality and verify requirements	With DUNIP, the development of test suite is done in parallel with that of the model. Test models are generated from the XML-based PIMs. The test models verify the atomic model's operation at various levels of system specifications, such as I/O pair, I/O function, etc. The Experimental Frames are also designed at this stage that ensure the requirements are met through the test models.
Deployment	Plan the delivery and make it available to end users	With ready deployment capabilities per model-continuity principles to SOA infrastructure, and zero transition times, the model is the actual software and is readily moved to the production servers
Configuration Management	Managed access to project artifacts	DUNIP is very well positioned to reuse and contribute to Model repository. PIMs are a strong contender for such tracking and version management. PSMs can very well be source versioned using tools like Subversion etc.
Project Management	Manage people, project, iterations and budget	These qualities are universal and due to the component nature of DEVS technology, the project plan can very well be partitioned into iterative cycles and milestones
Environment	Ensure that proper process, guidance, and tools are available for the team	DEVS has been in existence for over 30 years and there is a large community that is available for support in basic theory and toolsets.

that demands a M&S framework at higher levels of system specifications where System of systems interact together using net-centric platform. At this level, model interoperability is one of the major concerns. The motivation for this work stems from this need of model interoperability and the characterists of net-centric systems that are easier to simulate, test and deploy with an underlying foundation of systems engineering principles. DEVS, which is known to be component-based system, based on formal systems theoretical framework is the preferred means. Table 4 outlines how it could provide solutions to the challenges in net-centric design and evaluation.

Table 4. Solutions provided with DEVS Technology in support of M&S for T&E

Desired M&S capability for Test and Evaluation (T&E)	**Solutions provided by DEVS M&S technology**
Support for DoDAF need for executable architectures using M&S such as mission based testing for GIGSOA	DEVS Unified Process [24],[29] provides methodology and SOA infrastructure for integrated development and testing, extending DoDAF views [20].
Interoperability and cross-platform M&S using GIG/SOA	Simulation architecture is layered to accomplish the technology migration or run different technological scenarios [54]. Provide net-centric composition and integration of DEVS 'validated' models using Simulation Web services [22].
Automated test generation and deployment in distributed simulation	Separate a model from the act of simulation itself, which can be executed on single or multiple distributed platforms [39]. With its bifurcated test and development process, automated test generation is integral to this methodology [42].
Test artifact continuity and traceability through phases of system development	Provide rapid means of deployment using model-continuity principles and concepts like 'simulation becomes reality' [52].
Real-time observation and control of test environment	Provide dynamic variable structure component modeling to enable control and reconfiguratin of simulation on the fly [55],[56]. Provide dynamic simulation tuning, interoperability testing and benchmarking

The net-centric DEVS framework as laid out in this chapter required enhancement to these basic DEVS capabilities. Furthermore, this work describes distributed simulation using the web service technology. After the development of World Wide Web, many efforts in the distributed simulation field have been made for modeling, executing simulation and creating model libraries that can be assembled and executed over WWW. By means of XML and web services technology these efforts have entered upon a new phase. The proposed DEVS Modeling Language (DEVSML) is built on eXtensible Markup Language (XML) as the preferred means to provide such transparent simulator implementation. A prototype simulation framework called DEVS/SOA has been implemented us-

ing web services technology. It is currently at the forefront of DEVS net-centric research platform [47]. The central point resides in executing the simulator as a web service. The development of these kind of frameworks will help solve large-scale problems and guarantees interoperability among different networked systems and specifically DEVS-validated models. This chapter focused on the overall approach, and the symmetrical SOA-Based architecture that allows for DEVS execution as a Simulation SOA.

We have shown how a web service can be encapsulated into a DEVS atomic model and can be put towards a coupled DEVS system with other live web services as well as other DEVS models. We also have demonstrated the proposed use of Web Service Work Flow (WSWF) formalism in composing SOA, much like the same functionalities of BPEL. We have also described how WSWF can be mapped to DoDAF elements using the System Entity Structure (SES) and could achieve creation of DEVS net-centric coupled systems based on SOA. We have also shown how the developed DEVS coupled system can be simulated using the basic DEVSJAVA framework as well as distributed DEVS/SOA framework. Further, on the basis of our earlier work on DEVS/SOA we have basis for:

- Agent-Implemented Test Instrumentation
- Net-centric Execution using Simulation Services
- Distributed Multi-level Test Federations
- Analysis that can help optimally tune the instrumentation to provide confident scalability predictions.
- Mission Thread testing and data gathering:
  - definition and implementation of military-relevant mission threads to enable constructing and/or validating models of user activity.
  - Comparison with current commercial testing tools shows that by replicating such models in large numbers it will be possible to produce more reliable load models than offered by conventional use of scripts.

We have taken the challenge of constructing net-centric systems as one of designing an infrastructure to integrate existing Web services as components, each with its own structure and behavior with DEVS components and agents. The net-centric system is analogous to a System of System (SoS) where in hierarchical coupled models could be created. Various workflows can be integrated together using component based design. The net-centric system can be specified in many available frameworks such as DoDAF, SES, BPMN/BPEL, UML, or by using an integrative systems engineering-based framework such as DEVS. The proposed Web Service Work Flow formalism binds various frameworks like SES, BPEL, DoDAF and DEVS.

In this research, we have discussed the advantages of employing an M&S-integrated framework such as DEVS Unified Process (DUNIP) and its supporting DEVS/SOA infrastructure. We illustrated how M&S can be used strategically to provide early feasibility studies and aid the design process. We have established the capability to develop a live workflow example with complete DEVS interface. In this role, DEVS acts as a full net-centric production environment. Being DEVS enabled, it is also executable as a system under test (SUT) model towards

various verification and validation analysis that can be performed by coupling this SUT with other DEVS test models. Last but not the least, the developed DEVS system can be executed by both real and virtual users to the advantage of various performance and evaluation studies. We also summarized how DUNIP is agile and each of its modules fit to the agile practices.

As components comprising SoS are designed and analyzed, their integration and communication is the most critical part that must be addressed by the employed System of System (SoS) M&S framework. We discussed DoDs Global Information Grid (GIG) as providing an integration infrastructure for SoS in the context of constructing collaborations of web services using the Service Oriented Architecture (SOA). The DEVS Unified Process (DUNIP), in analogy to the Rational Unified Process based on UML, offers a process for integrated development and testing of systems that rests on the SOA infrastructure. The DUNIP perspective led us to formulate a methodology for testing any proposed SOA-based integration infrastructure, such as DISAs Net-Centric Enterprise Services. The present research is being considered and refined for standardization with the DEVS Standardization group [47],[48],[49]. Clearly, the theory and methodology for such net-centric SoS development and testing are at their early stages.

References

1. ACIMS software site: `http://www.acims.arizona.edu/SOFTWARE/software.shtml` Last accessed Sep. 2010
2. Atkinson, K, "Modeling and Simulation Foundation for Capabilities Based Planning", Simulation Interoperability Workshop Spring 2004
3. Badros, G. "JavaML: a Markup Language for Java Source Code", Proceedings of the 9th International World Wide Web Conference on Computer Networks: the international journal of computer and telecommunication networking, pages 159-177
4. Business Process Execution Language `http://www.ibm.com/developerworks/library/specification/ws-bpel/`
5. Business Process Modeling Notation `http://www.bpmn.org`
6. CHAT SOA web service at `http://www.saurabh-mittal.com/demos/ChatClient`
7. Cheon, S, Seo, S, Park, S, Zeigler, BP, "Design and Implementation of Distributed DEVS Simulation in a Peer to Peer Networked System", Advanced Simulation Technologies Conference, Arlington, VA, 2004
8. Chat-Weather Service Demo as .avi file at `http://duniptechnologies.com/training/demos/DEVS_CHAT_Weather_RealUserDemo.avi`
9. DoDAF Working Group , DoD Architecture Framework Ver. 1.0 Vol. III: Deskbook, DoD, Aug. 2003.
10. DOD Instruction 5000.2 Operation of the Defense Acquisition System, 12 May 2003.
11. DoD Architecture Framework Working Group 2004, DOD Architecture Framework Ver. 1.0 Volume 1 Definitions and Guidelines, 9 February 2004, Washington, D.C.
12. DUNIP: A Prototype Demonstration `http://duniptechnologies.com/training/demos/dunip.avi`
13. Fujimoto, RM, "Parallel and Distribution Simulation Systems", Wiley, 1999

14. Joint Interoperability Test Command, a Defense Information Systems Agency http://jitc.fhu.disa.mil/
15. Martin, JLR, Mittal, S, Zeigler, BP, Manuel, J, From UML Statecharts to DEVS State Machines using XML, IEEE/ACM conference on Multi-paradigm Modeling and Simulation, Nashville, September 2007
16. Department of Defense GIG Architectural Vision, Ver. 1.0, prepared by DoD CIO, June 2007, available at: http://www.defenselink.mil/cio-nii/docs/GIGArchVision.pdf
17. Kim, K, Kang, W, "CORBA-Based Multi-threaded Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-hierarchical One", International Conference on Computational Science and Its Applications, ICCSA, Italy 2004
18. Mak, E, Automated Testing using XML and DEVS, Thesis, University of Arizona, http://www.acims.arizona.edu/PUBLICATIONS/PDF/Thesis_EMak.pdf
19. Mak, E, Mittal, S, Hwang, MH, Automating Link-16 Testing using DEVS and XML, Journal of Defense Modeling and Simulation, 2008
20. Mittal, S, "Extending DoDAF to Allow DEVS-Based Modeling and Simulation", Special issue on DoDAF, Journal of Defense Modeling and Simulation JDMS, Vol 3, No. 2.
21. Mittal, S, Mak, E, Nutaro, JJ, "DEVS-Based Dynamic Modeling & Simulation Reconfiguration using Enhanced DoDAF Design Process", special issue on DoDAF, Journal of Defense Modeling and Simulation, Dec 2006
22. Mittal, S, Martin, JLR, Zeigler, BP, DEVSML: Automating DEVS Simulation over SOA using Transparent Simulators, DEVS Syposium, 2007
23. Mittal, S, Martin, JLR, Zeigler, BP, DEVS-Based Web Services for Net-centric T&E, Summer Computer Simulation Conference, 2007
24. Mittal, S, "DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures", Ph. D. Dissertation, University of Arizona
25. Mittal, S, Hwang, MH, Zeigler, BP, XFD-DEVS: An Implementation of W3C Schema for XML-Based Finite Deterministic DEVS, in progress, Demo available at: http://duniptechnologies.com/research/xfddevs
26. Mittal, S, Zeigler, BP, Martin, JLR, Sahin, F, Jamshidi, M, Modeling and Simulation for System of systems Engineering, chapter in Systems of Systems engineering for 21st Century, 2008
27. Mittal, S, Zeigler, BP, Martin, JLR, "Implementation of Formal Standard for Interoperability in M&S/System of Systems Integration with DEVS/SOA", International Command and Control C2 Journal, Special Issue: Modeling and Simulation in Support of Network-Centric Approaches and Capabilities, 2008
28. Mittal, S, Martin, JLR, Zeigler, BP, "DEVS/SOA: A Cross-Platform Framework for Net-Centric Modeling and Simulation in DEVS Unified Process", SIMULATION: Transactions of SCS, 2008
29. Mittal S, Zeigler BP, "DEVS Unified Process for Integrated Development and Testing of System of Systems", Critical Issues in C4I, AFCEA-George Mason University Symposium, May 2008
30. Net-Centric Enterprise Service http://www.disa.mil/nces/ last accessed Sep. 2010
31. Jon, P, " XPDL: The Silent Workhorse of BPM" , April 2007 online article http://www.bpm.com/FeatureRO.asp?FeatureId=232 last accessed Sep. 2010
32. Sarjoughian, HS, Zeigler, BP, "DEVS and HLA: Complimentary Paradigms for M&S?" Transactions of the SCS, (17), 4, pp. 187-197, 2000

33. SESBuilder, An Integrated Tool to utilize System Entity Structure, 2007, http://www.sesbuilder.com/
34. DEVS/SOA sample demonstration in .avi format http://duniptechnologies.com/training/demos/demoSOADEVS.avi
35. Simple Object Access Protocol http://www.w3.org/TR/soap/
36. Seo, C, Park, S, Kim, B, Cheon, S, Zeigler, BP, "Implementation of Distributed High-performance DEVS Simulation Framework in the Grid Computing Environment", Advanced Simulation Technologies conference (ASTC), Arlington, VA, 2004
37. Weather web service at: http://www.webservicex.net/WeatherForecast.asmx
38. Web Services Description Language http://www.w3.org/TR/wsdl
39. Zeigler, BP, Kim, TG and Praehofer, H, "Theory of Modeling and Simulation" New York, NY, Academic Press, 2000
40. DEVSJAVA: http://www.acims.arizona.edu/SOFTWARE/devsjava_licensed/CBMSManuscript.zip
41. Zhang, M, Zeigler, BP, Hammonds, P, "DEVS/RMI-An Auto-Adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies", ITEA Journal, July 2005
42. Zeigler, BP, Fulton, D, Hammonds P, and Nutaro, J, "Framework for M&SBased System Development and Testing In a Net-Centric Environment", ITEA Journal of Test and Evaluation, Vol. 26, No. 3, 21-34, 20
43. Zeigler, BP, and Hammonds, P, Modeling& Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange, 2007, New York, NY: Academic Press.
44. Zeigler, BP and Zhang, G, "The System Entity Structure: Knowledge Representation for Simulation Modeling and Design", in Artificial Intelligence, Simulation and Modeling, L. E. Widman, K. A. Loparo, and N. R. Nielsen, Eds. Wiley, p.47-73, New York, 1989.
45. Zeigler, BP, Mittal, S Enhancing DoDAF with DEVS-Based System Life-cycle Process, IEEE International Conference on Systems, Man and Cybernetics, Hawaii, October 2005
46. Martin, JLR, Mittal, S, Mendel, J, Zeigler, BP, "eUDEVS: Executable UML Using DEVS Theory of Modeling and Simulation", invited paper to SIMULATION, accepted 2009.
47. Wainer, G, Al-Zoubi, K, Dalle, O, Hill, D, Mittal, S, Martin, JLR, Sarjoughian, H, Touraille, L, Traore, M, Zeigler, BP, "Chapter 15: DEVS Standardization: Ideas, Trends and Future", Book: Discrete Event Modeling and Simulation: Theory and Applications, CRC Press, 2010.
48. Wainer, G, Al-Zoubi, K, Dalle, O, Hill, D, Mittal, S, Martin, JLR, Sarjoughian, H, Touraille, L, Traore, M, Zeigler, BP, "Chapter 17: Standardizing DEVS Model Representation", Book: Discrete Event Modeling and Simulation: Theory and Applications, CRC Press, 2010.
49. Wainer, G, Al-Zoubi, K, Dalle, O, Hill, D, Mittal, S, Martin, JLR, Sarjoughian, H, Touraille, L, Traore, M, Zeigler, BP, "Chapter 18: Standardizing DEVS Simulation Middleware", Book: Discrete Event Modeling and Simulation: Theory and Applications, CRC Press, 2010.
50. Highsmith, J., "What is Agile Software Development?", STSC Crosstalk, Journal of Defense Software Engineering, 2002
51. Ambler, SW, "The Agile Unified Process", http://www.ambysoft.com/unifiedprocess/agileUP.html

52. Hu, X, Zeigler, BP, "Model Continuity in the Design of Dynamic Distributed Real-Time System", IEEE Transactions on Systems, Man and Cybernetics - Part A, Vol. 35, 6, pp. 867-878, Nov. 2005
53. Carstairs, DJ, "Wanted: A New Test Approach for Military Net-centric Operations", Guest Editorial, ITEA Journal, Vol. 26, No. 3, Oct. 2005
54. Sarjoughian H, Zeigler BP, Hall, S, " A Layered Modeling and Simulation Architecture for Agent-Based System Development", Proceedings of IEEE Vol. 89, No.2; pp. 201-213, 2001
55. Mittal S, Zeigler, BP, Hammonds, P, Veena, M, "Network Simulation Environment for Evaluation and Benchmarking HLA/RTI Experiments", JITC report, Fort Huachuca, Dec. 2004
56. Hu, X, Zeigler, BP, Mittal, S, "Dynamic Configuration in DEVS Component-based Modeling and Simulation", SIMULATION: Transaction of the Society of Modeling and Simulation International, Nov. 03