Simulating adaptive service-oriented software systems

Mohammed A Muqsith, Hessam S Sarjoughian, Dazhi Huang and Stephen S Yau

SIMULATION 2011 87: 915 originally published online 11 October 2010
DOI: 10.1177/0037549710382431

Simulation

Simulation: Transactions of the Society of Modeling and Simulation International
87(11) 915–931

DOI: 10.1177/0037549710382431
sim.sagepub.com

Simulating adaptive service-oriented software systems

Mohammed A Muqsith[1], Hessam S Sarjoughian[1,3], Dazhi Huang[2] and Stephen S Yau[2]

Abstract

Simulation of dynamic service-based software systems is important for studying services that may change their composition and thus interactions at run-time. An approach based on Service Oriented Architecture-compliant DEVS (SOAD) and Dynamic Structure DEVS (DSDEVS) modeling approaches is developed to support structural changes in service model composition. To achieve this goal, a broker–executive model is devised based on the broker model defined for SOAD and the executive model defined for DSDEVS. The capability to simulate dynamic services is incorporated to the DEVS-Suite simulator. To demonstrate modeling of dynamic service-based software systems, a real voice communication system and a model of this system have been developed. The importance of enabling simulation-based design for adaptable systems is briefly discussed.

Keywords

adaptive service-based software systems, DEVS-Suite simulator, Dynamic Structure DEVS, SOA-compliant DEVS models

1. Introduction

Service Oriented Architecture (SOA)[1,2] is an attractive approach for developing enterprise scale distributed software systems. It emphasizes loosely coupled, protocol independent distributed system development with the 'software as service' concept - a self-contained component provided as a publishable contract for use by independent clients. SOA has evolved to address the demand to develop and deploy large-scale software systems that are cost-effective to reuse and maintain and easily adaptable to infrastructure change. A key promise of the SOA is supporting on-demand Quality-of-Service (QoS) for given business logics. Maintaining QoS, however, is a challenging task as it depends on the system architecture. Specific design decisions (e.g., flat versus hierarchial service composition) may significantly impact the run-time QoS, even though the decisions taken during the design phase are consistent with SOA. Hence, evaluation of the system design prior to development is necessary.

Simulation is widely accepted as an effective method for developing system design. It can be used to determine the run-time behavior of Service-based Software (SBS) systems (e.g., Bause,[3] Sarjoughian et al.,[4] and Tsai et al.[5]). To simulate SBS systems that are capable of simultaneously satisfying multiple QoS attributes, it is desirable to develop models that are not only grounded in a formal modeling formalism, but also are based on SOA concepts and principles. Simulation can be used for studying the time-based dynamics of systems including QoS tradeoffs. In particular, there is a need to model and simulate Adaptive SBS (ASBS)[6] where its dynamics can be observed and controlled by a monitoring system and an adaptation system, respectively. The users can select services and list desired behaviors under the presence of some

[1]Arizona Center for Integrative Modeling and Simulation, Department of Computer Science and Engineering, Arizona State University, USA.
[2]Department of Computer Science and Engineering, Arizona State University, USA.
[3]School of Electrical and Computer Engineering, University of Tehran, Iran.

Corresponding author:
Hessam S. Sarjoughian, Arizona Center for Integrative Modeling and Simulation, Department of Computer Science and Engineering, Arizona State University, Tempe, AZ, USA
Email: sarjoughian@asu.edu

uncontrollable, but predictable environmental fluctuations.

In the context of this research, we use the Discrete Event System Specification (DEVS) formalism[7] and DEVS-Suite[8] as our underlying modeling approach and simulator, respectively. Specifically, the SOA-compliant DEVS (SOAD) simulation framework has been developed to support modeling of SBS systems[4]. However, it lacks the basis for modeling structural change that is required for ASBS designs. There exist alternative methods for specifying component-based models that can change their structure and behavior at run-time.[9–11] The Dynamic Structure DEVS (DSDEVS)[9] is one of the approaches that can afford structural changes that may occur in models. The SOAD framework needs to be extended to support simulation of SBS systems that can have services added and removed dynamically.

Our contribution in this paper is the development of the DSOAD (Dynamic Structure SOAD) modeling approach by introducing dynamic structure modeling to the hierarchical SOAD modeling framework. A realization has been developed by extending the DEVS-Suite simulator with the DSDEVS simulation. To demonstrate the kinds of modeling that are needed for the design of ASBS systems, an exemplar real voice communication system is developed and compared with its simulated counterpart developed in DEVS-Suite. It is important to note that the focus of the paper is on DSOAD – a modeling approach to dynamic aspects of the SOA. It does not account for SOA as part of the simulation infrastructure and as such model interactions with real web services are not accounted for. So, all the DSOAD components (i.e. all services – publisher, subscriber, broker) are simulated and are not interchangeable with real web services.

2. Related work

Modeling and simulation research related to service-oriented computing may be broadly divided into three categories: (i) developing models that can specify simulatable web services and their interactions; (ii) distributed simulation execution using web services; and (iii) use of simulation as an aid for software development of actual web services. We provide a view of existing work in light of these categories.

In the first category, the aim is to represent the key aspects of services. In Narayanan and Mcilraith,[13] a Petri-net based approach is presented. The services are specified using the DARPA Agent Markup Language for Services (DAML-S)[14] and converted to Petri-net models and simulated using the KarmaSim environment.[15] While Petri-net provides a strong basis for verification and validation of the models, the mapping DAML-S to Petri-nets is ad hoc. In particular, the combination of DAML-S and Petri-net lacks a sound basis for describing time-based dynamics of SBS systems. In Bause,[3] the concept of process chain modeling is used to abstract web services as workflows. The approach supports modeling business processes as sequences of functional units that together represent web services. The approach treats services at the high level of processes rather than explicitly representing basic traits of services and SOA. It does not directly address the concept of publisher (provider), subscriber (client), broker and a service in general with the properties such as discoverability and statelessness, which are important for developing models that can be said to comply with SOA. In Sarjoughian et al.,[4] an approach is developed by unifying the DEVS and SOA frameworks. Based on the DEVS and SOA concepts and principles, a set of primitive and composite service model abstractions along with their interactions (i.e. message and service calls) are defined. To simulate network delay and throughput constraints among distributed services, a basic router model is also developed. The resulting SOAD framework supports simulations of SBS systems. However, there is no rigorous support for run-time service composition under dynamic scenarios (i.e. adding and removing services during simulation execution).

In the second category, the emphasis is on the adoption and extension of service-oriented computing techniques and technologies for stronger support for distributed and/or large-scale simulations. In Mittal et al.,[16] the DEVS approach is extended to support simulation of models as web services. The resulting DEVS/SOA environment, unlike DEVS/HLA[17] for example, allows DEVSJAVA[18] simulators to be treated as services. Grid computing as a simulation environment is also explored. Cosim-Grid[19] is a service-oriented simulation grid based on High Level Architecture (HLA),[20] PLM (Product Lifecycle Management),[21] and grid/web services. It applies OGSA (Open Grid Services Architecture)[22] to modeling and simulation to improve HLA in terms of dynamic sharing, autonomy, fault tolerance, collaboration, and security mechanisms. The XMSF (Extensible Modeling and Simulation Framework)[23] has also been developed using a set of technologies including web/XML, web services, and internet/networking for web-enabled simulation.

In the third category, researchers with an interest in software design of web services advocate the use of simulation to assist in the analysis, design, and testing of SBS systems. In Tsai et al.,[5] a framework has been developed to support the design of web services. Specification of real services including workflows can be described using the Process Specification and Modeling Language (PSML). Given pre-built code for un-timed services with pre-defined composition

policies, workflow specifications can be automatically generated. The DDSOS environment,[24] which employs HLA/run-time infrastructure,[20] is used to test the dynamics of web services as simulated components. ISTF (Interface Simulation and Testing Framework)[25] is another related effort. It is targeted at simulating end-to-end distributed application scenarios and showing how individual components will interface with each other in production.

3. Background

The research problem we have addressed in our work involves modeling and simulation of SBS systems that undergo structural changes at run-time. In the following, we provide a brief description of the SOA and the SOAD framework followed by dynamic structure modeling, with an exemplar adaptive SBS system.

A. Service Oriented Architecture

SOA represents architectural design principles toward building enterprise level distributed computing systems based on a set of loosely coupled computational components called 'services' that interact to provide functional utilities to interested clients.[2] All the software resources in SOA are termed as services. Services are defined using standard languages (e.g., Web Service Description Language (WSDL)),[26] provide publishable interfaces, and interact with each other to collectively execute a common task. Each service is independent of the state and context of other services. Furthermore, the interaction and communication is done using protocol independent message schemes (e.g., Simple Object Access Protocol (SOAP)).[27]

Similar to the producer–consumer scenario, service executioner and service requester are logically distinguished as publisher and subscriber, respectively. Publisher is the service provider whereas subscriber is the service consumer. The subscriber discovers an available publisher with the help of the third service known as the broker. The broker contains the publisher information in its registry which represents the published service interfaces of the publishers. To initiate a service invocation, the subscriber initiates a communication with the broker to search for service availability and if found the broker returns the service information so that the subscriber can directly interact with the publisher(s). In essence, a broker is the fundamental component in establishing the dynamic relations and interactions between the publisher and the subscriber and thus helps in maintaining the loosely coupled property of the SOA. When a publisher provides a service, it registers the service with the broker using a service specification (i.e. WSDL). The broker adds the new service specification to the Universal Description Discovery and Integration (i.e. UDDI),[28] an XML-based[29] registry of available services. Subscribers searching for a specific service request the broker to check for service availability and, if found, notify the subscriber of the service specifics using the WSDL specification of the publisher. The subscriber communicates with the broker and the publisher using messages (e.g., SOAP). Once the publisher gets the subscriber request, the service is executed and the resultant data service messages are provided to the subscriber.

The concept of service re-usability by composition is important in SOA. Adding new functionality from existing services is addressed by orchestration that specifies how the services would interact, including the execution order so that new functionality can be composed from existing services including the way services interact at the message level as well as the execution order of the interactions.[2] The ordering of the services is defined based on the message flow direction. The orchestration is always controlled from a single execution authority and can be categorized into two types – static and dynamic. Static orchestration is based on prior knowledge of the functionalities of existing services and is pre-configured. The dynamic orchestration requires run-time service composition by matching. Research on workflow systems has demonstrated its promise in dealing with service orchestration.

B. SOAD Modeling Framework

The basic building blocks for modeling SBS systems are the publisher, subscriber, broker, and their interactions. In the SOAD framework,[4] the specifications of the conceptual SOA descriptions are mapped to DEVS atomic and coupled models. In this approach, SOA compliancy for DEVS modeling is defined by extending the DEVS syntax and semantics with the SOA concepts and principles. SOA-compliant interactions among services are supported in accordance with WSDL message abstractions and SOAP specification using DEVS models that communicate with one another through input/output (I/O) ports and couplings.

The SOA framework has a higher level of abstraction as compared with the DEVS framework. The basic SOA model elements are divided into two groups. First, services, service description, and messages represent the *static* part of the SOA. Second, the communication agreement, messaging framework, and service registry and discovery represent the *dynamic* part of the SOA. As shown in Table 1, a set of DEVS elements have been developed that represent the static and dynamic aspects of the SOA. The elements have a one-to-one

Table 1. DEVS and SOA elements.

SOA Model Elements	SOA–DEVS Model Elements
Services (publisher, subscriber, broker)	Atomic models (publisher, subscriber, broker)
Service description entity (service-information)	Messages entity (service-lookup and service-message)
Messaging framework ports and couplings	Service registry and discovery executive model
Service composition	Coupled models (primitive and composite)

correspondence with the SOA services. A detailed mapping between DEVS and SOA can be found in Sarjoughian et al.[4]

The SOAD publisher, subscriber, and broker service specifications represent the basic structural and behavioral aspect of services. For each service, its I/O ports are defined in relation to those of other services. Similarly, the behavior of individual services as well as their flat and hierarchial compositions are defined such that all messages can be handled according to the DEVS and SOA specifications. The level of abstraction of the SOAP and WSDL specification accounts for the basic SOA interactions. Thus, SOA-compliant DEVS specification is consistent with SOA - i.e. service publication and subscription via broker and direct bindings between subscribers and publishers. The publisher, subscriber, and broker services are the basic elements for service-oriented software systems. The publisher publishes the service information to the broker. When a subscriber requests service information (i.e. service endpoint), the broker initiates a lookup in the service repository and returns the publisher information to the subscriber. If multiple publishers exist based on the user request, the first publisher found in the lookup is returned. Based on the returned publisher information, the subscriber initiates a service request to the publisher and the returned service data is consumed at the subscriber. Hierarchical composition of services is also supported. Flat and hierarchical service compositions have been developed and they are supported in the DEVS-Suite simulator.[8,30]

C. DSDEVS

Component-based modeling of dynamic structure systems has been well studied.[9–12] In Hu et al.,[10] the DSDEVS modeling approach has been developed to support structural changes of parallel DEVS models.[7] In Uhrmacher,[11] a variable structure modeling approach has also been developed using artificial intelligence concepts to support structural changes of DEVS models at run-time. The capability to model and simulate dynamic structure models according to DSDEVS was added to the DEVS-Suite simulator.[8] The simulator is developed based on the Dynamic Structure Discrete Event Network (DSDEN) system[9] specification.

The DSDEN is defined as a tuple $\langle \chi, M_\chi \rangle$, where χ is the name of an executive and M_χ is the model of the executive χ. The executive model is defined as a variant of an atomic DEVS model which has an element representing the network structure and a function that defines rules for adding and deleting DEVS model components and their couplings dynamically (i.e. during simulation execution). The simulator uses a single executive model for changing the structure of any modular, hierarchical parallel DEVS models. An executive model which conforms to the DSDEN specification is implemented in DEVSJAVA.[10] While the executive model has the knowledge of a network model structure at any time instance, it is not coupled to the network model or any of its components. The dynamic structure modeling and its implementation in DEVS-Suite is well suited for enabling dynamic structure modeling in SOAD.

D. ASBS System

Design and configuration of a SBS system demands making tradeoffs among multiple QoS features. Satisfying multiple QoS features such as timeliness, throughput, and accuracy requires the capability not only to model the logical specifications of the services, but also being able to assess their dynamic behaviors. This is because services often operate in environments where the services may become temporarily unavailable due to various system and network failures, overloads or other causes. Hence, development of SBS systems demands the capability to monitor the changing system status, analyze and control tradeoffs among multiple QoS features and adapt their service configurations accordingly. Such systems are referred to as ASBS.[6] The conceptual view of ASBS design is depicted in Figure 1, in which functional services used to compose the ASBS and the QoS monitoring and adaptation systems form a closed loop. The QoS monitoring system collects the measurements of desirable QoS features. Decisions provided by the QoS adaptation system are made to adjust the configurations and service operations of ASBS. Use of simulated services in

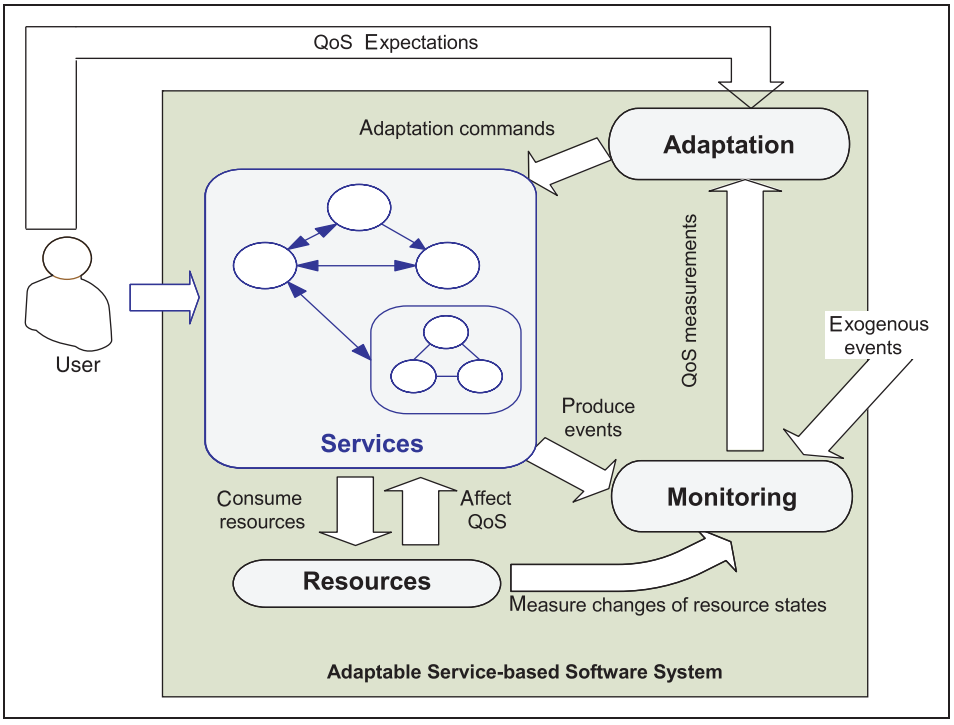

Figure 1. A conceptual view of ASBS design.

place of real services allows simulation-based design and provides support for verification and validation.

4. Approach

To simulate ASBS systems, it is important to support run-time addition or deletion of the publishers and subscribers as well as their connections to one another and the broker. The approach we have taken is to introduce DSDEVS modeling Barros[9] into the SOAD model.[4] The dynamic structure capability is appropriate for allowing SOA-compliant DEVS models to change their structures during simulation. The resulting DSOAD has a broker–executive model with a basic set of rules for supporting subscribers and publishers to be dynamically added or removed with proper interactions (couplings) supported between publishers and subscribers. Next, the broker–executive model and its design in DEVS-Suite are developed.

A. DSOAD

In compliance with the SOA specification, the relation between subscribers and publishers in SOAD is established through the broker. Here, the term 'relation' refers to the 'communication' between (and among) services. The SOAD model assumes pre-defined relations among broker and publishers (and subscribers). Hence, the SOA compliance structure of the system is defined by the modeler and appropriate abstractions are provided to relieve the modeler from creating service models at a low level of detail which complicates model validation. However, dynamic service creation and structural changes require one to adhere to SOA basic principles at run-time such that the general composition of the services can be assured.

With this goal in mind, we have developed DSOAD as an extension to SOAD with the addition of DSDEVS, which at its core has an executive model component with rules for adding (and removing) services and specifying how they are interconnected. The executive holds template structures along with rules for SOA-compliant structure changes. In accordance with DSDEVS, any service model developed in DSDEVS contains the executive model that enforces SOA compliancy. The semantics of the SOA include rules to relate the services. For example, SOA allows a relation between the broker and subscribers. So if a subscriber is dynamically instantiated in DSOAD the executive needs to ensure that it has couplings to the broker. The structural properties identified to make a configuration SOA compliant are defined below.

1. The subscriber(s) to publishers(s) communication must be discovered through the broker.
2. The publisher(s) can directly communicate to the broker.

3. The subscriber(s) can directly communicate to the broker.
4. The hierarchical composition of the publishers and subscribers cannot violate any of the above properties.

In addition, the direction of message flow among the broker, publisher and subscriber is also important for SOA compliancy. Although the concept of direction of message flow is not different from the concept already in SOAD (represented as incoming messages through input ports and outgoing messages through output ports), there is a need to account for this in DSOAD. SOAD component interactions through messages are pre-defined by the modeler and the modeler takes the direction of message flow into account. However, couplings are configured at run-time in DSOAD. Hence, the executive must contain the knowledge of the direction of the message flow (and thus input versus output ports) so that SOA-compliant structural rules are applicable for coupled models.

The executive model aids in enforcing SOA structural compliancy under dynamical settings. It facilitates the establishment of relations among publishers, subscribers, and the broker. The DSDEVS executive model by itself does not account for SOA and in particular does not account for the concept of the broker. One possible approach to accommodate the executive model is to associate it with the SOAD broker model. In this context, it is important to develop the association between the DSDEVS executive model and the SOAD broker. The complementary relation of the broker and the executive is as follows.

1. The broker mediates the publisher and subscriber relation. Similarly the executive model can facilitate dynamic flat and hierarchical structural component relations.
2. The broker implicitly enforces the direction of message flow by responding (or not responding) to incoming messages. The executive model can connect I/O ports of services to ensure the correct message flow direction.

However, the following dissimilarities exist between the broker and executive.

1. The executive can only support composite structures or can generate structures using predefined rules. The broker concept is aimed at supporting any publisher–subscriber relation.
2. The executive model, unlike the broker, does not conceptually distinguish between subscriber and publisher.
3. The executive model has complete knowledge of the service components and the structure of their composition. The broker may track publishers and subscribers that have communicated with it.
4. The executive model can support service composition and execution. The broker does not support composition and execution.

Considering the association between the executive and the broker models, we account for both the broker and the executive such that together they form the *broker–executive* model. Since the functionality of the executive is to enforce constraints on the structural composition of the services, the broker–executive accounts for the following rules.

A. When a publisher is added to the system:

1. connect the publisher output port `publish service` to the broker input port `publish service`.

B. When a subscriber is added to the system:

1. connect the subscriber output port `identify publisher` to the broker input port `identify publisher`;
2. connect the broker output port `found publisher` to the subscriber input port `found publisher`;
3. connect the subscriber output port `request service` to the publisher input port `request service`;
4. connect the publisher output port `publish service` to the subscriber input port `publish service`.

The rules A.1 and B.1 and B.2 establish the relations between the broker–executive and the subscriber and the publisher. Similarly, the rules B.3 and B.4 establish the relation between subscriber and publisher. In SOA, the subscriber–publisher relation is discoverable with the help of the broker and initiated by the subscriber. Interestingly, we need to account for the cases where publisher and subscribers can be dynamically added (and removed). In such cases, we need to resolve how and when the models and couplings should be reconfigured. Considering SOA's 'loosely coupled' property, delayed coupling is appropriate (i.e. create coupling prior to communication not at the time of instantiation). For simplicity, we include rules B.3 and B.4 as part of the broker–executive model. The rules for removal of the publisher and subscriber can be easily derived based on the rules that are defined for addition of services.

B. Broker–Executive Model Design

The broker–executive model in DSOAD is represented as a SOAD broker and a DSDEVS executive.

The model supports adding, removing, and coupling publishers, subscribers, and the broker that is subject to satisfying the SOA structural compliancy rules described in Section IV.A. As defined, the broker–executive model is a single logical component of DSOAD with its realization consisting of the DSDEVS executive and the SOAD broker.

The DSOAD uses the simple hardware model that was defined for the SOAD. The hardware model is defined as a component that orders messages it receives and routes using first in/first out (FIFO) discipline.[4] The publishers and subscribers exchange messages through this hardware component. The component supports dynamically adding (and removing) ports such that messages are properly routed to the intended recipient. Since dynamic addition and removal of publishers and subscribers is supported in DSOAD, the hardware component can accommodate I/O port addition and removal as needed. The broker–executive has the rules that couples publishers and subscribers to the hardware component and also to the SOAD broker according to the SOA rules. The important functionality of the broker–executive is twofold. First, it facilitates the change of structure of the system while maintaining SOA compliancy. Second, it has all the functionalities of the SOAD broker. The broker–executive adds couplings considering the direction of the message flow associated with the type of the DSOAD component being added. The message flow is directed from one component output port towards the input port of another component. So, the broker–executive takes into account the DSOAD component I/O port to ensure that couplings maintain compliancy. For example, if a subscriber is added to the system, the subscriber output port that sends a message to the SOAD broker to discover a publisher is coupled to the SOAD broker input port that handles the messages. All the couplings needed to connect the subscriber to the SOAD broker and router are created. Similarly, a publisher addition is also handled by the broker–executive. Multiple publishers and subscribers can also be added in DSOAD. If addition (and removal) are needed at the same time instance, then services can be added to (and removed from) the system without difficulty. The broker–executive handles such cases by adding (or removing) services one at a time with zero time advance (i.e. $dt = 0$) using an order defined by the parallel DEVS confluent function.[7] In the broker–executive model, the executive model has no DEVS-based I/O connection with the broker. However, the two models are related. The executive uses the broker model to create (or delete) structural connections with the model being added (or removed). The executive can add (and remove) SOA component models maintaining the SOA-compliant structural relations. However, once the structural compliancy is ensured the logical relations among the publisher and subscriber are maintained by the broker. This way SOA compliancy is ensured. This is a subtle yet important relation between the broker and executive – neither the executive nor the broker alone suffices in maintaining the SOA compliancy. This is important to note as their interaction is not apparent from DSOAD (no port connection or message interchange). The sequence diagram shows some of the main interactions among objects. The simulation engine in Figure 3 refers to the DEVS-Suite simulator that executes models, facilitates DEVS message exchanges among models and schedules events.[8] The underlying relation is captured in the UML model specifications as given in Figures 2 and 3. The sequence diagram shows some of the main interactions among objects. Details including object instantiations (e.g., creation of the service Publisher) are excluded for clarity. Whenever a publisher or subscriber is added (or removed) the executive must use the broker to add (or remove) couplings. The pseudo-code for the executive shows the generality of the model. The modeler can specify time instances to add and remove SOA component models (i.e. t_{add}, t_{remove}). The addSubscriber and addPublisher methods provide the generic capability to ensure SOA compliancy in coupling the components. As shown in Figure 4, $t, t_i \in \Re^{+}_{[0,\infty]}$ denote the simulation clock and initialization time, respectively. The time instances for adding or removing services can be arbitrarily scheduled (i.e. $t_{add}, t_{remove} \in t$ and the simulation clock t is updated by $dt \in \Re^{+}_{[0,\infty]}$). In addition, the states as presented in Figures 4(a) and (b) are symbolic representations of the complete states of the model. For example, the state denoted 'Waiting' in Figure 4(a) is different from the notation used to denote the phase of the model 'waiting'.

C. Flat and Hierarchical Model Compositions

Flat and hierarchical compositions correspond to the visibility of the service composition with respect to the other interested services. By visibility of a service, we mean whether the published service interface is available (or not) to the interested services. If the interface is available then it is public otherwise it is private. In flat composition, the services are visible to each other. In contrast, the visibility is set at levels of hierarchy in the hierarchical composition. The hierarchical composition can consist of publishers and subscribers or combinations of the two. In SOA, both flat and hierarchical compositions are possible and the hierarchical composition is supported at service levels. In DSOAD, the flat and hierarchical logical structure of services in SOAD is supported dynamically.

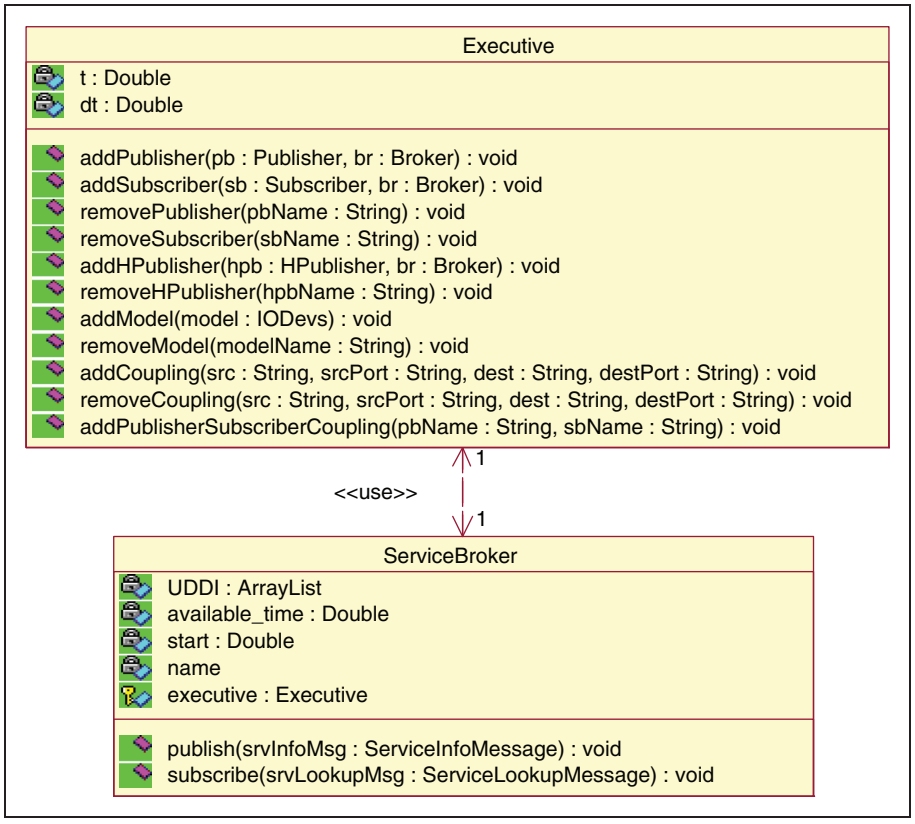

Figure 2. Unified Modeling Language (UML) class diagram for the broker–executive model.

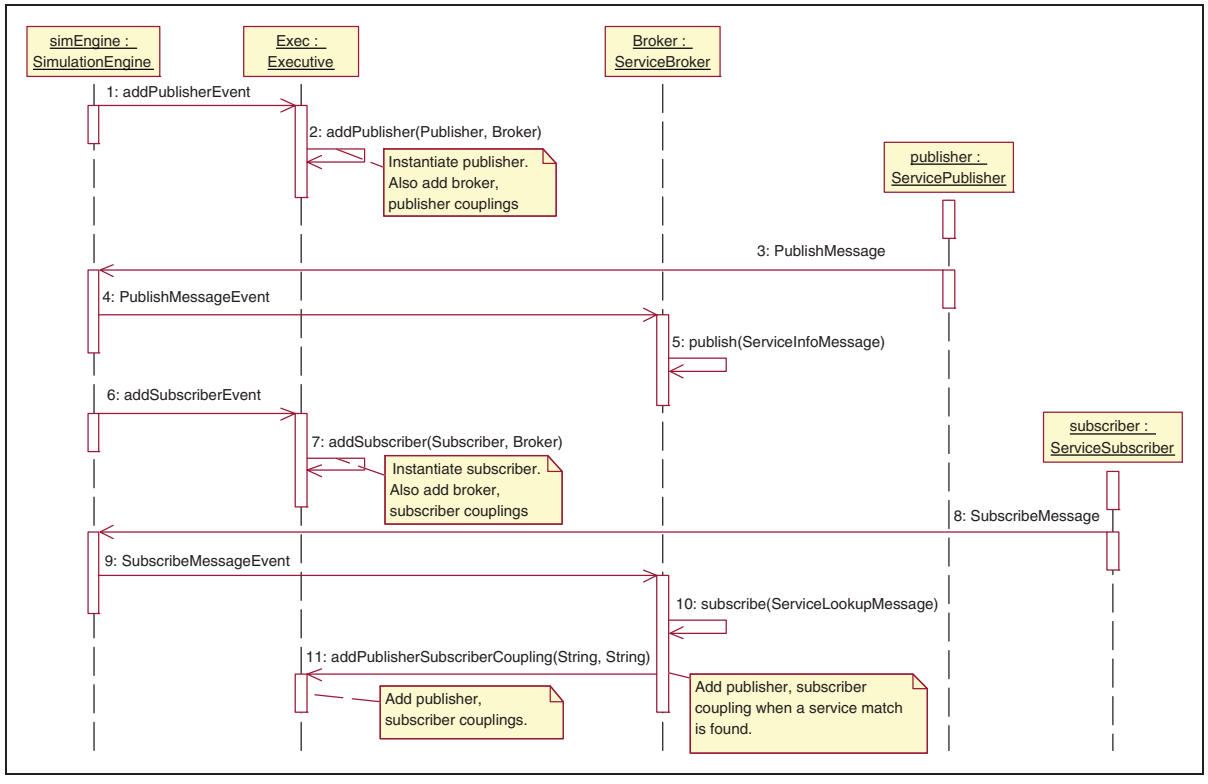

Figure 3. Sequence diagram for the broker–executive model.

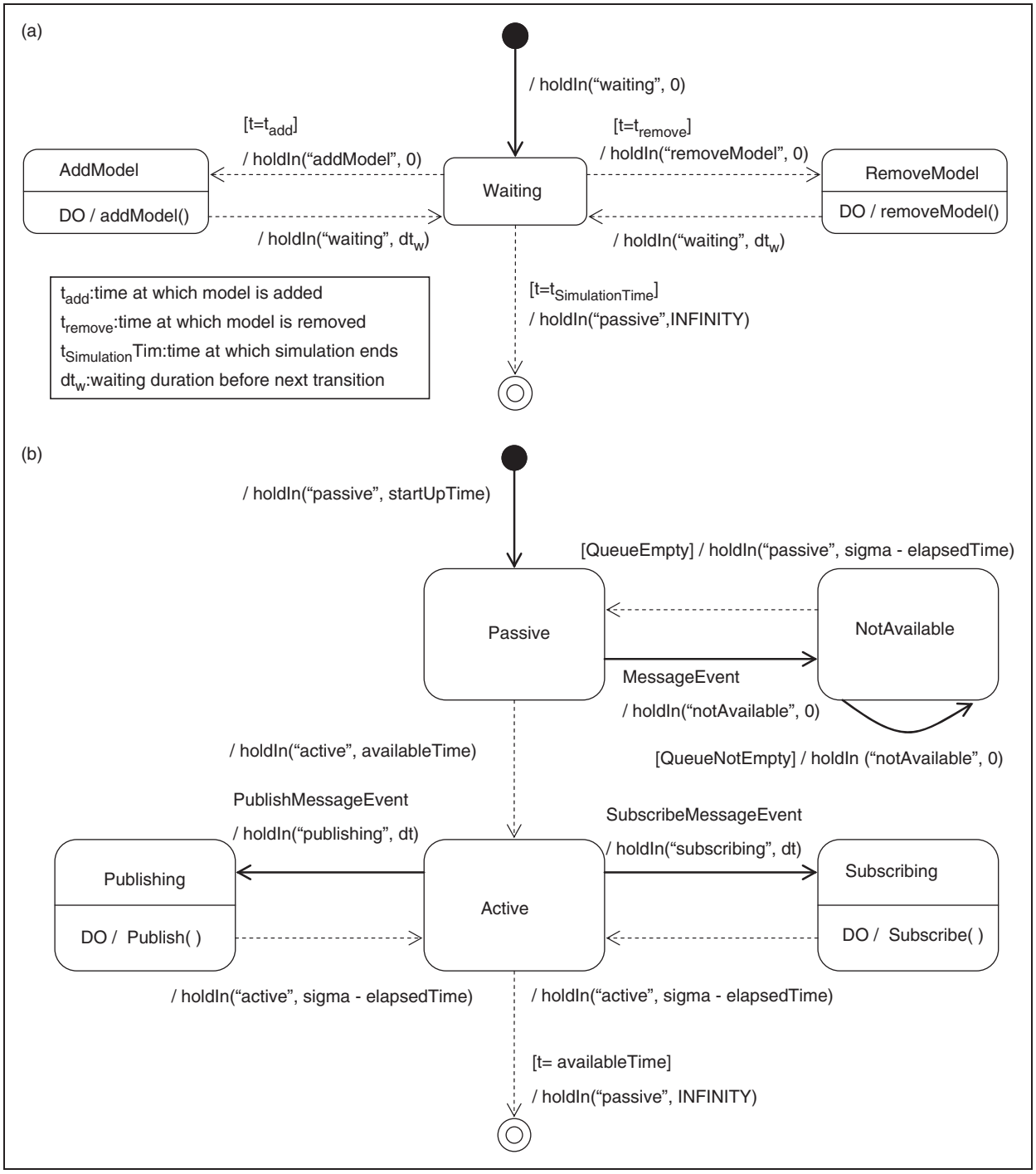

Figure 4. Statecharts for (a) the executive and (b) the broker models.

Services may be published publicly or privately. The pseudo-code for adding a publisher and subscriber is provided (see Listing 1).

The deletion of publishers and subscribers requires removing their couplings followed by the models. The implementation also handles the creation and deletion of router ports.

Based on the above formulation of DSOAD, the DEVS-Suite[8] simulator is extended with the DSDEVS API that is part of the DEVSJAVA simulator. This was attractive since the SOAD subscriber, publisher, broker, and router models as well as the DSDEVS executive model are implemented in DEVS-Suite. The extended DEVS-Suite simulator supports the

Listing 1

Listing for code template snippets for adding and removing publisher and subscriber

```
void addPublisher (Publisher pb, Broker br){
// add the publisher
addModel(pb);
// connect publisher to broker
addCoupling(pb.getName(), "publish", br.getName(), "publish");
}

void addSubscriber (Subscriber sb, Broker br){
// add the subscriber
addModel(sb);
// connect subscriber to broker
addCoupling(sb.getName(), "lookup", br.getName(), "subscribe");
addCoupling(br.getName(), sb.getName(), sb.getName(), "found");
// connect subscriber to router
addCoupling(sb.getName(), "request", router.getName(), "in");
}

void addHPublisher(HPublisher hpb){
// add the hierarchical publisher
addModel(hpb);
// add hierarchical publisher to broker
addCoupling(hpb.getName(), "publish", parent.broker.getName(), "publish");
}
```

basic capabilities for dynamically creating SOA-compliant DEVS simulation models.

5. Voice communication system example

To exemplify the DSOAD approach, a model of the voice communication system (VCS) is developed in DEVS-Suite. This is a simple example of an end-to-end system capable of streaming audio data from voice services to subscribers (i.e. clients).[6] The system can support various load scenarios given user requirements. The voice communication service publishes various quality audio data specified by sampling rates (e.g., 44.1 KHz) to which interested clients can subscribe to. The higher sampling rates produce higher quality audio data as it encodes more audio information per second. For example, a sampling rate of 220.5 KHz will produce superior quality audio data with respect to a 44.1 KHz sampling rate. The VCS under consideration supports two channel (i.e. stereo = 2 channel audio data versus mono = 1 channel audio data) audio data that can be sampled at 44.1, 88.2, 136.4, 176.4, or 220.5 KHz. The subscribers can request an audio data stream for a specified amount of time over the network and expect to receive the desired voice quality. The VCS can support multiple subscribers simultaneously such that each subscriber may request different qualities of audio data. The throughput provides a measure of the VCS performance. In general (i.e. under normal operating modes), the VCS throughput is proportional to the number of audio streams being delivered. Simulation models of this VCS system have been developed and then verified and validated with respect to its actual .Net implementation. In particular, a model of this system with the capability to change the number of publishers and subscribers has been developed. The resultant dynamic behavior in terms of voice publisher throughput has been compared to the actual implementation of the VCS. As a general scenario, we consider subscribers $S_1, S_2, \ldots, S_n$ to request service from any available publisher. Each publisher $P_1, P_2, \ldots, P_m$ can provide a service to one or more subscribers at some desired, specified sampling rates.

A. Throughput Calculation

The VCS sampling rate and number of audio channels are related to voice quality and they have an impact on

the VCS throughput. For example, let us assume the following: the given client $i \in \{1, \ldots, n\}$, S_i is the sampling rate (Hz), C_i is the number of audio channels, B is the bits per audio channel (bits/channel). Then the VCS throughput for the ith subscriber is $T_i = S_i \times C_i \times B$ (bits per second). To maintain the voice quality for any subscriber, the VCS generated data needs to be transported from the publisher (VCS) to the ith subscriber end (through the network). Hence the net VCS throughput for N subscriber would be $T = \sum_{i=1 \ldots N} T_i$ and for good audio quality the required network bandwidth (NBW) is $NBW \geqslant T$. Considering a scenario with one publisher (VCS) and one subscriber, let $S = 220.5$ KHz, $C = 2$, $B = 16$ bits. Then the publisher (VCS) throughput, $T = T_1 = 220.5 \times 2 \times 16 \times 10^3$ bps $= 7.056$ Mbps and the required network bandwidth NBW $\geqslant 7.056$ Mbps.

B. Flat Model Composition Example

To illustrate the core capabilities of the DSOAD framework, we consider systems consisting of flat composition. A router is used to represent the network used by services at a high level of abstraction. Also transducers are added that collect the information regarding the status of the publishers' throughputs. As the simulation progresses, additional publishers and subscribers are dynamically added to the system and later removed. Each subscriber can request voice data at a 44.1 or 220.5 KHz sampling rate for two-channel stereo data encoded in 16 bits. The voice publisher throughput is traced at approximately 0.93 simulation second intervals. The simulation is run for 60.0 logical seconds. As DSOAD has a continuous time base, the choice of sampling interval and simulation run period is chosen based on the actual VCS experiments. The implementation of the broker–executive (see Listing 1) ensures consistent model structures (e.g., removal of publishers and subscribers and their couplings). It is important to note that DSOAD does not limit the basic SOA system compositions possible in SOAD.[4] However, the approach only addresses the systematic modeling of dynamic SOA system compositions and as such solutions concerning dynamic service composition (i.e. workflow composition) are not addressed in this paper. Therefore, for illustration, we consider the following dynamic scenarios:

- a single publisher, multiple subscriber system;
- a multiple publisher, multiple subscriber system;
- dynamic publisher discovery.

The above scenarios capture the basic SOA-based dynamic system compositions and the corresponding examples 1, 2, and 3 illustrate the modeling and simulation capability in DSOAD for such systems. In each example, an executive model is configured with the time and type of model to be added and deleted. For simplicity, the broker is always added once during the simulation start time in each scenario, although it can be removed and added at a later time. Similarly, a transducer (collects publisher(s) statistics) and a network router (routes messages) are also added at the start of the simulation. All the publishers considered in the examples are instances of the VCS. The network configuration in each example is a star topology. The simulation event instances of publisher and subscriber model addition and removal are provided in Table 2. The simulation duration and throughput calculation interval are given in Table 3.

B.1. Single publisher, multiple subscriber system (Example 1).

In this scenario, we consider a single publisher and multiple subscriber system. A publisher P_1 and subscriber S_1 is added at the beginning of the simulation ($t = 0.0$) by the executive and it initiates the couplings (code listing 1). Once the service is published, the service lookup request from S_1 (at $t = 1.0$) to the broker results in a service being found. The broker then uses the executive to add the coupling for P_1 and S_1 communication (code listing 2).

Other subscribers S_2, S_3, S_4 are added to the system at a later time. Figure 5 shows a snapshot at simulation time $t = 41$ s where subscribers S_2 and S_3 are removed from the system.

B.2 Multiple publisher, multiple subscriber system (Example 2).

Here, we consider a multiple publisher and multiple subscriber system where, at $t = 20$, P_2 is

Listing 2

Listing for code snippets for dynamically coupling publisher and subscriber

```
 //Broker searches UDDI repository
//if exists uses executive to initiate couplings
. . .
iExecutive.addPublisherSubscriberCoupling(publisher,subscriber);
. . .
```

Table 2. Event times of publisher and subscriber addition and removal.

Example	Simulation Time (s)	Component	Operation	Sampling Rate
1	11.00	S_2	Add	220.5
	11.00	S_3	Add	220.5
	11.00	S_4	Add	220.5
	41.00	S_2	Remove	–
	41.00	S_3	Remove	–
2	11.00	S_2	Add	220.5
	11.00	S_3	Add	220.5
	11.00	S_4	Add	220.5
	20.00	P_4	Add	–
	41.00	S_2	Remove	–
	41.00	S_2	Remove	–
3	1.00	S_1	Add	44.1
	11.00	P_1	Add	–
	30.00	S_3	Add	44.1
	30.00	S_4	Add	44.1
	41.00	S_2	Remove	–
	41.00	S_3	Remove	–

Table 3. Simulation duration and throughput interval for all examples in Table (II).

Simulation Duration (s)	Throughput Interval (s)
60.0	0.93 (appx)

added to the system in addition to P_1 and S_1 at $t=0$. In this case, S_1 and S_2 request data from P_2 (after being served by P_1 at 220.5 KHz) at a 44.1 KHz sampling rate and simultaneously get a service from P_1 (and P_2).

B.3 Dynamic publisher discovery (Example 3). For dynamic discovery demonstration, we consider another example where, at $t=1.0$, subscriber S_1 is added; however, no publisher exists in the system prior to adding P_1 at $t=11.0$. As a result, the lookup request to the broker from S_1 returns a "Not found" message prior to $t=11.0$. Once P_1 is added and it publishes to the broker, the broker creates the couplings for P_1–S_1 communication, returns a service information message on the lookup request and then S_1 initiates the communication with P_1. Listing 3 shows code snippets for publisher discovery.

To show the capability of capturing dynamic characteristics of the service-based system in the DSOAD framework, we traced the throughput of the real and simulated VCS with similar scenario and timing characteristics. However, syncing the timing of the real system with the simulated system is difficult. We can precisely control the simulation timing events. However, the real system timing showed drifting for different runs. As we want to demonstrate that the dynamic characteristics of SBS can be properly captured by DSOAD, precise time syncing is not required for our purpose. As a rule of thumb, we assume that the simulation and the real system are time synced if timestamps are the same when rounded to the nearest integer. The real versus simulation throughput of the voice

Listing 3

Listing for code snippets for dynamic discovery of publisher through broker

```
 //Broker
if(!serviceFound)  {
//Service(s) are not found
returnMsg = new ServiceInfoMessage("Not Found", null, null, null);
}
else  {
//Service(s) are found
iExecutive.addPublisherSubscriberCoupling(publisher,subscriber);
returnMsg = UDDI.get(index);
}

//Publisher
if(phaseIs("publishing"))   {
ServiceInfo = new ServiceInfoMessage(ServiceName,Endpoints);
m.add(makeContent("publish", ServiceInfo));
}
```

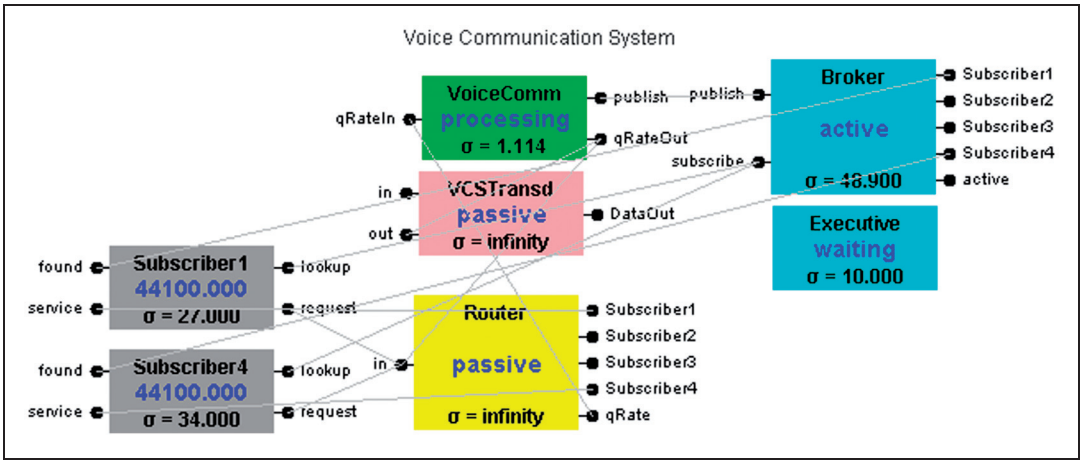

Figure 5. Component view of a flat structure VCS model with a subscriber added at $t = 41$ s (Example 1).

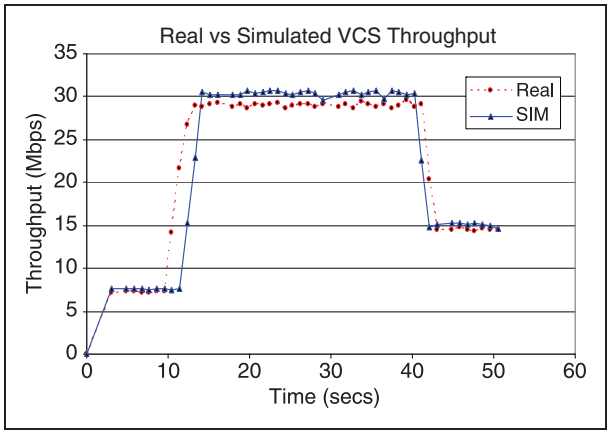

Figure 6. Throughputs for a flat structure VCS and its simulated model (Example 1).

publisher is shown in Figure 6. Each subscriber request increases the streaming data throughput at the publisher. Once the subscribers are removed from the system the publisher stops streaming data for that particular subscriber and a reduction in publisher throughput is noticed.

An interesting point to note in the graph is the gradual increase in VCS throughput ($t > 10$ s) for both the real system and the simulation. In the simulation, S_2, S_3, S_4 are instantiated at the same simulation clock ($t = 11$ s). The order of the addition of these subscribers has no significance. However, each subscriber's request goes through the router (FIFO) incurring delays, which results in requests being processed in a sequential way at the publisher. Therefore, the data streaming begins at different time instances resulting in the gradual increase in VCS throughput. In the real system, the behavior is attributed to the fact that all threads are processed sequentially; hence, even if we instantiate all the threads at the same time (approximately, as such precision control is difficult in the real VCS system) the threads get processed sequentially by the operating system and the resultant data streaming begins at different time instances. The trend in the real system and the simulation carry the signature of the dynamic change in the system behavior. Simulating such a dynamic system and capturing the resultant behavior is the fundamental capability that DSOAD provides in addition to modeling and simulating SBS systems.

C. Hierarchical Model Composition Example

Here, we consider a Hierarchical VCS as depicted in Figure 7. It consists of one subscriber (i.e. S_1), one broker (i.e. B_1), one router (i.e. R_1), and three publishers (i.e. *HCVoiceCommService*, *Sampling*, and *SendVoice*). To demonstrate hierarchical models in DSOAD, the Sampling and the Sendvoice are composed together to form the HCVoiceCommService in a hierarchical structure. Broker B_1 serves as the mediator for the publisher and subscriber. The router R1 routes messages between the publisher and subscriber. For simplicity, we chose the hierarchy in this example not as a logical hierarchy from the SOA perspective. However, it is important to note that DSOAD does not prohibit or limit modeling such scenarios. Here, the .Net implementation of VCS is modified to show a real world example of hierarchical composition. The VCS service is composed of two services – SamplingService and SendVoiceService. The SamplingService samples voice data and sends it to the SendVoiceService that sends the data to the subscriber. In this particular case, the simulation and the real system hierarchical components have a one-to-one relation. The developed simulation models are also consistent with the real system hierarchical structure. Similar to the example in the flat model

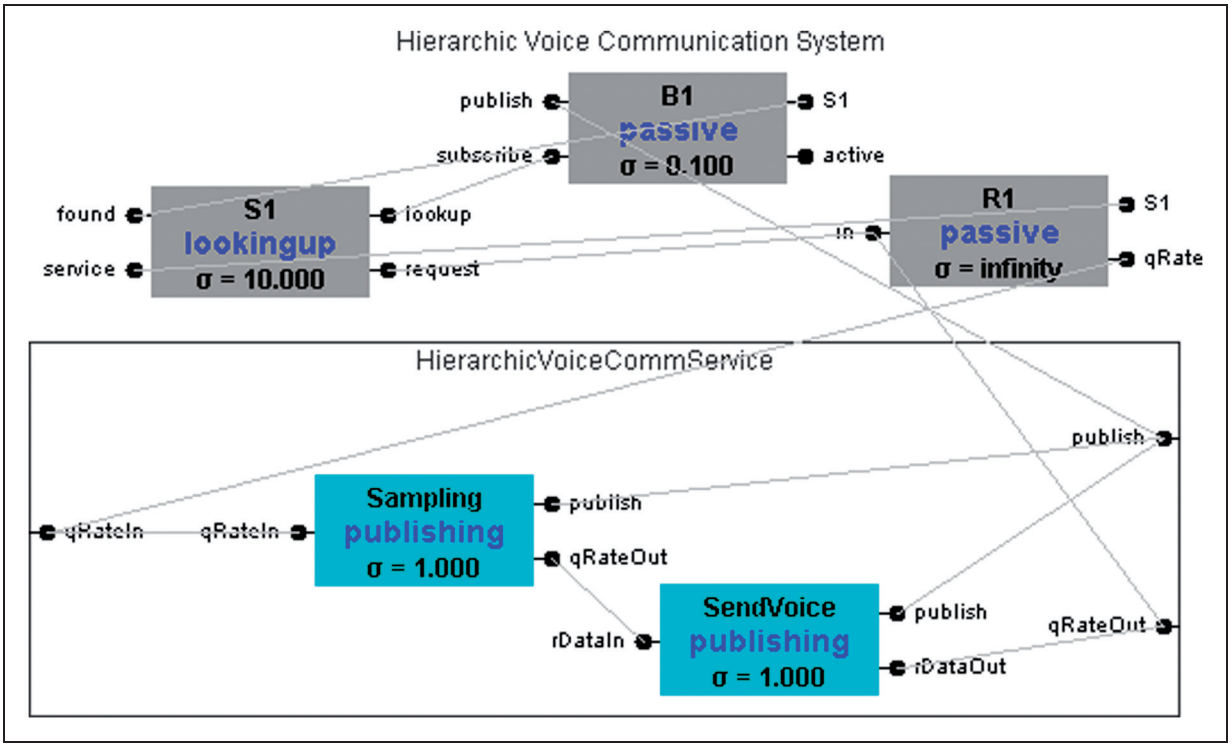

Figure 7. A component view of a hierarchical VCS model at $t = 0$ s.

composition, we added three subscribers (S_2, S_3, S_4) to the system at $t = 10.0$ s and then removed S_1, S_3 at $t = 40.0$ s. The resultant throughput for the real and the hierarchical VCS is shown in Figure 8.

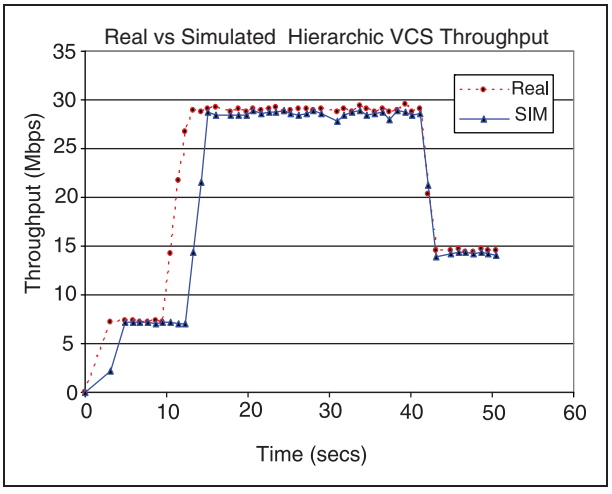

Figure 8. Throughputs for a hierarchical structure VCS and its simulated model.

6. Discussion

Modeling service behavior in a dynamic scenario in accordance with SOA has been emphasized in DSOAD. Based on the idea of first principles, the service models and their interactions are derived from specifications in the SOA. The service models support a detailed interaction semantics and provide timing parameters to account for time-dependent behavior in service interactions. This level of modeling is important in simulations to address QoS prediction and performance analysis of SBS systems.[6] Simulations in DSOAD can provide information that are closely related to the service layer. However, as DSOAD focuses on modeling abstractions for the software, it does not account for detailed hardware dynamics. It accounts for the hardware at a high level of abstraction, i.e. a network router. In real systems, services may exhibit behaviors that are strongly affected by hardware constraints. In addition, data collected on real systems represent an aggregate level view of the system resources. For example, Windows Performance Object[6] used in developing the above examples is closely related to the system resource and hardware and an Activity-State-QOS (ASQ)[6] model was developed to relate the service level behavior with the traced data. To bridge the gap between modeling the service behavior and the hardware constraints, a combined service model and hardware model along with service and hardware interaction called SOC–DEVS[31] is under development. This kind of modeling is more attractive for verification and validation with real SOA. Thus, DSOAD can be used for simulation of adaptive SBS

systems where the logical dynamical structures at the service levels are of interest.

7. Conclusion

The main contribution of this paper is the DSOAD approach and its support in the DEVS-Suite simulator. The SOAD framework is extended to support simulation of dynamic structure SOA-compliant DEVS models. Hence, with this approach, dynamic service-based systems can be modeled using the concept and principles of DEVS and SOA and simulated in the DEVS simulator. In particular, we defined the broker–executive model and implemented it such that modelers can develop simulations that conform to dynamically changing SOA structure compositions. We developed model templates for constructing hierarchical, dynamic SOA-complaint DEVS modeling. We exemplified the role of DSOAD in simulating dynamic SBS systems using the DEVS-Suite simulator. Such simulation models can support the design of ASBS systems.

There is interesting future research to be undertaken. An ongoing effort is focusing on supporting mixed real and simulated services. This kind of capability is key for having a testbed that can represent a large number of interchangeable real and simulated services. The capability to support a mix of real and simulated services is important for verification and validation of ASBS designs. Another research direction is to support distributed simulation using web services. For example, the DEVS/SOA environment[16] can be used with the DEVS-Suite simulator. The results of these research efforts can lead to more efficient simulations enabling design and development of complex SBS systems.

Acknowledgments

Part of this research was carried out while the second author was with the Department of Electrical and Computer Engineering, University of Tehran, Iran.

Funding

This research was supported by NSF (grant number CCF-0725340).

References

1. Chen Y and Tsai WT. *Distributed service-oriented software development*. Kendall/Hunt Publishing, 2008.
2. Erl T. *Service-oriented architecture concepts. Technology and design*. Prentice Hall, 2006.
3. Bause F, Buchholz P, Kriege J and Vastag S. A framework for simulation models of service-oriented architectures. In: Kounev S, Gorton I and Sachs K (eds) *SIPEW (Lecture Notes in Computer Science, Vol. 5119)*. Springer, 2008, pp. 208–227.
4. Sarjoughian HS, Kim S, Ramaswamy M and Yau SS. A simulation framework for service-oriented computing systems. In: *Proceedings of the Winter Simulation Conference*. Miami, FL, USA, 2008, pp. 845–853.
5. Tsai WT, Cao Z, Wei X, Paul R, Huang Q and Sun X. Modeling and simulation in service-oriented software development. *Simul Trans* 2007; 83: 7–32.
6. Yau SS, Ye N, Sarjoughian HS, Huang D, Roontiva A, Baydogan M and Muqsith MA. Towards development of adaptive service-based software systems. *IEEE Trans Services Comput* 2009; 2: 247–260.
7. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems*, 2nd ed, Academic Press, 2000.
8. Sarjoughian HS. DEVS-Suite http://devs-suitesim.sourceforge.net (2009, accessed January, 2010).
9. Barros FJ. Modeling formalisms for dynamic structure systems. *ACM Trans Model Comput Simul* 1997; 7: 501–515.
10. Hu X, Zeigler BP and Mittal S. Variable structure in DEVS component-based modeling and simulation. *Simul Trans* 2005; 81: 91–102.
11. Uhrmacher AM. Dynamic structures in modeling and simulation: A reflective approach. *ACM Trans Model Comput Simul* 2001; 11: 206–232.
12. Zeigler BP. Toward a simulation methodology for variable structure modeling. In: Elzas MS, Zeigler BP and Oren TI (eds) *Modeling and simulation of methodology in the artificial intelligence era*. Amsterdam: North Holland, 1986, pp. 185–210.
13. Narayanan S and Mcilraith S. Analysis and simulation of web services. *Comput Networks* 2003; 42: 675–693.
14. DARPA Markup Language for Services http://www.daml.org/services (accessed January 2009).
15. Narayanan S. Reasoning about actions in narrative understanding. In: *Proceedings of International Joint Conference on Artificial Intelligence*. 1999, pp. 350–358, Morgan Kaufmann, San Francisco, CA.
16. Mittal S, Risco-Martín JL and Zeigler BP. DEVS/SOA: A Cross-platform framework for net-centric modeling and simulation in DEVS unified process. *Simul Trans* 2009; 85: 419–450.
17. Zeigler BP, Hall SB and Sarjoughian HS. Exploiting HLA and DEVS to promote interoperability and reuse in Lockheed's corporate environment. *Simulation Journal* 1999; 73: 288–295.
18. DEVSJAVA, 'Arizona Center for Integrative Modeling and Simulation' http://www.acims.arizona.edu/SOFTWARE (2001, accessed January 2009).
19. Li BH, Chai X, Di Y, Yu H, Du Z and Peng X. Research on service oriented simulation grid. In: *Proceedings of the 8th International Symposium on Autonomous Decentralized Systems* 2005; 7–14, Chengdu, China.
20. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules, Version 1516-2000, IEEE, 2000.

21. IBM Product Lifecycle Management http://www-03.ibm.com/solutions/plm/index.jsp (accessed December 2008).
22. Open Grid Services Architecture http://www.globus.org/ogsa (accessed December 2008).
23. Extensible Modeling and Simulation Framework http://sourceforge.net/projects/xmsf/ (2003, accessed December 2008)
24. Tsai WT, Fan C and Chen Y. DDSOS: A dynamic distributed service-oriented simulation framework. In: *Proceedings of The 39th Annual Simulation Symposium* 2006; 160–167, Huntsville, AL.
25. IONA Interface Simulation Testing Framework http://www.iona.com/solutions/it_solutions/istf.htm (accessed January 2009).
26. Web Services Description Language http://www.w3.org/TR/wsdl (accessed January 2010).
27. Simple Object Access Protocol http://www.w3.org/TR/soap/ (accessed January 2010)
28. Universal Description Discovery and Integration http://www.uddi.org/pubs/uddi_v3.htm (accessed January 2009).
29. Extensible Markup Language http://www.w3.org/XML/ (accessed January 2010)
30. Kim S, Sarjoughian HS and Elamvazuthi V. DEVS-Suite: A simulator for visual experimentation and behavior monitoring. In: *High Performance Computing & Simulation Symposium, Proceedings of the Spring Simulation Conference*. San Diego, CA, March 2009, pp. 1–7.
31. Muqsith MA and Sarjoughian HS. A simulator for service-based software system co-design. In: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, SIMUTools 2010*. Torremolinos, Malaga, Spain, 2010, pp. 1–9.

APPENDICES

I. Appendix: DSOAD simulation environment

The examples just described are developed in DEVS-Suite, an open source simulator with an API that supports DSOAD.[31] The simulator is efficient[30] – the wall clock time periods required for executing the above simulations are 0.8 s and 0.9 s for the examples shown in Figures 9 and 7, respectively. We also ran

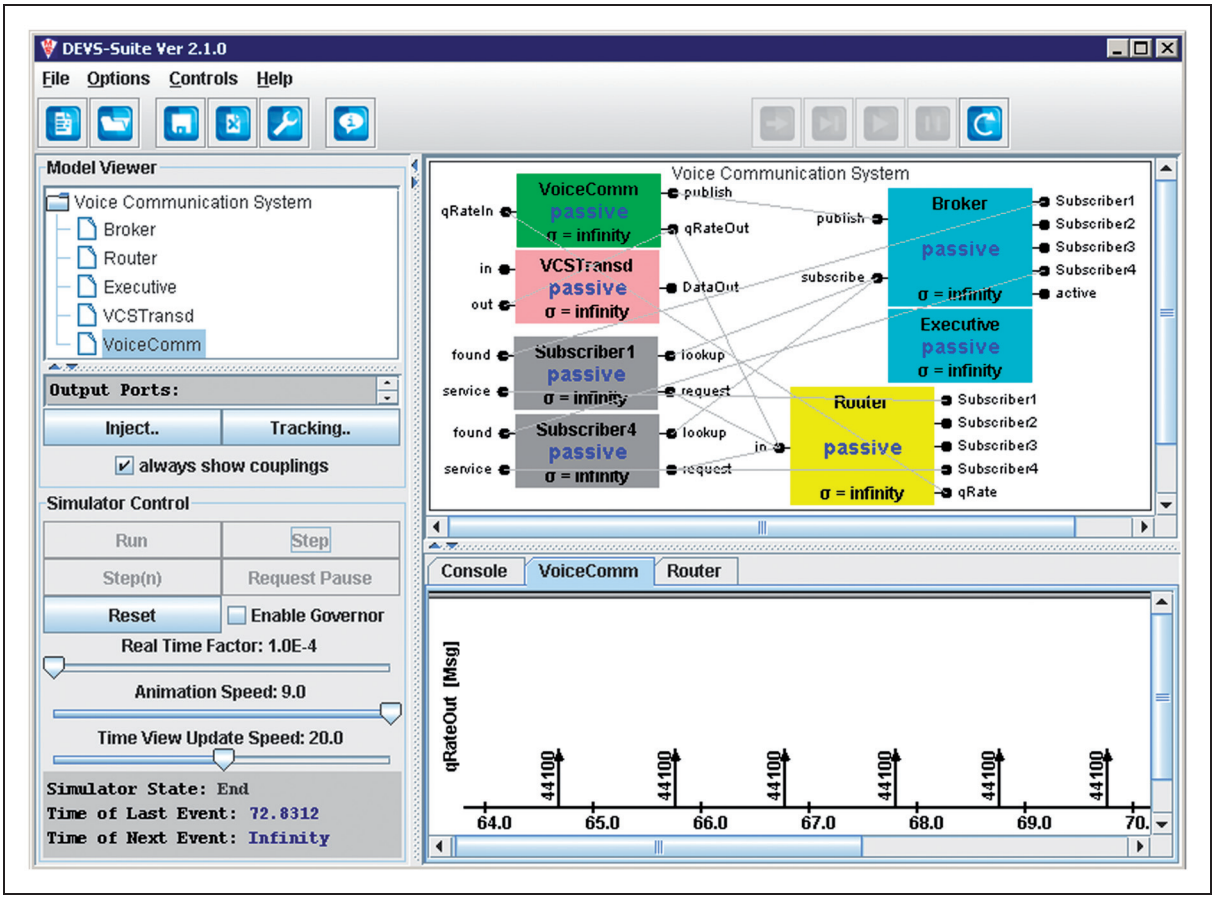

Figure 9. DEVS-Suite simulator UI (User Interface) depicting a flat structure VCS model.

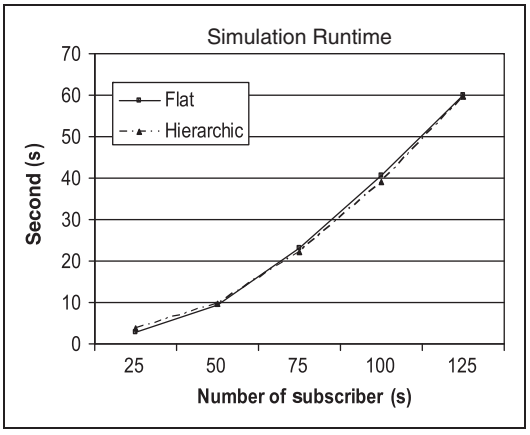

Figure 10. Simulation run-time efficiency (refer to Figures 5 and 7).

simulations with more than 100 subscribers for both the flat and hierarchical scenarios to compare the wall clock time periods for executing the simulations (see Figure 10). We ran the simulation 10 times for each data point and took the average. Under ideal conditions we would expect the hierarchical model to be less efficient. In general, the simulator's performance degrades as the hierarchy of a model increases due to additional I/O messaging. However, for a model with hierarchy of depth two, the performance degradation is negligible as compared with a flat model. The computer has an Intel Core 2 Duo CPU E8200 at 2.66 GHz and 3.23 GB of memory and an Intel 82566DM Gigabit network card set at 100 Mbps. The operating system is Microsoft Windows XP Professional with Service Pack 3, .NET version 3.5 and JDK version 1.5. For a comparative view, we also run the same scenario in the real VCS system with similar timing events and collected the voice throughput samples. The real VCS experiment was also executed using the same machine configuration. It should be noted that the simulator's User Interface (UI) supports dynamic addition and removal of block components (see Table 2), but not the creation of I/O and state trajectories. Plotting trajectories for components that are added during simulation execution remains as future work.

Hessam S. Sarjoughian received a Ph.D. degree in electrical and computer engineering from the University of Arizona, Tucson, Arizona, USA in 1994. He is an Associate Professor of Computer Science and Engineering at Arizona State University, Tempe, Arizona, USA. He is a co-director of the Arizona Center for Integrative Modeling & Simulation (ACIMS). His research focuses on modeling and simulation theory, methodology, model composability, co-design modeling, visual simulation modeling, and agent-based simulation. He lead the establishment of the online M&S Masters of Engineering Degree Program at Arizona State University in 2004.

Mohammed A. Muqsith is currently a Ph.D. candidate in computer science at Arizona State University, Tempe, Arizona, USA. He is a member of the Arizona Center for Integrative Modeling & Simulation at Arizona State University. His research interest includes simulation-based design, service-oriented software/hardware co-design, system performance modeling, software engineering and networked embedded systems.

Dazhi Huang is a Ph.D. student in computer science at Arizona State University, Tempe, Arizona, USA. He received the BS in computer science from Tsinghua University in China. His research interests include middleware, mobile and ubiquitous computing, and workflow systems in service-oriented computing environments.

Stephen S. Yau is a fellow of the IEEE and the American Association for the Advancement of Science. He received a Ph.D. degree in electrical engineering from the University of Illinois, Urbana. He is currently the director of the Information Assurance Center and a Professor of Computer Science and Engineering at Arizona State University, Tempe. He was previously with the University of Florida, Gainesville, and Northwestern University, Evanston, Illinois. He served as the president of the IEEE Computer Society and the editor-in-chief of *Computer*. His current research is in distributed and service-oriented computing, cloud computing, software engineering, ubiquitous computing, trust management, cyber situation awareness, and data privacy.