A Simulator for Service-Based Software System Co-design

Mohammed A. Muqsith
School of Computing, Informatics, and
Decision Systems Engineering
Arizona State University, Tempe, AZ 85281-8809
011-281-300-1261
mmuqsith@asu.edu

Hessam S. Sarjoughian
School of Computing, Informatics, and
Decision Systems Engineering
Arizona State University, Tempe, AZ 85281-8809
001-480-965-3983
sarjoughian@asu.edu

ABSTRACT
The adoption of the Service Oriented Architecture (SOA) as the foundation for developing a new generation of software systems poses important challenges in system design. While simulation tools serve a principal role in design, there is a growing recognition that simulation of Service-Based Software Systems (SBS) requires modeling capabilities beyond those that are developed for the traditional distributed software systems. In this paper, a novel simulator is developed based on the SOC-DEVS co-design approach and realized as an extension to the SOA-compliant DEVS simulator. The tool supports DEVS simulation modeling of not only the services consistent with Service-Oriented Computing (SOC) concepts and principles, but also the networked hardware components on which services must execute on. An example model for a voice communication system is developed to illustrate the kind of hardware and software components that can be modeled and simulated.

Categories and Subject Descriptors
I.6.2 [**Simulation and Modeling**]: Application; I.6.3 [**Simulation and Modeling**]: Model Development – modeling methodologies; I.6.5 [**Simulation and Modeling**]: Simulation Languages; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems – environments.

General Terms
Design, Verification.

Keywords
Co-design, DEVS/DOC, DEVS-Suite, Service-Based Software System (SBS), Service Oriented Architecture (SOA), SOA-DEVS (SOAD), SW/HW Simulation.

SIMUTools 2010 March 15–19, Torremolinos, Malaga, Spain.

1. INTRODUCTION
Service Oriented Architecture (SOA) [7] 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 subscribers. SOA has evolved to address the demand to develop & deploy large scale software systems that are cost effective to reuse, maintain and easily adaptable to infrastructure change. A key promise of 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. Design decisions spanning software, hardware, and their combination have significant roles in achieving the desired runtime QoS. To attain a level of tractability in developing such systems, the use of modeling and simulation tools can aid in exploring alternative designs,

In the remainder of this section, we will present our motivation for developing an SOA-compliant simulator which supports co-design concept. In Section 2, we describe background closely related to this paper. In Section 3, the basic approach for Service-Based Software System co-design is described. In Section 4, the model components for the SOC-DEVS simulator are detailed. In Section 5, an example model is developed and simulation results are presented and discussed. In Section 6, we present our conclusions and future work.

1.1 Motivation
In the design of Service-Based Software Systems (SBS) capable of satisfying multiple QoS attributes, simulation-based modeling is desirable as simulation can play a central role in enabling tradeoff study among time-based QoS attributes. To build Service-Based Software Systems with capability to support multiple QoS, simulation can play an important role in system architectural design verification and validation. The idea is to develop a model of the system under design using a simulator that supports SOA concepts and principles. Service models can capture the fundamental dynamics among SOA components and the system behavior can be observed under various configurations and the resultant impact on the QoS. Such a simulator enables analysis and design capabilities by aiding in design, implementation and testing of the Service-Based Software Systems. This is an important tool for architectural design validations that are impractical to support with actual service deployment.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8735
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8735

For SOA-compliancy, it is important to capture service functionality with direct representation of SOA artifacts like the service publisher, service subscriber, and service broker. In addition, the relationship among the publisher, subscriber and broker needs to be accounted in the simulator such that it allows the application model builders to concentrate on application specification rather than on the details for SOA compliancy. From system architecture, design and development perspective, system specification with separation of hardware and software is important. Ability to model software and hardware separately and their combined interaction have important practical use. For example, the users can select services and list their desired QoS under the presence of some uncontrollable, but predictable environmental fluctuations [18]. System resources such as available memory, CPU speed, network bandwidth etc. are important environmental factors. Such environmental factor fluctuations manifests through resource constraints of the execution environment of services as results of complex interactions among services. However, to provide the service level agreement on QoS and support resource management, a simulator supporting not only software, but also hardware aspect of the system is needed. Without this capability the system design is incomplete since resultant QoS of complex software-software and software-hardware interactions with the resource conflicts cannot be adequately accounted for. In essence, an appropriate level of abstraction for models with SOA compliancy along with a capability to separately model software and hardware with support for model synthesis is critical for understanding any SBS design.

Existing work on Service-Based Software System modeling and simulation emphasizes on process specification and workflow aspect of service. Business Process Execution Language (BPEL) [3], Process Specification and Modeling Language, (PSML-S) [16] consider process flow to represent service functionality that treats QoS primarily in terms of the runtime behavior of the software components, with no (or limited) consideration for underlying hardware. Select approaches and tools consider hardware [14, 2, 9, 12, 13]. For example, Bause et al. [2] use OMNeT++ [12] to simulate detailed network protocols. However, the service functionality can only be simulated as a process chain comparable to BPEL with no direct representation of SOA artifacts. In addition, even though network resources (e.g., CPU speed, system memory) are accounted [8] for using OMNeT++, complex service interactions that dynamically impact system resources and hence the impact on QoS cannot be independently observed. In [14], service models are developed using SOA concepts and principles. However, in terms of hardware, a simplified abstraction of a network router is used and the limitation of simplified abstractions of hardware is discussed and the importance of detailed hardware models is outlined. In OMNeT++ [12], OPNET [13], ns-2 [11] detail network protocol level simulation is supported. However, the software layer in these tools do not account for SOA compliancy. It is also important to note that none of the above tools and their underlying approaches apply the co-design concept (i.e., systematic sw/hw separation and synthesis) as part of modeling methodology. In DEVS/DOC [9], co-design concept is applied in a systematic way for separately capturing the software-hardware dynamics as well as their interactions to simulate object-based distributed software systems. The Distributed Co-operative Object (software) and Loosely Coupled Network (hardware) modeling layers in DEVS/DOC are suitable for validation of design architectures for Distributed Object Computing [4] systems. However, since DEVS/DOC simulator is intended for distributed objects co-design [9], its software layer does not account for SOA concepts. Thus, it lacks support for simulating SBS architectural designs. In another approach [1], hardware is abstracted as a parameterized Layered Queuing Network (LQN). However, such approaches lack the concept of co-design – i.e., the advantages of a systematic separation and synthesis of HW/SW is missing.

2. BACKGROUND

2.1 Service Oriented Computing

Service Oriented Computing (SOC) is the computing paradigm based on Service Oriented Architecture (SOA) [7]. It defines a set of loosely coupled computational components called "services" that interact to provide functional utilities to interested subscribers. All the software resources in SOA are termed as services. Each service is a well defined self contained software module providing functionality to interested subscribers.

In SOA, services are defined using standard language (e.g. WSDL), provide publishable interfaces, and interact with each other (as well as the subscribers) to collectively execute a common task. In addition, each service is independent of the state and context of other services – making services stateless. Furthermore, the interaction and communication is done using protocol (e.g., TCP/IP) independent message scheme (e.g., SOAP). Similar to the producer-consumer scenario, service executioner and service requester are logically distinguished as – Publisher & Subscriber, respectively. Publisher is the service provider whereas Subscriber is the service consumer. The subscriber discovers available publisher with the help of the third software entity known as the Service Broker. It 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 service information is returned so that the subscriber can directly interact with the publisher(s). In essence, a broker is the fundamental component in establishing the dynamic interaction/relation between the publisher and the subscriber and thus helps in maintaining the loosely-coupled property of SOA.

2.2 Hardware Software Co-design

The concept of HW/SW co-design refers to partitioning a system under design in terms of hardware and software parts such that each can be developed separately and thereafter synthesized with the other. Such efforts are aimed at enabling robust system designs with emphasis on improving hardware and software interaction. The advantage of HW/SW co-design is that it allows system architects and system engineers three degrees of freedom – i) separate specification of software, ii) separate specification of hardware, and iii) synthesis of software and hardware. While (i) and (ii) allow flexibility in independent software and hardware designs, (iii) provides an important capability to account for integrated system behavior under various software and hardware configurations. The HW/SW co-design concept [17] has been successfully applied in embedded system simulation, design and development.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8735
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8735

2.3 SOA-DEVS

SOA-compliant DEVS (SOAD) [14] refers to a modeling and simulation framework targeted for Service-Oriented Computing (SOC) systems. The elements of the SOC model specifications are based on the conceptual SOA descriptions that are mapped to DEVS atomic and coupled models [19]. The resulting SOA-DEVS services adhere to the combined semantics of DEVS and SOA principles. In particular, SOA-compliant message abstractions are designed in accordance to the WSDL and SOAP specification and exchanged through DEVS ports and couplings. To simulate SOC-compliant DEVS models, the DEVS-Suite simulator is extended to specify SOA abstractions [10, 6]. The simulator supports basic SOA elements including services, service registry, service discovery, and messages. The SOAD models communicate with messages that represent service description, look up, and service messages [14]. Services communicate with one another via messages that contain service description or other content consistent with a chosen messaging framework. For example, a message from the broker to the subscriber is a service description which contains an abstract definition (an interface for the operation names and their input and output messages) and a concrete definition (consisting of the binding to physical transport protocol, address or endpoint, and service). The fundamental architecture and high-level design of software-based systems can be simulated and validated before developing low-level design, implementation, and testing. It should be noted that a fundamental difference between DEVS and SOA is the 'broker' concept. SOA is grounded in the separation of publisher and subscriber services which can send and receive messages. The message-based interactions between the publisher and subscriber services can only be established by the broker service. However, while SOAD approach noted the importance of co-design, it did not support developing co-design simulation models.

2.4 DEVS/DOC

DEVS/DOC is a simulator for Distributed Object Computing (DOC) systems [9]. It supports simulation of distributed reusable objects distributed over multiple, heterogeneous, computing and networking elements and applications efficiently, flexibly, and robustly. A formal model of DOC systems is specified by Butler in [4] where he proposed an abstract mathematical framework for specifying a static, structural model of a generic distributed object computing environment. DEVS/DOC is a DEVS based realization of Butler's framework. It introduces the capability to specify the time-based dynamics of software and hardware components as well as the mapping of the former to the latter. DEVS/DOC enables modeling of hardware components responsible for executing software components. The software and hardware models are referred to as the Distributed Cooperative Object (DCO) and Loosely Coupled Network (LCN) layers, respectively. The framework defines the Object System Mapping (OSM) to provide for the mapping of software components to hardware components. A set of metrics is defined to extract key parameters of interest to enable studies of alternative architectural designs given various choices for DCO and LCN layers as well as their mappings. This framework takes a simple, yet powerful view by providing models to characterize dynamic behavior of a distributed object computing environment by representing two distinct layers of behavior – one for software objects and another for hardware objects (independently of one another) – and allows a mapping between them. The framework facilitates modeling abstract behavior of the software components independent of the computing and networking components. DEVS/DOC supports hardware and software component specifications (e.g., processors, networking topologies, communication protocols, software objects) of a distributed system with varying degrees of resolution and complexity in a systematic and scalable manner. It provides a characterization for representing dynamic, time-driven behavior of software and hardware components. The Discrete Event System Specification/Distributed Object Computing (DEVS/DOC) methodology and environment was proposed and developed to enable and support simulation studies of distributed object computing systems, not service-based software systems.

DOC is based on the "quantum modeling" concept which allows higher level model abstractions to be developed without precisely modeling fine level of details (e.g., detailed transport protocol modeling). The concept is primarily used in software object specification by introducing probability of method invocations. Software object interact by random selection of method executions. Since modeling the aggregate level behavior of the system is the primary objective in DOC, quantum modeling concept aids in modeling complex object interaction for aggregate system behavior. It is important to note that the DCO in DEVS/DOC supports basic concurrent execution models (i.e., none, method, and object) for the software object. The concurrency abstraction used in the software object is based on the concept of the concurrency support of the operating systems. A set of software objects executing on a mixed collection of OS's with (e.g., UNIX) and without (e.g., DOS) concurrency support behave differently. Such models of concurrent execution allow the modeler to support such scenarios with different granularity of concurrency. However, support for concurrency using multithreaded execution is prevalent in recent OS's and any standard OS (e.g., Windows XP/2000, Linux, and UNIX) fully supports multithreaded capability.

3. SOC-DEVS

Service-Based Software System design approaches largely ignore the importance of hardware or otherwise make strong simplification about the role of hardware [2, 16, 14]. Since Service-Based Software Systems depend on message interchange and computation resources, the emphasis on the "software only" design approach can leave out a critical part of the system, i.e., the underlying hardware. To address the lack of hardware representation, the co-design concept used in networked embedded systems [15] can be employed for Service-Based Software Systems. Service-Based Software Systems executing on networked hardware is similar in concept yet at a different level of abstraction compared to embedded and networked systems. Based on this observation, the emphasis in SOC-DEVS is on the introduction of the concept of co-design in Service Based Software System design (see Figure 1). To model and simulate the dynamics of service based components executing on hardware, we want to consider co-design modeling as activities to simultaneously simulate hardware and software layers of a Service Based Software System.

Unlike the term "co-design" used in the embedded systems literature where emphasis is, for example, at low-level specification of FPGA, the term "networked HW/SW co-design" [9] refers to a collection of distributed software components

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8735
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8735

executing on a collection of networked hardware components. Characterization of co-design for Service Based Software System, consequently, entails modeling and simulation for high-level specification of software and hardware layers as well as separate mappings of the former to the latter.

In the design of SBS, the networked HW/SW co-design concept allows the following

- Specification of SOA compliant service as software component and hardware components separately and establishing a well defined relation to allow synthesis as well as service mapping to hardware.
- Specification of software–software interaction and accounting for the impact of hardware resource (e.g., CPU speed and memory size) constraints.
- Accounts for the impact of multiple software component interactions that are connected by a mesh of network hardware resources (e.g., network bandwidth, router speed, and link capacity).

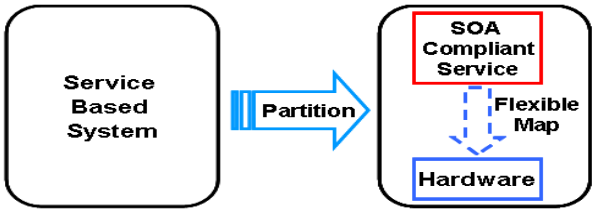

Figure 1: HW/SW co-design of SBS

The components capture the basic functional and resource capabilities of the system. The service performance is related to the hardware resources by developing an assignment between service and hardware in terms of their interactions and resource requirement. A flexible mapping provides assignment that specifies which service is assigned for execution in which hardware. We define the co-design of networked services executing on distributed hardware components as two types. First, resource constraints (CPU time and available memory) for interaction of services are restricted to a single hardware. Second, resource constraints (CPU time, available memory, communication bandwidth) for interaction of services are allowed for networked hardware components. Both types of interaction are modeled in SOC-DEVS.

4. SOC-DEVS Simulation

SOC-DEVS simulation is supported by extending the DEVS-Suite simulator [10, 6], an integrated modeling and simulation tool that supports SOA-compliant DEVS based software and hardware model development. The simulator, implemented in JavaTM, is developed using the MFVC (Model-Façade-View-Controller) design pattern. DEVS-Suite provides a scalable framework for visualization of I/O, model specific parameters, and simulation system parameters (i.e. phase, sigma, and state events) while providing capability to model software and hardware of Service-Based Software Systems. The design of the DEVS-Suite simulator (see Figure 2) separates execution control from the tightly integrated simulator kernel and view. The visualization of models and their animations are supported by modules that support user interactions and control of simulation execution. The control supports logical and soft real-time simulation execution. The simulator includes a tracking environment and time view environment. The tracking environment provides capability to simplify design of experiments for simulation models. Its graphical user interface allows a user to select model components to be monitored and thus design experiments in terms of components' inputs/outputs and state variables. Simulation model data sets, which include states such as Time of Next Event, Time of Last Event, and user selected input/output ports, can be dynamically tracked. The user, therefore, is able to observe simulation data for any number of atomic and coupled models without any code development.

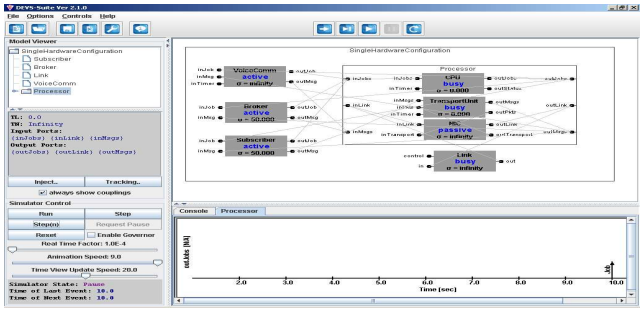

Figure 2: SOC-DEVS simulation environment

The TimeView is a module developed for run-time display of data sets as two dimensional plots (every plot has a variable y representing (input, output, or state) event coordinate and a variable x representing time coordinate). Its operation is similar to an oscilloscope. For example, number of job output of a CPU can be plotted at time instances 0, 1, 2, ..., 100. The discrete time increment duration and the units for time and variable to be plotted can be set by user plotting time-based simulation data. As an example, number of job output of a CPU can be plotted at time instances 0, 1, 2, ..., 100. The time increment duration and the units for time and variable to be plotted can be set by user.

4.1 SOC-DEVS COMPONENTS

The design objective for SOC-DEVS is to apply appropriate details required for architectural design verification and validation. As a result, detailed design specifics suitable for real system implementation is not appropriate rather the components need to account for fundamental behavior such that the service interaction through the networked hardware is captured. With this requirement along with applying the co-design concept, the SOC-DEVS is designed and developed as two modeling layers which consist of a software layer and a hardware layer. The software service (swService) in the software layer is modeled to capture basic service interaction semantics (e.g., message exchange and service invocation). The hardware layer is modeled to represent computing node and network system resources (e.g., CPU speed, memory capacity, and network bandwidth) important for composite service execution. For synthesis (i.e., combined software/hardware configuration), a mapping from software layer to hardware layer called System Service Mapping (SSM) is needed. With the services mapped to hardware components, the hardware layer acts as a constraining factor on the software layer maximum performance capability under various dynamic conditions that may exist during service interaction and system resource fluctuations.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8735
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8735

4.1.1 Software Layer

The software layer is specified based on the SOA-compliant DEVS. It consists of an abstraction for services that provide the basis for modeling composite service interactions. The software layer is extended to support such interactions with a hardware layer in the purview. Based on an extended concept for the basic service model in SOAD, the fundamental software building blocks of SOA (i.e., Broker, Publisher and Subscriber) are accounted for in the software layer.

4.1.1.1 Software Service

The basic abstraction is the "software service" which accounts for the common service properties. In designing the software service, a service in its basic form is considered as an entity with message based I/O such that a service can provide some functionality and support interaction by receiving and sending messages. Any functionality in the software service requires "operation" to be executed and the software service maintains a list of operations it can provide. A message exchange interaction is defined as communication between software services. Considering one of the services, it decodes an incoming message and returns an associated message after the execution of the associated operation.

The co-design approach requires the SW/HW interaction to be explicitly specified. Hence, in addition to supporting basic service behavior, the software service needs to account for the dependency on hardware. The software service captures the service execution under CPU and memory constraints, so each operation is parameterized with a CPU load and a memory load that determines the resource requirement on the hardware layer. The CPU load is the required CPU cycles (e.g., 1200 cycles) to complete the operation and the memory load is the amount of memory (e.g., 2 MByte) consumed while the operation is being executed. An invocation of an operation sends a job parameterized with the CPU and the memory load to the hardware layer. The software service creates a service context whenever an operation is requested and maintains a list of active service contexts currently has jobs in execution in the hardware. Any job initiated from a software service is associated with a service context which maintains the state of the operation and the message that requested the operation. The use of service context allows concurrent multi-threaded execution of simultaneous requests. Once an operation is completed a message can be sent and the associated service context is removed. The software service can support multiple operations and the operation is specified in the incoming message.

The software service is specified as a DEVS atomic model swService. It maintains an outgoing message queue and an outgoing job queue. When a swService receives a message, it decodes it to find the operation that is requested and sends a job associated with the operation to the hardware layer. Once the job completes executing, the job is returned to the swService so that the service context associated with the job can continue with the execution. Since multiple swServices can be associated with the same port, the swService receiving the completed job checks whether the job originated from itself. If so, the swService checks the service context associated with the job, creates a message and sends it to hardware layer.

4.1.1.2 Broker, Publisher, and Subscriber

The swService provides a generic skeletal support to build the fundamental SOA building blocks. The specifications for the generic Broker, Publisher and Subscriber are defined by extending the swService in the context of the message interaction each model supports and the resultant message exchanges (see Figure 3). In SOA, the message interactions among Broker, Publisher and Subscriber define the dynamics of the system. For example, a broker's response to a service look up request message from a subscriber is to perform a lookup operation on the service repository and return the relevant information to the subscriber. Similarly a publisher's response to the subscriber's service request message is to perform the service using the operation associated with service endpoint. Based on the SOA principles, the subscriber's interaction with publisher needs to be preceded by the subscriber to broker interaction. This way the SOA-compliant service interaction of SOAD is ensured in SOC-DEVS by capturing the message interactions. A message contains the node address and port value (e.g., ("swService1", 6880)) to exactly define the source and destination of the message. The service invocation is a message exchange between a publisher and a subscriber. The service invocation message invokes a service endpoint in the publisher and the associated operation(s) may be executed multiple times based on requested duration and return messages may be sent for each operation execution.

Similar to DOC, the quantum concept can be used in SOC to model aggregate level system behavior. For example, if a subscriber requests a large number of service request to publishers for multiple endpoints, a probability can be assigned to the endpoints and a random endpoint can be executed without exactly specifying in the service invocation message. However, current implementation does not support such capability.

4.1.2 Hardware Layer

The hardware layer consists of abstraction of single hardware as well as networked hardware. The processor represents single hardware on which software services can be executed. The other abstraction is the multiple processors connected via network switches and links. The interconnected processors with the network switches and links allow the modeler to account for network configuration and topology typical in a Service-Based Software System.

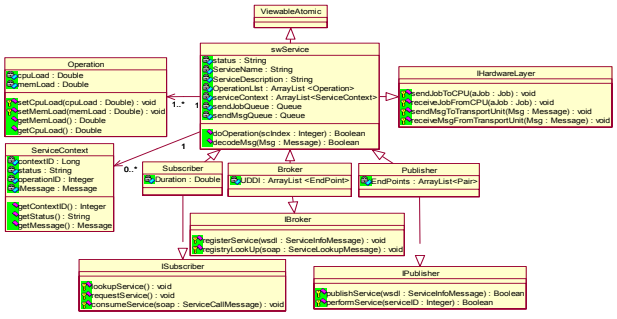

Figure 3: swService, broker, publisher, and subscriber simulation model components

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8735
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8735

4.1.2.1 Processor

The processor model is capable of performing computational work for software services and it enables these software services to interact via messages through the hardware layer. The processor is developed as a DEVS coupled model consisting of central processing unit (CPU), transport unit and network card [9]. Each of the models is developed as DEVS atomic model.

The CPU is specified as a DEVS atomic model with one input port, one output port, and two parameters (i.e., CPU speed, and memory size). The "processor.inJobs" input port accepts requests from software service to execute jobs. The competed jobs are emitted via the "processor.outJobs". The CPU speed parameter determines how quickly data processing operations are executed and the memory size determines the number of jobs that can be loaded without using swap memory. The CPU speed and memory size constraints software services in their competition for CPU time and memory resources as well as the rate at which jobs from software services are processed. If the available memory is insufficient for an incoming job, the job is put into a waiting queue. Once memory becomes available, the job is put into the active queue with a swap time penalty. The model of CPU is different from DEVS/DOC where software layer decides when to swap in/out and instructs the CPU to load into disk and load into memory. Our approach allows software layer to be more concerned with service level behavior. Behavior appropriate for OS (e.g., FIFO scheduling algorithm) is included in the CPU. The CPU can also be specified to support other kinds of scheduling algorithms without any change to the software service.

The transport unit provides message I/O including message segmentation into packets and reassembly of message from packets. A message contains data abstractions for the software layer and packets are data abstractions for the network layer. Messages are transported to the software layer or fragmented into packets to the network card based on the message to software layer mapping. Outgoing messages for the network card are fragmented before sending and incoming packets from the network card are queued for reassembly and then send to the software layer. The transport unit at a destination node receives and collects packets. When all packets for a message are received, the destination transport unit delivers the message to the destination software service. It must be noted that the transport unit does not account for complex transmission control rather it provides basic transport capability like packetization (i.e., data size) overhead, and end-to-end communication with message to software layer mapping. In contrast to DEVS/DOC, the transport layer is extended to support node address and logical port embedded in messages so that source and destination can be exactly identified. The network card provides network I/O for incoming and outgoing packets. The network I/O is also buffered (i.e., fragments are queued) to prevent packet loss during packet transmission.

4.1.2.2 Link

Link is the abstraction for physical medium used to interconnect networks. Use of link is only suitable when the modeler is interested in detail link layer behavior (e.g., propagation delay, frame collisions). For SOC-DEVS, we model link with an input and output queue that can transmit packet fragments to/from network card (also network switch) at a specified speed (e.g., 100 Mbps). Unlike DEVS/DOC, the link model does not have an error coefficient to emulate physical level noise.

4.1.2.3 Network Switch

Network switch is developed as DEVS atomic model. It is used to route packets among more processors. The network switch queues incoming packets and puts them in the outgoing queue after processing. The bandwidth of the outgoing links and the queue length are two important parameters that can be used to configure various packet loss scenarios. The packet processing is done after an address lookup (i.e., input link to output link mapping) in the routing table. Packet address information is used to make switching decisions to send a packet to a specific output link as necessary. Unlike in real network switch where the address mapping is done using routing algorithm that automatically updates routing table [20], the modeler needs to specify and initialize static address lookup tables in network switches to represent a network topology under consideration.

4.1.3 Service System Mapping

The Service System Mapping (SSM) provides the assignment of software services to processors. The flexible mapping in SSM allows a modeler to assign a software service to processor with different configurations. From implementation perspective, it couples the "swService.outMsg" port of each software service component to the "processor.inMsg" port of the processor. Similar couplings are made for "swService.outJob" to "processor.inJobs". It also couples the "processor.outMsgs" port of the processor to the "swService.inMsg" port for each software service component. Similar couplings are made, as well, for "processor.outJobs" to "swService.inJob" (see Figure 4). Jobs from the service are sent to processor and the completed jobs are returned. The messages are sent to the transport unit and disseminated to the destination processor. Also incoming messages are delivered to the software services. As a result, the couplings facilitate the interactions during simulation. The mapping from software to hardware layer is applicable for atomic services and as well as composite services as the interface design is applicable for a generic swService interaction (see the IHardware Layer component in Figure 3).

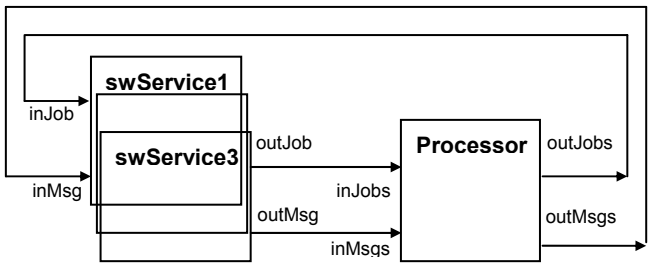

Figure 4: System Service Mapping

The behavior and performance of a SBS is dependent on both the software service and the hardware layer. Dynamic behavior is specified in the software service by computational load and service interactions. By mapping the software services onto processors, the dynamics of the software service are constrained by the capabilities and topology of the hardware layer. As the software services compete for processor resources in terms of memory to load and CPU cycles to execute, their dynamics drive the performance of the processor which in turn also determines

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8735
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8735

the performance of the software service to software service interaction which in turn also drive the dynamics of the network performance and hence the QoS of the SBS. In single hardware configuration (see Figure 5a), services get mapped to a single processor. In networked hardware configuration (see Figure 5b), services get mapped to multiple processors interconnected by network switches or links.

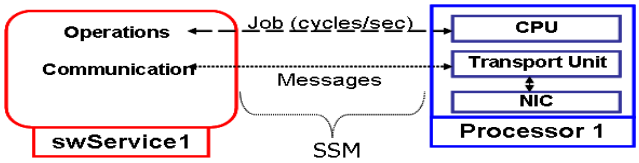

Figure 5a: Single Hardware

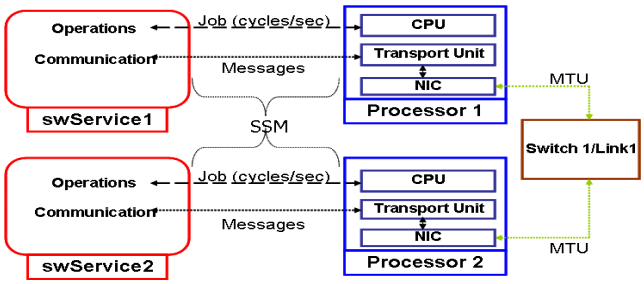

Figure 5b: Networked Hardware

5. Simulation Example

5.1 Voice Communication System

Proliferation of VoIP and digital music makes audio data transmission a major contributor to internet traffic. Based on end user preference and policies of service publishers, real time audio traffic may require minimum QoS to be maintained under various network traffic scenarios. For the case study, we consider an end-to-end Voice Communication System (VCS) capable of streaming audio data to the subscribers. The VCS publishes various quality audio data specified by sampling rates (e.g., ranging from 44.1 to 220.5 KHz) and the interested subscribers can subscribe to the channels with certain QoS constraints on the data quality [18]. The higher sampling rates produce higher quality audio data as it encodes more audio information per second. For example, sampling rate of 220.5 KHz will produce superior quality audio data w.r.t. 44.1 KHz sampling rate. The VCS under consideration supports a 2-channel (i.e., stereo) audio data that can be sampled at any of the following rates – 44.1, 88.2, 136.4, 176.4, 220.5 KHz. The subscribers request audio data stream for a specified amount of time over the network and expect the VCS to ensure QoS. The subscriber requests are processed by the VCS and it streams audio data for the specified duration. The VCS can support multiple subscribers simultaneously such that each subscriber may request different quality audio data. The throughput provides a measure of the VCS performance. In general (i.e., under normal operating conditions), the VCS throughput is proportional to the number of audio streams being delivered. The VCS also supports secured voice communication by encrypting sampled voice data. If the subscriber requests encrypted data, the VCS encrypt sampled data with 256 key DES [20] algorithm prior to sending to the subscriber.

5.2 VCS Simulation using SOC-DEVS

To exemplify the SOC-DEVS approach, the above Voice Communication System (VCS) is modeled. The real VCS is implemented in C# .NET. The VCS represents a number of subscribers subscribing to a voice publisher. A model of this system is simulated. The resultant throughput behavior is compared to similar scenario (i.e. testbed setup with same configuration) in the real VCS. The effect of resource fluctuation on the QoS is also observed.

To illustrate the core capabilities of the SOC-DEVS approach, we consider the scenario with a single VCS, a broker and five subscribers in a networked hardware system consisting of two processors connected via a network switch. The VCS publisher and the broker are mapped on a single processor and the five subscribers are assigned to the other processor. Each subscriber requests DES encrypted voice data at 220.5 KHz sampling rate for 2 channel stereo data encoded in 16 bits. A multiple subscriber scenario is used with 1-5 subscribers active at a time in each scenario. The dynamics of the system is captured by the average voice publisher throughput over 60 simulated logical seconds. To observe system behavior under heavy network load condition, a background traffic generator is used at the network switch at 75 Mbps rate. For a comparative view, we also run the same scenario in the real VCS system (see Table1) and collected the voice throughput data, CPU utilization at the processor assigned to the publisher. The results are discussed in Section 5.4.

Table 1: System configuration

Category	Real System	Simulation System
Processor (CPU, Memory, Network Card)	2.2 GHz, 1024 MB, 100 Mbps	2.2 GHz, 1024MB, 100Mbps
Network Link Bandwidth	100 Mbps	100Mbps
Subscriber Number	1-5	1-5, 10, 20
Data Collection Duration	60 sec (wall clock)	60 sec (logical clock)

5.3 Simulation Parameter Estimation

The real system uses DES [5] encryption algorithm. However, the simulated VCS model does not implement the logic for DES rather accounts for the effect on encryption on the CPU load as a function of sampling rate. As a result, the simulation setup needs CPU load factor approximation to account for the encryption operation in the CPU. If data encryption is disabled, the CPU load is negligible. Now, the real VCS samples voice data at 44.1−220.5KHz rate and generated data is encrypted by DES algorithm. Then data generation rate, G = S*B*C bits/sec where, S denotes the Sampling rate (KHz), C is the Channel number (mono or stereo), B denotes bits per sample. Under nominal load condition in the real system, the CPU utilization is primarily due to the encryption load. Hence, if the encryption rate is E then E=G. Now, CPU load factor, LF = (V*U)/E; where V= CPU speed (MHz), U = CPU utilization (%). Replacing E=G we get the final equation, LF = (V*U)/G. Also for a time duration T, the average CPU load for encryption operation, L = LF*S*T. For

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8735
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8735

example, S=44.1 KHz, C=2, B=16 bits/sample, V=2200MHz, U=5.7%, we get, LF =704 cycles/byte. Also the average load on the CPU is, L = 125.334 Mcyles.

The processor parameters in the simulation are assigned based on real system's CPU configuration. The CPU speed for the processor is set at 2.2 GHz and 1024MB of memory. For processor configuration test, the CPU speed of 3.2 GHz and 1024 Mbyte of memory is used. The swap penalty is set for 0.1 logical second. The network parameter is 100 Mbps which is the bandwidth of I/O link at the network switches and network cards.

5.4 Discussion

In demonstrating the capability to capture dynamic characteristics of Service-Based Software Systems in SOC-DEVS, we traced the average throughput and CPU utilization of the real and simulated VCS under similar scenarios. The objective is to observe the VCS behavior and its QoS under varying load conditions by varying the number of simultaneously active subscribers.

In Figure 6a, the simulated vs. real throughput of the VCS is shown. In this scenario the data encryption is disabled. A linear increase in VCS throughput with increasing number of subscribers is observed. This is comparable to the real system, where each active subscriber increases the streaming data throughput under normal operating conditions. Approximately,

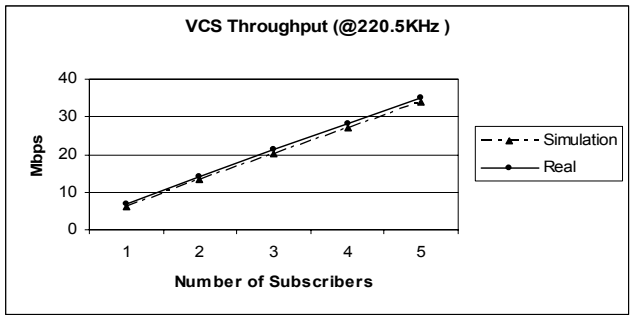

Figure 6a: Average VCS throughput for Sampling rate =220.5KHz and no background traffic

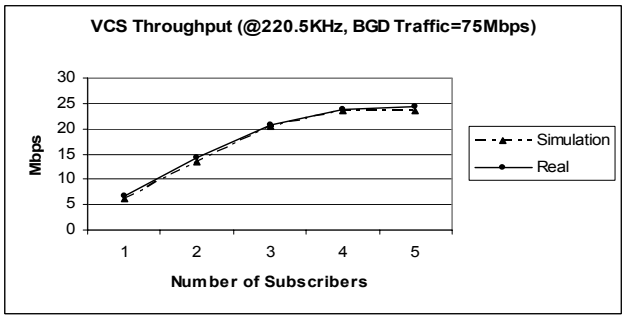

Figure 6b: Average VCS throughput for Sampling rate=220.5KHz and background traffic=75Mbps

35% of 100Mbs bandwidth is used by the active 5 subscribers. In Figure 6b, the real vs. simulation throughput of the VCS is shown. In this scenario, the network has background traffic to compete with VCS traffic. A linear increase in VCS throughput with increasing number of subscribers is observed up to 25Mbps for 4 subscribers. This is comparable to the real system, where each active subscriber increases the streaming data throughput under normal operating conditions (up to 25% of 100Mbs bandwidth is used by the active 4 subscribers). Throughput is clamped at 25 Mbps (appx.) for 5 subscriber scenario due the background traffic consuming 75% bandwidth at the network switch.

In the Figures 7a and 7b, the CPU utilization with the number of active subscribers along with CPU saturation effect on the VCS throughput is plotted. In this scenario, data encryption is enabled and the CPU load reflects the data encryption load. With increased CPU load and increasing subscribers, the software layer reaches a saturation point such that the effective throughput is not sufficient to use the available network bandwidth (note: no background traffic is used to load the network switch). It is evident that the CPU utilization increases with increasing client number and the CPU becomes saturated which effects the VCS throughput. Though the saturation differs with respect to the real system where various background threads interact in a complex manner, the trend line similarity verifies the effect of the system dynamics clearly as required of SOC-DEVS as an early architectural verification and validation tool for SBS.

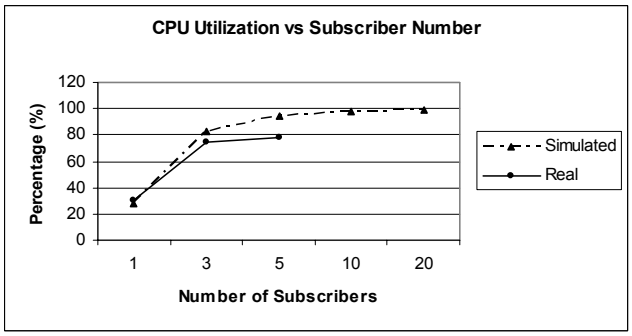

Figure 7a: CPU utilization and subscriber number

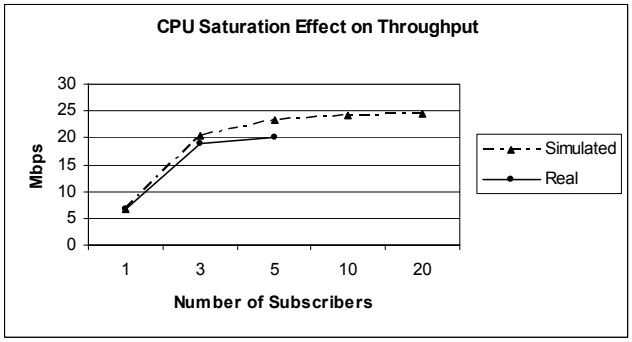

Figure 7b: CPU saturation effect on VCS throughput

In the Figure 8a, the processor configuration is modified by changing the CPU speed [from (2.2 GHz, 1024 MB) to (3.2 GHz, 1024 MB)] to demonstrate how processor configuration can improve the throughput behavior. For the same load scenario as in Figure 7a the increase in CPU speed improves the software layer execution speed with reduced CPU saturation and the effective cap on the VCS throughput is increased. The improvement reason is evident from Figure 8b, as the reduction in average CPU queue length denotes improved execution rate and hence increased VCS throughput.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8735
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8735

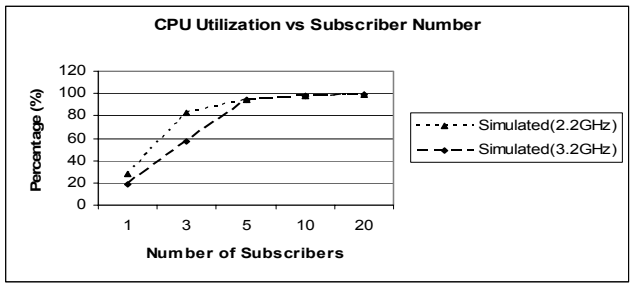

Figure 8a: Average VCS throughput on different CPU speeds

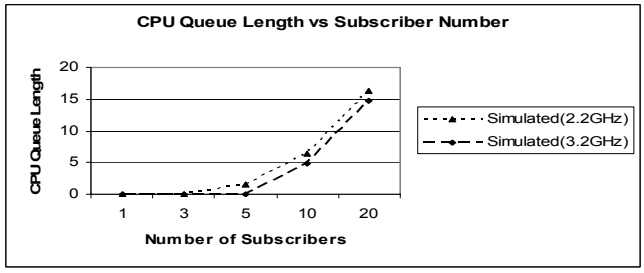

Figure 8b: Average CPU queue length on different CPU speeds

6. Conclusion & Future Work

In this research, an extension to the DEVS-Suite simulator supporting co-design concept is presented. It aids developing simulation-based SBS architectural design and verification. The SOC-DEVS simulation capability provides a systematic way of designing Service-Based Software Systems by simulating the synthesized software and hardware components where improved observation capability is supported due to explicit software and hardware interactions. The simulator is suitable for supporting a generalized integrated design process as in [15] which is similar to well established co-design based approaches (particularly in embedded systems). However a co-design process for SBS design and development is a new dimension yet to be explored. As part of future research, we intend to further investigate the co-design approach as a standard in SBS design and development process.

7. ACKNOWLEDGMENTS

This research is supported by NSF Grant #CCF-0725340. The real Voice Communication System and the DES service are developed by our collaborators Ho An, Dazhi Huang, and Dr. Stephen Yau.

8. REFERENCES

[1] Ambrogio, A. D', P. Bocciarelli, 2007, "A Model-Driven Approach to Describe and Predict the Performance of Composite Services", Proceedings of the 6th International Workshop on Software and Performance, 78-89, Buenos Aires, Argentina.

[2] Bause, F., P. Buchholz, J. Kriege, S. Vastag, 2008, "A Framework for Simulation Models of Service-Oriented Architectures", Lecture Notes in Computer Science: Performance Evaluation: Metrics, Models and Benchmarks, Vol. 5119, 208-227.

[3] Business Process Execution Language for WebServices (BPEL), Version 1.1, http://www.ibm.com/developerworks/library/specification/ws-bpel/, 2009.

[4] Butler, J.M., 1995, "Quantum Modeling of Distributed Object Computing," Simulation Digest, Vol. 24, No. 2, 20–39.

[5] DES, Data Encryption Standard, http://www.itl.nist.gov/fipspubs/fip46-2.htm, 2010.

[6] DEVS-Suite, http://devs-suitesim.sourceforge.net/, 2009.

[7] Erl, T., 2006, Service-Oriented Architecture Concepts, Technology and Design, Prentice Hall.

[8] Gibbs, J.D., H.S. Sarjoughian, 2009, "Assessing the Impact of a Modeling Tool and its Support for Verification and Validation", International Symposium on Performance Evaluation of Computer and Telecommunication Systems, 73-80, Istanbul, Turkey.

[9] Hild, D.R., H.S. Sarjoughian, B.P. Zeigler, 2001, "DEVS-DOC: A Modeling and Simulation Environment Enabling Distributed Codesign", IEEE SMC Transactions-Part A, Vol. 32, No. 1, 78-92.

[10] Kim, S., H.S. Sarjoughian, V. Elamvazhuthi, 2009, "DEVS-Suite: A Simulator for Visual Experimentation and Behavior Monitoring", High Performance Computing & Simulation Symposium, Proceedings of the Spring Simulation Conference, San Diego, CA, ACM Press.

[11] ns-2, network simulator, http://www.isi.edu/nsnam/ns/, 2009.

[12] OMNeT++, http://www.omnetpp.org/, 2009.

[13] OPNET, http://www.opnet.com/, 2009.

[14] Sarjoughian, H.S., S. Kim, M. Ramaswamy, S.S. Yau, 2008, "A Simulation Framework for Service-Oriented Computing Systems", Winter Simulation Conference., 845-853, Miami, FL, USA.

[15] Schulz, S., J.W. Rozenblit, M. Mrva, K. Buchenrieder, 1998, "Model-Based Codesign", IEEE Computer, Vol. 32, No 8, 60-68.

[16] Tsai, W.T., Z. Cao, X. Wei, R. Paul, Q. Huang and X. Sun, 2007, "Modeling and Simulation in Service-Oriented Software Development", Simulation Transactions, Vol. 83, No 1, 7-32.

[17] Wolf, Wayne H., 1994, "Hardware Software Co-design of Embedded Systems", Proceedings of the IEEE, Vol. 82, No. 7, 969-989.

[18] Yau, S.S., N. Ye, H.S. Sarjoughian, D. Huang, A. Roontiva, M. Baydogan, and M.A. Muqsith, 2009, "A Performance-Model-Oriented Approach to Developing, Adaptive Service-based Software Systems", IEEE Transactions on Service Computing, In Press.

[19] Zeigler, B.P., H. Praehofer and T.G. Kim, 2000, Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Second Edition, Academic Press.

[20] Zengin, A., H.S. Sarjoughian, H. Ekiz, 2008, "Study of Biologically-Inspired Network Systems: Mapping Colonies to Large-scale Networks", 20th European Simulation Conference, 537-545, Amantea, Italy.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8735
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8735