A Co-Design Modeling Methodology for Simulation of Service Oriented Computing Systems

by

Mohammed Abdul Muqsith

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved November 2011 by the
Graduate Supervisory Committee:

Hessam Seyed Sarjoughian, Co-Chair
Sik-Sang Yau, Co-Chair
Dijiang Huang
Wei-Tek Tsai

ARIZONA STATE UNIVERSITY

December 2011

ABSTRACT

The adoption of the Service Oriented Architecture (SOA) as the foundation for developing a new generation of software systems - known as Service Based Software Systems (SBS), poses new challenges in system design. While simulation as a methodology serves a principal role in design, there is a growing recognition that simulation of SBS requires modeling capabilities beyond those that are developed for the traditional distributed software systems. In particular, while different component-based modeling approaches may lend themselves to simulating the logical process flows in Service Oriented Computing (SOC) systems, they are inadequate in terms of supporting SOA-compliant modeling. Furthermore, composite services must satisfy multiple QoS attributes under constrained service reconfigurations and hardware resources. A key desired capability, therefore, is to model and simulate not only the services consistent with SOA concepts and principles, but also the hardware and network components on which services must execute on. In this dissertation, SOC-DEVS - a novel co-design modeling methodology that enables simulation of software and hardware aspects of SBS for early architectural design evaluation is developed. A set of abstractions representing important service characteristics and service relationships are modeled. The proposed software/hardware co-design simulation capability is introduced into the DEVS-Suite simulator. Exemplar simulation models of a communication intensive Voice Communication System and a computation intensive Encryption System are developed and then validated using data from an existing real system. The applicability of the SOC-DEVS methodology is demonstrated in a simulation testbed aimed at facilitating the design & development of SBS. Furthermore, the simulation testbed is extended by integrating an existing prototype monitoring and adaptation system with the simulator to support basic experimentation towards design & development of Adaptive SBS.

To my parents & Saeed bhai.

&

To my unborn child for whom I eagerly wait.

ACKNOWLEDGEMENTS

I would like to thank Dr. Hessam Sarjoughian for his guidance and suggestions. Without his advice, feedback and incremental refinement approach - it would have been a much more difficult journey for me.

I would like to thank Dr. Stephen Yau for his guidance, suggestions and feedback in the presentation/review sessions in the regular group meetings of the National Science Foundation (NSF) Science of Design (SoD) project.

I would also like to thank my committee members Dr. Dijiang Huang and Dr. Wei-Tek Tsai for their comments and constructive feedback in reviewing my dissertation.

As a member of the NSF SoD project team at ASU for the last 3 years, I gained valuable experience in software system simulation, design and development. Thanks to Ho An, Yin Yin, and Dazhi Huang who worked tirelessly in developing the real Voice Communication System and the encryption services used in conducting the real experiments. Also, thanks to Dr. Nong Ye, Billibaldo, Suseon for their help in data analysis. Special thanks to Dazhi Huang for sharing his valuable insight in system design and development.

I am grateful to my wife, Kahkashan, for her support in this endeavor and standing by me throughout the journey. It would have been extremely difficult to pursue the path of PhD without her unwavering support.

I thank my parents, Mohammed Abdul Hasib and Noorjahan Begum, for their blessings that have helped me stay on course. I am grateful to my elder brother Muquith, sister-in-law Humaira, sisters Atika and Aziza, and my brothers-in-law Saeed Bhai and Zaki Bhai for extending their support and words of encouragement throughout these years. Special thanks to my nieces Nazha,

Raima, and Mahera & nephews Rehan, Mehran, and Amer for the joy and privilege of having them by me. At times of stress, their warmth gave me the much needed peace of mind and respite from all the pressure.

Finally, I would like to thank the National Science Foundation (NSF) for supporting this research under Science of Design Grant # CCF-0725340. I would also like to thank Intel Corporation for supporting part of my research under research grant # 4875173.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The dissertation focuses on Service Oriented Computing (SOC) Systems from a system modeling perspective and emphasizes on the fundamental concepts and techniques towards developing a methodology in supporting simulation of SOC systems. This chapter covers the motivation for research in simulation based design of SOC systems. In addition, the research problem is formulated along with the outline of associated challenges. Finally, a brief overview of the contributions of the dissertation is highlighted.

1.1 Motivation

System design is challenging. Designers need to understand the requirements and the objective of the system and transform the requirements into a cost effective design that can meet the desired objectives. Conflicting requirements of minimizing system development cost while ensuring all requirements are fulfilled is common [3, 47]. Evaluation of a system under design at an early phase is desirable as it can highlight system characteristics since cross interaction between factors can be exposed - specifically from system performance perspective. Researchers hence try to develop new approaches,tools and methods to aid in design process through early system evaluation techniques. In this context, modeling and simulation can play an important role in early design evaluations. Simulation based design and evaluation of software intensive systems can provide insight into the system performance that are otherwise impossible or impractical [28, 6].

Interestingly, simulation modeling can be challenging and error prone requiring significant verification effort. However, in the absence of a real system, a simulated system can be effective in aiding in architectural (i.e., high-level) design decisions by providing early analysis capability of complex system dynamics. An

important aspect of simulation is that it can capture time based behavior of a system. This is in contrast to non simulation based design where system structure and component relations are defined and prior to system evaluation at a late stage when all system components are realized and deployed - there exists little or no quantified measure of system performance. From a performance evaluation point, such design approaches have limitation in capturing cross interactions among system components that may arise at runtime. For example in software system design, Architecture Description Language [1] can provide architecture of the system while Unified Modeling Language provide design of system that can be mapped to software modules [66]. However, without the concept of precise timing, system specification are limited in expressing time based aspects and representing system performance - specifically in terms of time-varying system dynamics.

Simulation has been extensively used in design and evaluation of embedded systems. Over the last decade, simulation based design of embedded systems has become the standard. Such systems are generally designed towards specialized applications (e.g., control systems) with application specific hardware and software design. The early design evaluation in embedded systems is important as such systems are resource constrained in terms of computation power, speed, memory, communication bandwidth, energy availability and the design challenges include space (i.e., physical layout), power and computation resource constraints [8, 14]. Within the embedded systems domain, Network on Chip (NoC) is a highly specialized systems that are concerned with efficient communication among subsystems and designers apply networking theory, methods and tools to improve data communication speed among intellectual property (i.e., IP) cores, processors and memory while utilizing minimal chip area [14]. NoC is targeted towards improved scalability of the system where efficient communication among subsystems is of primary importance. In general, embedded system design emphasizes more

on optimizing hardware resource utilization while improving system performance. Design space exploration is targeted towards improving system performance using highly specialized hardware.

In contrast, networked systems are targeted towards providing a hardware independent computing platform using heterogeneous (w.r.t. hardware as well as software) systems that are interconnected via network infrastructure. Networked systems evolved to support large scale systems with significant computation resources - distributed in disparate geographic locations, yet able to seamlessly interact via standardized communication mechanism and interconnection infrastructure. Such systems generally consist of non-specialized hardware with software stacks providing generic application interfaces developed to ensure seamless access to computation resources. A classic example of a networked system is the Internet. In this context, a more recent development is the concept of computing in the Cloud [68] which envisions resources in the system exposed as services and seamlessly accessible according to the end users' service level agreement. Software as a Service (SaaS), Infrastructure as a Service (IaaS), Platform as a Service (PaaS) are the key notions in Cloud Computing - each of which accounts for various aspects of the system towards the Cloud based computing paradigm shift. However, while the concept of Cloud is promising, understanding the cloud based system depends on knowledge of the fundamental system dynamics of Service Oriented Computing (SOC) systems - as Cloud is evolving on the foundation and concept of services and Service Oriented Architecture (SOA) [23, 68, 46]. From system modeling perspective, a thorough understanding of implications of design choices in Service Oriented Computing systems can play an important role towards developing Cloud based systems. As such, a modeling approach based on SOA principles can serve as a pivot towards simulation based design of SOC systems [53].

Service Oriented Architecture (SOA) [23] is the architectural principle for developing SOC systems and software systems based on SOA is known as Service Based Software Systems (SBS) [76]. As such, simulation based design of SOC systems essentially entails developing models for SBS components. So, an important factor to consider is the appropriate abstractions of the system and develop models based on this conceptualization. As SOA provides architectural specification for developing software system, the focus is on the software aspect of the system (i.e., services). However, SBS (in a broader sense) is a networked system and thus contain hardware elements in addition to software ones. Like any networked systems, SBS requires services mapped (e.g., service hosting) onto alternate networked hardware components. As such, a SBS modeling approach without a suitable abstraction for representing the system hardware components or its impact can be considered inadequate. From a SBS designer perspective with interest in architecture level performance evaluation using simulation, a holistic conceptual view of the system in terms of software and hardware parts is important as the designer gains flexibility in evaluating performance under various system configurations (i.e. services mapped onto alternate hardware components) that are impractical otherwise. To address both software as well as hardware aspect of the system, co-design modeling [71, 55] allows representing a system in terms of software, hardware elements and provides systemic synthesis capability. Thus, applying the concept of co-design in SBS modeling can account for network hardware components - an important aspect of SBS, while providing a systematic approach towards design and development.

Currently research in Service Based Software System modeling and simulation approaches emphasize on process specification and workflow aspect of services [9, 62, 73, 74, 7, 2] and the QoS dependency on system resources are not accounted. While a few approaches [44, 45, 42, 35] support representation soft-

ware and hardware aspect, the concept of service is absent. In DEVS/DOC [28], distributed co-design concept is used towards modeling distributed object systems and it is not intended for representing services. In [12], the approach is focused on resource scheduling aspect in cloud systems and model abstractions are targeted for the virtualized resource environment, it does not account for concepts of SOA, co-design or the representation of physical hardware resources important in system QoS evaluation. In [53], service models are developed using SOA concepts and principles but it lack the concept of co-design. In essence, existing networked system modeling approaches [1] do not account for concepts, artifacts and interactions of SOA. Thus the existing modeling approaches and tools are limited (or incapable) in representing SBS. In addition, formal model specification with a capability to precisely specify timing and structural aspect of SBS is important as time dependent system dynamics are critical in performance evaluation. In this context, Discrete EVent System Specification (DEVS) can provide a strong theoretical basis towards SBS model specification and formal verification.

To overcome the limitations of the existing modeling methodologies, the following elements are brought together to support combined software and hardware simulation of Service Based Software Systems, namely -

1. Co-design concept
2. SOA concepts and principles
3. Formal model specifications

The motivation in the dissertation is developing a modeling and simulation methodology based on these combined elements. Such methodology can provide software and hardware separation and flexible synthesis capabilities in simulation

[1] Details are discussed in related work section

based evaluations and thus aid in developing a desired Service Based Software System architecture.

1.2 Research Problem Description

Service Oriented Computing (SOC) is the computing paradigm based on Service Oriented Architecture (SOA) [23]. SOA is the architectural principle that serves as the guideline towards building such systems and considered an attractive approach for developing enterprise scale distributed software systems. It defines a set of loosely coupled computational components called "services" that interact to provide functional utilities to interested clients. All the software resources in SOA are termed as services. Each service is a well defined self contained software module providing functionality to interested clients. SOA emphasizes loosely coupled, protocol independent distributed system development with the "software as a 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, 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 and its constituent parts.

Design and configuration of Service Based Software System demands making trade-offs among multiple QoS features. Design decisions spanning software, hardware, and their combination have significant roles in achieving the desired runtime QoS. Satisfying multiple quality of service 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 unavailable due to various system and network failures, overloads or

other causes. To attain a level of tractability in developing such systems, the use of modeling and simulation tools as an aid in alternative designs is proposed as a necessity by some experts [34, 28]. To achieve the goals set forth for service-oriented computing, a growing number of researchers are formulating detailed concepts, methods, and techniques that can be used to build Service Based Software Systems. The most common approach in defining a systems structure and behavior is to develop models. The choice of a model is driven by the role it can play in the system development and operation life-cycle [15, 10, 24]. Models can be developed to define technical requirements and architectural design of a Service Based Software System. Such models may represent dynamics of the services and their interactions in such a way to study the systems capability to support the quality of service attributes such as performance, timeliness, accuracy, and security.

To design Service Based Software Systems capable of satisfying multiple Quality of Service (QoS) attributes, simulation-based modeling is desirable as simulation can play a central role in enabling tradeoff study among time-based quality of service attributes. To develop the SBS framework and design an SBS prototype, we can develop a set of simulated services that accounts for SOA concept and principles. Use of such simulated services enables simulation based analysis and design capabilities that are impractical using testbed with real services.

One aspect of service based systems is that their highly intricate dynamics that are rooted in architectural design choices and constraints. The difficulties of designing a large-scale service based software system provide challenging research questions for simulation-based system modeling. Developing solutions to these questions is considered important fundamental research with key implications in the engineering of software systems. Another aspect of a service based system is that the software components can be mapped onto alternative hard-

ware components. Therefore, a basic concept in simulation aided system design is to distinguish between software and hardware components while allowing the flexibility to synthesize these different components to create the desired system architectures.

To create service based systems and in particular support simulation-based system design, it is advantageous to bring together concepts, methods, and practices from modeling and simulation, software engineering, and system engineering. Modeling and simulation has been widely applied in system engineering which provides an overarching framework for defining, developing and deploying systems [47]. Analysis and design are essential parts of system engineering as well as software engineering. In order to develop design and analysis models for service based systems, a system has to be decomposed into its comparatively simpler components. Decomposition of a system into a set of cohesive software and hardware components remains challenging for engineers. Similarly, it is difficult to select and fit a component appropriately into a systems architecture. The structure and behavior of the components of a system affect the overall system and, conversely, the structure and behavior of the system influences the choices of the components.

The challenges associated with SBS design & development is conceptually similar to those that have been considered for embedded systems [71] and networked systems (e.g. DEVS/DOC [28]) . To allow combined model-based software and hardware design, the concept of co-design [71, 55] was developed. It allows designers to simultaneously account for requirements that span to both the software and to the hardware on which the software is expected to execute. The separation of embedded software and hardware and the successful application of co-design offer a strong case for their use in designing service based systems. Indeed, a co-design methodology has been developed for simulation-based analysis and design of systems. Engineers can try different designs with the convenience

and systematic use of co-design simulation models that separate and integrate software and hardware parts of systems. Enabling the design of service based systems, however, requires co-design capabilities such that software models are SOA compliant.

Considering the complexities involved in design and development of Service Based Systems, co-design based simulation can aid in early design phase by providing testbed with capabilities to specify, build, experiment, and evaluate alternative Service Based Software Systems. The modeling and simulation of such complex system brings up a number of research challenges to be addressed −

1. Representing a service based software system in terms of software service and network hardware models and support their syntheses. Service models and network hardware can change independent of each other and their flexible synthesis should allow evaluation of alternate system configurations.

2. Modeling SOA-compliant services at a level of abstraction suitable for architecture level simulation. Service models need to account for the fundamental models (Broker, Publisher, Subscriber) and their interrelations to be SOA-compliant.

3. Developing generic service and hardware models and their interactions to relate service activity with the system QoS. Service models need to be detailed enough to account for essential service properties and capable of representing the software aspect of SBS. The networked hardware model needs to be able to support network topology, communication capability and represent hardware resources like cpu cycles, memory, link bandwidth. The interactions need to account for service to service interactions via hardware within the constraints of underlying networked hardware resources.

4. Supporting architecture level simulation of SBS for design evaluation in terms of system QoS such as service delay, service throughput, network throughput and processor utilization.

1.3 Summary Of Contributions

This dissertation presents SOC-DEVS co-design modeling methodology for architectural simulation of Service Based Software Systems. In SOC-DEVS, the concept of SW/HW co-design is infused with the SOA-Compliant DEVS modeling. A simulator is developed to implement the methodology that allows system engineers and designers evaluate Service Based Software Systems and allows separation and synthesis of alternate software services and network hardware components. The developed simulator is used in a simulation based experimentation testbed that support experimentations for early analysis of system architecture and dynamics that are challenging to accomplish using real systems (e.g., complex distribution/hosting of services under various hardware configurations). The simulation testbed allows exploration of design alternatives using system QoS as the evaluation criteria. The testbed is extended by integrating the simulator with a prototype monitoring and adaptation system for experimentations in design of SBS with adaptation capability [33, 76].

Compared to existing approaches, the co-design modeling of SBS provides a layered view of the system with clear separation of concern in terms of software services and networked hardware components. The details of the software service model is capable of representing fundamental properties (e.g., statelessness, discover-ability, publish-ability) as opposed to a generic software component - which is an important distinction w.r.t. existing co-design approaches that support system view in terms of software and hardware models (e.g., DEVS/DOC, DEVS/NS2, OMNET and OPNET) but cannot account for the service concept and representation [53]. In addition, the interaction via the networked hardware

captures important time dependent behavior of a service (e.g., service delay, service response time). The synthesis of software service and networked hardware is based on a flexible mapping which allows ease of alternate system configuration evaluation without changing the models. The generic models are built using first principle and system models are developed with service model and hardware model synthesis. The applicability and usefulness of the methodology has been shown by building exemplar models that can capture fundamental behavior observable in real systems. Development of the exemplar models and model verification using real system's data demonstrates that the approach and the abstractions are suitable for representing SOC systems and its fundamental dynamics - computation and communication aspect. Finally, the simulation study using the proposed modeling methodology underlines the importance of co-design modeling in Service Based System design and early architectural evaluations.

A brief summary of contributions of the dissertation is as follows :

1. Developed SOC-DEVS, a SOA-Compliant modeling and simulation methodology that applies co-design concept in the design of SBS systems.

2. Developed SOC-DEVS simulator based on SOC-DEVS methodology.

3. Developed a simulation based experimentation testbed for early architectural SBS design evaluation.

4. Integrated SOC-DEVS simulator with prototype monitoring and adaptation subsystem [2].

It is important to note that concept of co-design and its application in simulation based design is not new. However, application of the co-design concept in modeling SBS has not been explored in existing literature on SBS modeling and

[2]Prototype monitoring and adaptation system is developed by Dazhi Huang

simulation. In addition, SOA related important research aspects such as simulation of SBS in a SOA environment (e.g. DEVS/SOA [54]) or multi-tenancy [4] support in SOA application development is beyond the scope of this dissertation. Rather the focus and thus the contribution of the dissertation is in developing a modeling approach based on the concept of co-design and SOA to represent fundamental artifacts of SBS system. As such, the modeling abstractions are targeted towards representing core software elements of SOA that are based on first principles with support for networked hardware representation.

Chapter 2

BACKGROUND

In this chapter, an overview of the background related to the dissertation is highlighted. The chapter highlights the Service Oriented Computing (SOC) and outlines Service Oriented Architecture (SOA) as the guiding principles towards building SOC systems. In this context, Service Based Software Systems (SBS) are addressed from a modeling perspective. The concept of co-design is explained and discussed in the context of SBS modeling.

2.1 Service Oriented Computing

Service Oriented Computing is a paradigm of computation that is based on the concept of service. Services are autonomous, platform-independent, loosely coupled, publishable, discoverable, composable and re-usable entities that provide functionalities to interested clients in a subscription based manner [46].

While SOC is the model of computation, the principle towards building a SOC system is provided in Service Oriented Architecture (SOA) [23]. In SOA, services are defined using platform independent languages, provide publishable interfaces, and interact with each other (as well as the clients) to collectively execute a common task. In addition, each service is independent of the state and context of other services,thus making services stateless. The interaction and communication is done using protocol independent message scheme. 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

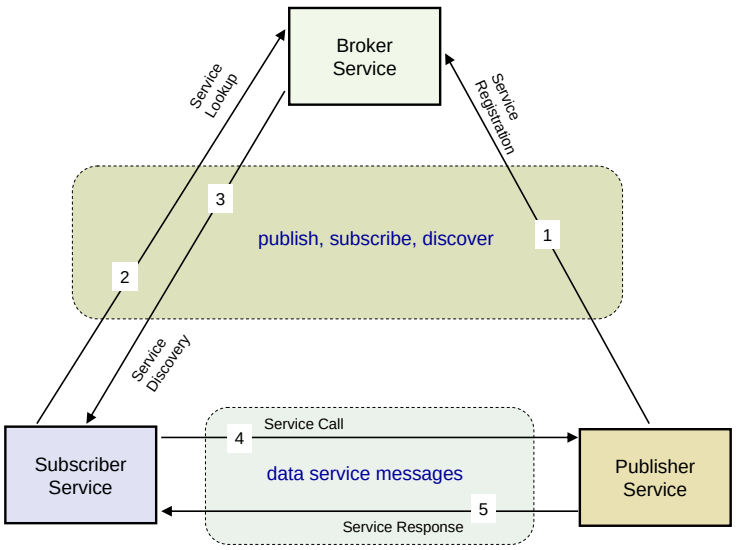


Figure 2.1: SOA-Compliant System Overview

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. As shown in the Figure 2.1, a SOA-Compliant [53] system needs to account for the basic architectural components (publishers, subscribers, and brokers) and their interactions that are fundamental to SOA.

2.2 Hardware/Software Co-Design

Co-design is a set of engineering processes to simultaneously consider hardware and software aspect of a system [37]. The concept of HW/SW Co-design refers to partitioning a system under design in terms of software and hardware components such that each can be developed separately and simultaneously. Thereafter, they can be synthesized. The goal is to enable robust system designs with emphasis on improving hardware and software interaction.

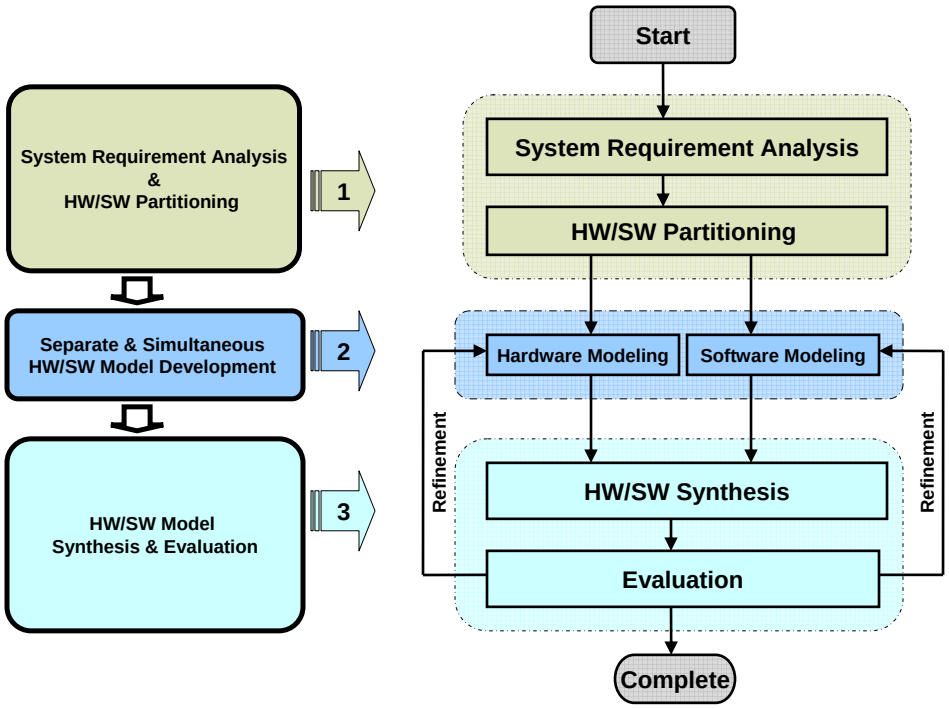


Figure 2.2: Basic concept for HW/SW co-design process

In HW/SW co-design, there are three fundamental steps that must be accounted for:

1. System requirements analysis and functionality partitioning into software and hardware parts
2. Separate and simultaneous development of software and hardware models
3. Syntheses of software and hardware models for evaluation

System requirements are analyzed to establish required system functionality. Hardware/software partitioning determines which of the functions needs to be implemented in hardware and which ones in software. Then functionalities are mapped onto software components and hardware components. Software and hardware models are developed simultaneously based on the mapped functionalities in each. Once the models are developed, the software and hardware models

are synthesized and evaluated through simulation. Based on evaluation, repartition of functionalities and model refinement may be necessary. Refinement uses an integrated hardware/software model to produce an efficient hardware/software combination. Finally, a set of alternate acceptable system designs are proposed based on evaluation that determines the impact of the alternative design on overall system performance. Thus, co-design allows software and hardware components to interact with each other and designer can evaluate system performance at an early design stage. As the software, hardware and their interactions are evaluated at an early stage and iterated for refinement, the final design is more robust and cost effective by avoiding potential mismatches in system configurations. Hence, 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 [49].

The SW/HW co-design concept has been successfully applied in embedded system simulation, design and development (e.g., [22, 55, 71]). In embedded system, designers have the flexibility in partitioning (and mapping) system functionality into software and hardware components under the system level design constraints (e.g., performance and cost). This is due to the highly specialized hardware that can be designed simultaneously as the software functionality is evolving. Thus, software and hardware components can be very specialized particular to the system under design. Depending on mapping of functionalities into software or hardware, different system performance can be achieved. However, the designer has to make a trade-off in system development cost vs system performance. In general, embedded system design emphasizes more on optimizing

hardware resource utilization while improving system performance. Design space exploration is targeted towards improving system performance using highly specialized hardware. The design choice of ensuring extensibility of system functionality with the software stack is secondary and generally results as a trade-off for addressing time critical performance aspects in application specific hardware [71]. Thus, ensuring compatibility and extension of embedded system functionality by leveraging the flexibility of software stack, without a purview of the hardware capability, is limited.

In networked systems, interconnection among heterogeneous hardware components is important. Compatibility to ensure interconnection is achieved by providing standard software stacks and functionalities that hides the heterogeneity of the hardware components. Essentially in networked systems, the software plays an important role in ensuring system interoperability. Hence, HW/SW co-design modeling in networked system design requires developing a set of generalized networked hardware models that allow networked interconnection and support interactions with software models accounting for the behavior of the software components (e.g., DEVS/DOC [49]). To reduce complexity in system modeling by hiding the heterogeneous aspect of hardware components, the hardware models is abstracted to support a generalized view of the hardware as perceived by the software models while providing mechanism to specify hardware resource requirement from the software models. In this context, system partitioning in networked system entails decomposing requirements into system functionalities and constrained in assigning generalized interconnection functionalities into hardware models while the rest are assigned in the software models. The software and hardware interaction is modeled to the represent basic interaction and the software model is based on the software behavior specification as well as software to software interaction specification. The software models thus play a major role in

representing characteristics of the software systems while the hardware models accounts for resource constraints. Hence, the importance of co-design modeling in the networked system is the HW/SW synthesis capability that allows system performance evaluation under alternate system configurations. For example, in DEVS/DOC [49], the object behavior is modeled in the software layer and the processing unit, network components are modeled in the hardware layer. While the synthesized system represents the system behavior of the interacting objects constrained under the hardware resources, the object model and its behavior serves as the driving force of the resultant system dynamics. Hence, both the behavior of software models and hardware models is important in developing networked system. However, due to the non-specialized nature of the hardware, the software models play the distinguished role in characterizing fundamental software system dynamics.

Co-Design in Service Oriented Computing System Modeling

Service Based software system design is largely focused on service modeling and ignore the importance of hardware. Since Service Based Software Systems depend on message interchange and computation resources, the emphasis on the "software only" design approach leaves out a critical component of the system - i.e., the underlying hardware. The co-design concept applied in embedded systems can find its application in a similar yet considerably different domain of Service Based Software Systems. Service Based Software Systems running on networked hardware is similar in concept yet at a different scale and abstraction level. Based on this observation, the emphasis is on the introduction of the concept of co-design in Service Based Software System design. To model and simulate the dynamics of service based components executing on hardware, co-design can be considered as activities to simultaneously (and separately) design hardware and software layers of a Service Based Software System and support their synthesis. In contrast to

the term "co-design" used in the embedded systems literature where emphasis is, for example, at low-level specification of FPGA, the term refers to software service models communicating with the models of the underlying hardware. 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 distinct mappings from the former to the latter.

In developing Service Based Software System - there is a need to represent service, define service to service interactions with hardware topologies (e.g., network and cpu) supporting the service interactions. In addition, the software dynamics that depend on the underlying hardware must be accounted. For example, QoS for Service Based Software System depends on the design and realization of the specification. In addition, QoS attributes like timeliness and throughput depend on the runtime behavior of the system which relies on the available hardware resources. Design decisions needs to take into account scenarios where system may operate in regions where critical resources may not be available as needed to maintain the performance level. Incorporating the hardware models in the SBS simulation provides an important dimension in analyzing the system behavior under stress. Thus, the co-design concept applied in Service Based Software System design can provide a missing link needed for critical system analyses as mentioned.

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

- Specification of SOA compliant services (i.e., as software components) and hardware components separately and establishing a well defined relation to allow synthesis and service mapping onto hardware.

- Specification of software to software interaction and accounting for the impact of hardware resource (e.g., cpu speed, memory size, and network bandwidth) constraints.

- Account 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).

Thus, the co-design can account for the basic SBS components - service as software and computation resources (i.e. cpu , memory and network) as hardware. The components can capture the basic functional and resource capabilities of the system. The service performance can be related to hardware resources by developing an assignment between service and hardware in terms of their interaction and resource requirements. A flexible mapping (i.e., an assignment that specifies which service gets assigned for execution in which hardware) would allow forming alternate system configurations. The system models need to represent the base cases for resource constraint (e.g., cpu time, available memory, and available network bandwidth) on interaction of services assigned to a single hardware (Figure 2.3) as well as the the service interactions through a mesh of interconnected (i.e., networked) hardware (Figure 2.4).

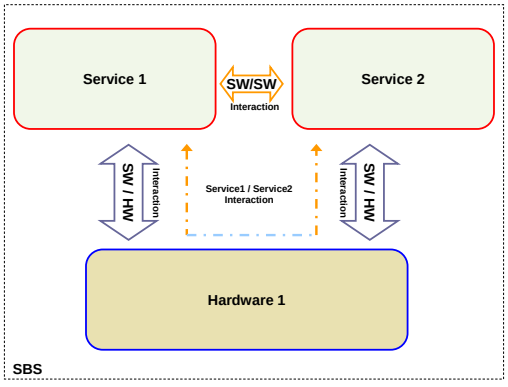


Figure 2.3: Single Hardware Interaction

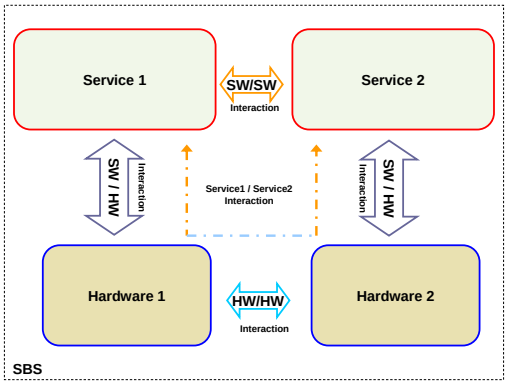


Figure 2.4: Multiple Hardware Interaction

2.3 DEVS Modeling & Simulation

Discrete Event System Specification

Discrete Event System Specification (DEVS) [81] is a modeling formalism that allows modeling dynamic systems with flexible and time-based specification for atomic and coupled model components. Parallel DEVS [16] modeling approach allows modelers to describe discrete event systems. Atomic models capture the dynamics of individual parts of a system. Coupled models can be hierarchically constructed from atomic and other coupled models using well-defined I/O interfaces and couplings. This formalism uses mathematical set theory and provides a framework to support model development with structural and behavioral specifications and abstract simulator protocols for atomic and coupled models. Sequential and various forms of parallelism or distributed methods can be used to simulate atomic and coupled models [81].

The DEVS framework has been extended with object-oriented abstraction, encapsulation, and modularity and hierarchy concepts and constructs [48]. An atomic model specifies input variables and ports, output variables and ports, state variables, internal and external state transitions, confluent, and time advance functions (see Listing-2). This type of model is a stand-alone component capable

of autonomous and reactive behavior with causality and timing concepts. They can also handle multiple inputs and generate multiple outputs. A coupled model description specifies its constituents (atomic and coupled models) and their interactions via ports and couplings (see Listing-3). A coupled model can be composed from a finite number of atomic and other coupled models hierarchically. Due to its inherent component-based support for model composition, this framework lends itself to simple, efficient software environments [57]. Atomic and coupled models have sound causality, concurrency, and timing properties that are supported by various simulation protocols in distributed or stand-alone computational settings.

A DEVS parallel atomic model is defined as $M = \langle X, Y, S, \delta_{ext}, \delta_{int}, \delta_{conf}, \lambda, ta \rangle$ where,

- X, set of input events;
- S, set of sequential states;
- Y, set of output events;
- δ_{ext}, external transition function specifying state transitions;
- δ_{int}, internal transition function specifying state transitions;
- δ_{conf}, confluent transition function specifying handling of simultaneous external and internal transition functions;
- λ, output function generating output events;
- ta, time advance function.

The sequential and parallel views play a central role in modeling and simulation of coupled models since each coupled model is essentially comprised of

multiple atomic models. Two different formalisms exist in this context. The sequential formalism treats components simultaneous transitions sequentially, while the parallel formulation [16] treats them concurrently.

A DEVS coupled model is defined as, $CM = \langle X, Y, D, M_d | d \in D, EIC, EOC, IC \rangle$ where

- X, set of input events;
- Y, set of output events;
- D, set of component names;
- M_d, set of basic components for each $d \in D$;
- EIC, external input coupling;
- EOC, external output coupling,
- IC, internal coupling;

Given (atomic or coupled) components of a coupled model, the couplings among them can be systematically captured using three different types of coupling (internal coupling, external input coupling, and external output coupling). DEVS can ensure semantically identical input/output interfaces for atomic and coupled models. Internal coupling interconnects components of a coupled model. External input coupling interconnects input ports of a coupled model to input ports of its components. Similarly, external output coupling interconnects component output ports of a coupled model to the output ports of the coupled model itself. With coupled models, increasingly more complex models can be constructed using simpler models in a stepwise model development enabling modular model verification and validation and distributed execution.

Dynamic Structure DEVS

Component-based modeling of dynamic structure systems has been well studied [80, 5, 65, 32]. In [32], Dynamic Structure DEVS (DSDEVS) modeling approach has been developed to support structural changes of parallel DEVS models [81]. In [65], a variable structure modeling approach has also been developed using AI 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 [18]. The simulator is developed based on the Dynamic Structure Discrete Event Network System (DSDEN) [5] 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 [32]. 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.

2.4 Modeling and Development Tools

DEVS-Suite Simulator

DEVS-Suite simulator is an integrated modeling and simulation tool that supports SOA-compliant DEVS based software and hardware model development. The suite is developed with MFVC (Model-Faade-View-Controller) architecture

in JAVA. DEVS-Suite provide a scalable framework for visualization of I/O, model specific parameters, simulation system parameters (i.e. phase, sigma, events) while providing capability to model software and hardware of SOA based systems.

The design of the DEVS-Suite simulator separates execution control from the tightly integrated simulator kernel and view.The visualization of models and their animations are supported by module that supports 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 simplifying 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.

The timeview is a module developed for run-time display of data sets as two dimensional plots (every plot has x and y coordinates). Its operation is similar to an oscilloscope. It can display sets of (x, y) values where x (or y) values are plotted with respect to y (or x). In order to use it for plotting time-based simulation data, the x-coordinate for all plots is defined to represent time. 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 plotting time-based simulation data, the x-coordinate for all plots is defined to represent time. 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.

The simulator also supports saving simulation data traces (e.g. tracking data) as Comma Separated Value (CSV) encoded file. Such capability aids in data analysis using independent analysis tools (e.g MATLAB).

.NET Development Framework

The .NET Framework [19] is a software library to support common runtime environment for software developed in multiple programming languages (i.e., C++, C#, Visual Basic). The primary objective of the framework is to allow interoperability among programs developed in C++, C# and Visual Basic. In .NET, programs developed in any of the languages (C++, C#, VB) is compiled to a common intermediate language (CIL) which is a machine and platform independent instruction set. The CIL is then converted by the Common Language Runtime (CLR) to native machine code. Common Language Runtime is a virtual machine that provides supports security, memory management, and exception handling. The use of CIL and CLR allows programs to be portable across machines and platforms. The .NET Framework provides a set of libraries for user interface, low level system data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications thus aiding in rapid application development.

The .NET Framework is also suitable for rapid system prototype development particularly for experimentation testbeds. Experimentation requiring support for data collection from different layers of the system (i.e., device driver, kernel, application) is supported by the Windows Performance Object (WPO) [72] related .NET Application Programming Interfaces. Windows performance objects provides access to system component (e.g., CPU, Memory, Network Card)

performance data. Applications can access performance data using WPO related API set to collect system status. Such data collection capability are important for experimentation testbed intended for system performance evaluation.

Network Packet Monitoring Tool

Network Monitoring Tool provides packet level tracing capability useful for network system performance analysis. Microsoft Network Monitoring 3.4 (i.e., NetMon 3.4) [43] is such a packet monitoring and logging tool provided by Microsoft. It is designed to be used in Windows operating system. NetMon can trace MAC layer data and upper layers protocol packets. The packet traces contain the exact network packets and MAC frames as collected during the observation period. The detailed information contains the time stamps of send/receive events for the network packets. For example, TCP/IP source, destination addresses and ports, packet size, packet fragment numbers, packet type, mac addresses, higher level protocols (i.e., HTTP, FTP, SOAP). Time stamps are logged in local machine's hardware clock and time differences of events can be logged at the 10e-6 of a second of granularity.The NetMon 3.4 allows the top level protocol (TCP/UDP) packets can be tracked up to the MAC level frames. Such capability is important in understanding the packet processing internals. For example, application level data throughput to network layer packet throughput can be correlated based on such data. Based on such data, it can aid in understanding cause and effect relationships of application layer activity and network layer QoS status.

NetMon 3.4 provides an API set that can be used in Microsoft C# /C++ .NET towards automated data tracing and detailed protocol analysis. Traced data can also be saved in repository (as plain text format) that can be used for analysis for 3rd party software (e.g. MATLAB). In network system design and development such tools can play an important part in collecting detailed network related system data and statistics that critical for performance evaluation.

Chapter 3

RELATED WORK

In this chapter, related research work on Service Based Software System (SBS) modeling methodologies, techniques particularly in the context of Hardware/Software Co-design and architectural design is explored. Emphasis is given on i) representation of concepts like SOA and co-design ii) the level of model abstractions in relating system resource with system QoS. For example, while logical workflow level service abstractions and analysis is suitable for service verification [64], the architectural design evaluation with focus on system QoS and system performance requires a holistic approach (i.e. services, resources and their relations) in representing the Service Based Software Systems. It means models have to account for basic cause and effect of fluctuations in system resources on service execution and hence the impact on system QoS. Thus, related work on SBS modeling and simulation is analyzed in terms of capability to represent Service Based Software Systems. Also the section on Network System Modeling Approaches presents the research works of particular interest in relation to the approach presented in the remainder of this dissertation. In this section, Service-Based Software System modeling and simulation for architectural design evaluation is approached from a co-design modeling perspective and a classification of Networked Systems modeling and simulation approaches is outlined in this context.

3.1 Overview

Simulation platform built to take advantage of SOA and SOC is not addressed in this work, rather the research focus of the dissertation is on modeling Service Based Software Systems. Thus, related work is addressed from SOA and SOC system modeling perspective.

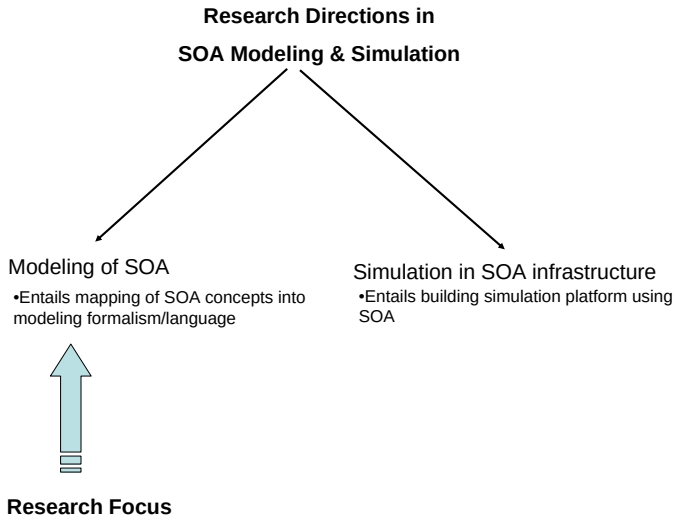


Figure 3.1: Research directions in SOA Modeling & Simulation

Existing SBS modeling and simulation approaches emphasize on process specification and workflow aspect of services [9, 62, 73, 74]. Business Process Execution Language (BPEL) [9] and Process Specification and Modeling Language (PSML-S) [62] consider process flow to represent service functionality that treats QoS primarily in the context of the software components, with no (or limited) consideration for underlying hardware. Similarly, a workflow compatibility analysis in webservice composition using Petri-nets is explored in [74]. Service interaction and the syntactic dependency is modeled as a C-net (composition net) and the compatibility of services is devised as a deadlock structure problem. The service interaction model is focused on using formal basis for structural analysis for checking webservice complatibility in business processes. However, QoS of webservices and dynamic effects of system resources on webservice interactions (e.g. service delay, service throughput) are not addressed. In [73], the proposed algorithm for dynamic service selection uses petri-net based representation of workflow and service dependency. They acknowledge the effect of resource constraint on QoS, however, the service QoS dependency on system resources (i.e. cause and effect)

are not accounted for. In [55], a model-based system co-design is explored and simulation-based design is suggested. The development process includes virtual system (i.e. combined SW/HW) prototype design and development from system specification & requirement analysis. The use of simulation for design evaluation is applied as an integrated part of design evaluation. This research is concerned with embedded systems design (and not with enterprise systems which preceded Service-Based Software systems).

The importance and a variety of software architecture design techniques for enterprise systems are presented in [6]. The described concepts and methods are concerned with functional, runtime, and non-time system quality attributes. The key role of design with respect to quality attributes (such as performance and thus dependency on hardware network) is addressed. The concrete impact of hardware resources is described in terms software design. Furthermore, designs are not targeted to support simulation, instead they are developed for implementing actual software systems. In OMNeT++ [44], OPNET [45], and ns-2 [42] detailed network protocol level simulation is supported. However, the software service layers in these tools do not account for SOA concepts and compliancy. Select approaches [53, 7, 28] and tools consider hardware [44, 45]. In [7], a hybrid approach is considered where OMNeT++ [44] is used to simulate network resources while service functionality is simulated as a process chain model (Proc/B). OMNET++ offers detailed network models (i.e., INET framework with IPv4/v6,TCP/UDP/Ethernet models etc.) and the service execution ordering and sequence is specified using Proc/B. However, the modeling approach does not provide direct representation of SOA artifacts rather treats services as processes. Also, time dependent QoS (e.g., service delay) is only accounted for in the network delay without any representation of other system resources (e.g., CPU and memory) which can have a significant effect on QoS under resource constraint

scenarios [77, 76]. As such, accounting detail network resources in OMNET++ is insufficient to independently observe complex service interactions that dynamically impact system resources and hence the system QoS [26]. In addition, it is desirable to have a formal basis for mapping Proc/B Functional Units to the hardware components in OMNET++. This is important in order to directly account for time synchronization between the software and hardware components.

In another approach [2], a model driven approach to predict performance of webservices is addressed. BPEL process is described with a UML model and annotated with performance data and transformed into a Layered Queuing Network (a.k.a LQN) [70] model. The LQN model representation is then used to predict the performance of the system. Such an approach can be used to study general software and hardware systems, however, the LQN representation does not account for the SOA concepts. In addition, as both software and hardware are abstracted as a parameterized Layered Queueing Network, the approach lack the concept of co-design and thus the advantages of a systematic separation and synthesis of HW/SW is missing. In [76], Activity-State-QoS (ASQ) models are developed using data modeling techniques by analysis of experimental data. System activity, state and QoS are selected in factorial design experiments. Then cause and effect relationships are established by building ASQ models. However, ASQ models are static in nature and thus once a model is derived it is limited in representing cause and effect relations beyond the particular system configuration it is derived. In addition, such models are not suitable for Service Based Software System design (i.e., module structure and relation). In [53], service models are developed using SOA concepts and principles. Although the role of hardware and its interaction with hardware is noted, only a simplified abstraction of a network router is used. Also, the work acknowledge the roles of SW/HW co-design and outlines the importance of detailed hardware models. In CloudSim

[12], a software framework is developed towards modeling and simulation of the Clouds [68]. The software abstraction (i.e., cloudlet) in the framework models cloud applications and the hardware abstraction (i.e. host) models the physical machines. Emphasis is given on modeling abstractions for cloud infrastructures (physical host, virtual machines) along with support for simulation of resource management (CPU,Memory,Bandwidth allocations, job scheduling, VM Management etc.). Separation of software and hardware from a Co-design modeling perspective is absent in addition to the simulation approach framework (i.e. simjava library [56]) lacking a formal theoretical basis. In PrimoGENI [67], a mix of emulated host and simulated network capability using a real-time simulator is applied toward realistic network simulation. The virtual machines with unmodified application code emulates hosts in a network while the network router and topology is simulated with packet level simulation. Simulated networks can interact with emulated hosts in real-time packet exchanges. Primarily the limitation of scaling the virtual machine based experimentation capability is addressed by simulating the network such that larger networks can be simulated without the need for larger scale experimentation infrastructure. The focus of the research is not modeling and simulation of SBS rather on accurate and realistic network experimentation. Experiments with SBS can be conducted, however the service realization has to be at the level of real software modules and as such too detailed to be suitable for early architectural design evaluations. In addition, generating alternative service hosting scenarios can be a challenging given the VM based approach. In DEVS/NS-2 [35] DEVS and NS-2 models are combined akin to how Proc/B and OMNET++ are integrated. DEVS models can be developed to model high level application behavior while pre-built NS-2 models can provide detailed network layer behavior. This common approach does not have a strong theoretical basis for model interactions and time synchronization. It is also important to

note that the above tools and their underlying approaches are not based on the co-design concept (i.e., systematic SW/HW separation and synthesis) as part of their modeling methodology.

3.2 Networked System Modeling Approaches

In this section, the modeling and simulation approaches of particular interest to this dissertation is highlighted. From the perspective of Co-design, networked system modeling & simulation approaches can be broadly categorized into three types, namely

1. **Integration based ad-hoc approaches**: Such approaches do not consider a systematic separation & synthesis of software and hardware models rather use ad-hoc integration techniques to support interactions among components. Models are generally developed in different modeling approaches and later integrated (i.e. using software integration approach) to support interactions. Such approaches lack systematic and generalized methods and procedures towards model integration. Example, DEVS/NS2, Proc/B.

2. **Co-Design based approaches**: This kind of approaches provide a systematic separation & synthesis of software and hardware models. The approaches are developed using a common modeling methodology and language that provides systematic separation of software & hardware models and a SW/HW mapping scheme to simulate the complete system. Example, DEVS/DOC, OMNET++, OPNET.

3. **Non Co-Design approaches** : This category contains the approaches that can model software or hardware models but may not consider systematic synthesis and simulation of models of two types. Generally such approaches take software only view of the system (or hardware only view as well) and do not emphasize on the interactions between SW/HW models. Example, PSML-S in

Table 3.1: Overview of related work

Models Service Concept	**Co-Design**	**Non Co-Design**	**Ad-Hoc**
YES	N/A	SOAD, DSOAD	Proc/B
NO	DEVS/DOC, OMNET++ OPNET	Cloudsim	DEVS/NS2

DDSOS, SOA-DEVS, Dynamic SOA-DEVS.

Based on the above categorization, a brief explanation of specific modeling & simulation approaches (see Table 3.1) is provided in the following subsections.

Integration Based Ad-hoc Approaches

DEVS/NS2

DEVS/NS-2 [35] is an integrated network simulation tool which combines modeling capability of DEVS and NS-2. It provides high level software modeling capability using DEVS and low level detailed network modeling capabilities using NS-2. DEVS serves as a base framework to support a well-defined formalism specification for structure and behavior of systems. NS-2 discrete event network simulator is used to build and run various detailed network models and protocols. The DEVS/NS-2 approach allows inter-operable simulation of the two modeling and simulation systems and helps in model development effort in terms of increased high and low level networks modeling capability and enhanced re-usability. The hierarchical modeling support of DEVS formalism can simplify complex & inherently hierarchical systems and the detail protocol level modeling support of NS-2 can provide detailed low level system representation resulting in a environment that can support complex networked system simulation. However, model synthesis between NS-2 and DEVS are ad-hoc in nature lacking any formal basis. nature based on the DEVS/NS-2 interface.

Proc/B

ProC/B [7]is a process chain-based modeling approach for modeling and performance evaluation of logistics networks. Proc/B provides hierarchical modeling capability for systems requiring process flow modeling with support for structural as well as behavioral aspects. ProC/B accounts for the specifics of the application area by capturing the structure in form of Functional Units (FUs) and the behavior by process chains (PCs). In ProC/B, FU's might offer services, which can be used by activities of process chains. Each service is again described by a process chain. Proc/B and OMNET++ integration has been used in [7] to model service behavior and utilizes detail network level modeling support of OMNET++. The approach is an ad-hoc integration of two modeling layers and allows process flow model of service interactions that do not support modeling the impacts of service interactions on system resources and how it effects the system QoS.

Co-Design Based Approaches

DEVS/DOC

Distributed Object Computing (DOC) [20] is a computing paradigm based on interacting objects distributed in interconnected networked computing nodes. DOC focuses on providing a structured high level conceptual model on the inherently complex distributed software systems.

DOC is based on fusing the concept of Object Oriented Programming (and Design) (OOP) towards developing distributed computing systems. The OOP techniques for developing systems can reduce complexity through creation of reusable software frameworks using well understood software architectures and design patterns. Thus, the primary challenge in developing DOC systems is the design of the interacting software modules on a cost effective hardware platform while ensuring the scalability and reducing complexity of the system. In this

context, DOC utilizes capability in Object Oriented techniques in distributing software modules flexibly onto multiple heterogeneous networked computing resources.

DEVS/DOC [49, 29, 28] is a simulator for Distributed Object Computing (DOC) systems. 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[11] is proposed as 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 the 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, net-

working 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 high-level model abstractions to be developed without precisely modeling low-level of details (e.g., detailed transport protocol modeling). The concept is primarily used in the DCO layer by introducing probability of method invocations for software objects. The 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 at early stage during system architecture design. It is important to note that the DCO in DEVS/DOC supports basic concurrent execution models (i.e., none, method, and object) for software objects. 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 operating systems with (e.g., UNIX) and without (e.g., DOS) concurrency support behave differently. Such models of concurrent execution allow the modeler to support scenarios with different granularity of concurrency. However, support for concurrency using multi-threaded execution is prevalent in recent computing platforms and any standard operating system (e.g., Windows XP/2000, Linux, and UNIX) supports multi-threading.

OMNET++ and OPNET

OMNeT++ [44] is a public-source simulation environment that has been developed for the modeling of communication protocols and has been extensively used in this area. OMNeT++ has been used for the analysis of networked systems. The tool environment includes a graphical front end and several other tools that support the modeling and simulative analysis of complex networked systems. The simulation kernel is written in C++ and offers several classes to support the specification of complex models. Due to the implementation in C++, the resulting simulation models are efficient.

OPNET [45] is a closed source simulation environment that has been developed for commercial purposes to support complex network system modeling and simulation. The environment provides a graphical model development front end and a simulation kernel developed in C++. OPNET provides prebuilt models of communications devices, protocols, technologies, and architectures, and can simulate their performance in a dynamic network environment. The environment allows code debugging and data analysis features to facilitate the design process [45]. OPNET provides software (i.e. application) model that simulate the application behavior on a real-world network. By changing the configuration, link capacity, traffic volumes, and characteristics of a network model a network systems performance can be measured. This capability aids in studying various wired and wireless routing protocols, understanding complex network architectures and designs.

Non Co-design Approaches

SOA-DEVS

SOA-compliant DEVS (SOAD) [53] 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 [81]. 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 SOA-compliant DEVS models, the DEVS-Suite simulator is extended to specify SOA abstractions [18, 36]. The simulator supports basic SOA elements including services, service registry, service discovery, and messages.

In compliance with SOA specification, the relation between subscribers and publishers in SOAD is established through the broker. The SOAD model assumes predefined 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 low level of details which complicates model validation. The SOAD models communicate with messages that represent service description, look up, and service messages [53]. 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.

Dynamic SOA-DEVS

Dynamic SOA-DEVS (DSOAD) [40] is developed to support simulation of dynamic aspects of Service-based Software Systems. It supports runtime addition or deletion of the SOA models (publishers, subscribers, broker) as well as their connections.The approach introduces dynamic structure DEVS modeling [5] into the SOAD modeling [53]. The dynamic structure capability is appropriate for allowing SOA-compliant DEVS models to change their structures during simulation. The resulting Dynamic SOAD (DSOAD) has a Broker-Executive model with a basic set of basic rules for supporting subscribers and publishers to be dynamically added or removed with proper interactions (couplings) supported between publishers and subscribers. As a result dynamic service creation and structural changes require adhering to SOA basic principles at runtime such that general composition of services - this is assured in DSOAD.

DSOAD has been developed as an extension to SOAD with the addition of Dynamic Structure DEVS, which at its core has an executive model component with rules for adding (and removing) services and specifying how the services 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 SOA include rules to relate the services. For example, SOA allows relation between broker and subscribers. So if a subscriber is dynamically instantiated in DSOAD the executive needs to ensure that it has couplings to the broker. In addition, the direction of message flow among broker, publisher and subscriber is supported in SOA-compliancy. DSOAD component interactions through messages are developed based on template interactions based on the di-

rection of message flow as couplings are configured at runtime in DSOAD. The executive contains the knowledge of the direction of the message flow (and thus input vs 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 broker. The DSDEVS executive model by itself does not account for SOA and in particular does not account for the concept of the broker. So a combined broker-executive model is provided to support runtime model coupling and ensuring structural SOA compliancy of the modified system.

CloudSim

CloudSim [12] is a modeling and simulation framework towards simulation of cloud systems. The framework provides the basic model abstractions for concepts and artifacts for representing cloud systems. The model abstractions in CloudSim includes the concept of Virtual Machine(VM), hosts (Physical Machines) and Data Centers, DataCenterBroker and cloud applications as cloudlets assigned to run in VMs. The simulation engine is based on SimJava - an event driven simulation kernel. The models interact with the simulation engine through events. FIFO queue of global event list is maintained by the engine and each model gets notification of event based on the FIFO event order. When model's generate further events, they are queued in the global event queue in non decreasing event time order.

From modeling perspective, CloudSim model abstractions are more atuned to represent cloud based systems from resource allocation, resource assignment and resource scheduling aspect. Software abstraction at the level of cloudlet do not represent SOA concepts (e.g.,publisher, subscriber,broker). The request, response, publish, subscribe etc. behavior of services is not represented in the cloudlet abstraction rather it is at the level of representing a single context of execution that

requires virtual resources to get executed. The concept of service and basic service properties (e.g., publishable, stateless, discoverable) is not accounted for by the cloudlet abstraction. It is more suitable for abstracting the resource requirement for cloud software execution. In addition, it is difficult to model application specific behavior with cloudlet abstraction. There is a set of hardware abstractions (hosts, virtual machines) that accounts for system resources (e.g., cpu cycles, network bandwidth, available memory). However, from a modeling perspective, the systematic separation of software and hardware and their synthesis is not present (i.e., concept of co-design is also not used). In essence, the simulation framework is suitable for evaluating resource scheduling algorithms and policies (as opposed to architectural design evaluation) in cloud systems.

DDSOS and PSML

The DDSOS [60] framework based on HLA Runtime Infrastructure (RTI) [30] supports PSML [63] based modeling and simulation of service based systems. The framework is intended for software engineering based on model driven development approach. PSML is a process specification and modeling language. It is based on behavior, structure and constraint aspect of a software system. The core model defines the data types and operations in other models, the structure model defines the constructs to specify the static structure of the system under modeling, the behavior model defines the constructs for describing the dynamic behaviors and the constraint model defines constructs for specifying the non-functional constraints in the structure and behavior models. PSML-S [59] and PSML-C [61] are PSML derivations to support services and service collaboration modeling.

Using a model of the system, it is possible to evaluate the system architecture and simulate system behaviors in the early phases of software engineering life cycle. As PSML is intended towards system modeling from software engineering perspective, its primary capability is focused on software aspects of systems. Con-

ceptual representation of timed element is limited (specified as time constraints) in PSML. Modeling software systems with representation of system hardware components is not well supported.

Chapter 4

MODELING METHODOLOGY

In this chapter, the basic concepts of Service Oriented Computing-DEVS (SOC-DEVS) is discussed. Co-design modeling is contextualized from SBS modeling and simulation perspective. In addition, DEVS formal specifications for the core SOC-DEVS models are outlined. Finally, a design of SOC-DEVS including the core models are explained.

4.1 Co-Design Concept

Service-Based Software System design approaches do not emphasize on the functionalities of hardware or otherwise make strong simplification about the role of hardware [7, 62, 53]. However, system QoS and performance can depend on service execution on hardware components and system resource status. Hence, approaching the simulation based design fo SBS from a software only perspective leaves out a critical aspect of the system - the system resources represented by the hardware components. Since Service-Based Software Systems depend on message based interactions and computation resources,the lack of hardware representation in simulation can limit the observable dynamics of the system. In this context, the co-design concept can be applied in design of Service-Based Software Systems.

Based on this observation, Service Oriented Computing DEVS (SOC-DEVS), a co-design modeling methodology is developed. The emphasis is on the introduction of the concept of co-design in Service Based Software System design (see Figure 4.1).

To model and simulate the dynamics of service based components executing on hardware, the co-design modeling allows to simultaneously model hardware and software layers of a Service Based Software System and support simulation after their synthesis. As noted earlier, HW/SW co-design in this dissertation refers

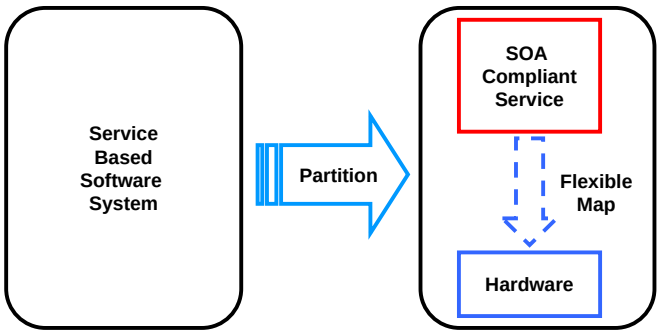


Figure 4.1: Basic concept for SBS HW/SW co-design

to a modeling and simulation of a set of software service models executing on a set of distributed networked hardware models. Characterization of co-design for Service Based Software System includes modeling and simulation for high-level specification of software and hardware layers as well as separate mappings of the former to the latter.

In short, SOC-DEVS modeling methodology integrates the concept of co-design, concept of SOA and DEVS formal modeling specifications. As such, SOC-DEVS allows the following towards simulation-based design of SBS -

1. Specification of SOA-compliant service as software component and system resources as hardware components separately and then establishing a well-defined relation to allow their synthesis.

2. Specification of software service to software service interaction while accounting for hardware resources (e.g., CPU speed, memory capacity, network bandwidth). Hardware resource constrains can thus be part of system configuration and resultant system dynamics can be observed.

3. 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).

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. Networked services executing on distributed hardware components support two types of high level system configuration - 1. resource constraints (CPU time, available memory) for interaction of services are restricted to a single hardware. 2. resource constraints (CPU time, available memory, network bandwidth) for interaction of services are allowed for networked hardware components. Both types of interaction are supported in SOC-DEVS.

4.2 Design and Model Specification in DEVS

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 (see Figure 4.2).

The software service (*swService*) in the software layer is modeled to capture basic service interaction semantics (e.g., message exchanges and service invocations). The *swService* accounts for atomic service concept. The hardware layer is modeled to represent computing node (e.g., CPU speed, memory capacity)

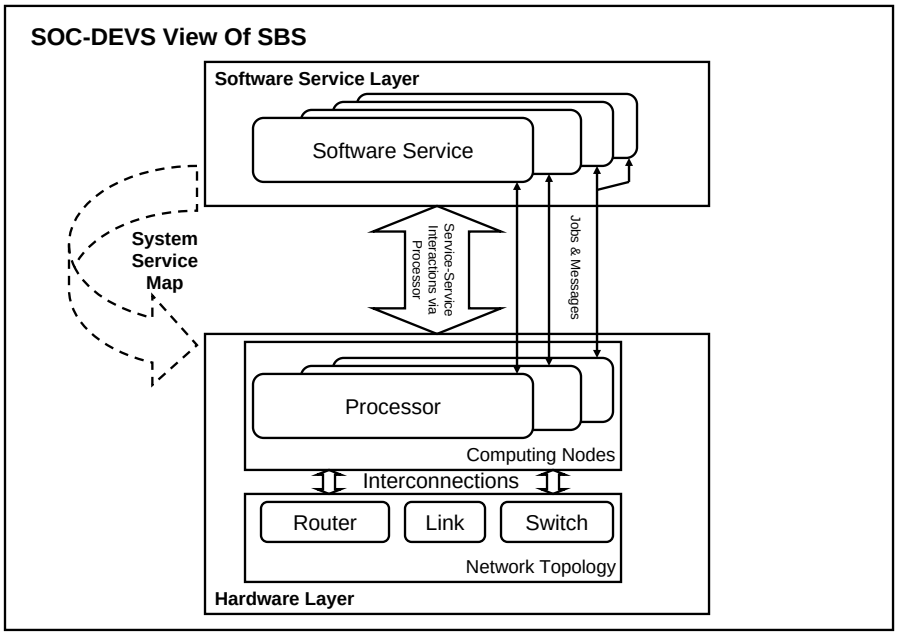


Figure 4.2: A generic conceptual view of the software,hardware model parts and mapping for Service-Based Software Systems

and network system resources (e.g., network bandwidth) important for composite service execution. Software service interactions via hardware layer is developed using the concept of jobs and messages (see Figure 4.3). The job captures the concept of CPU resources (cpu cycles, memory) consumed for software service execution while the message represents the software service to software service communication.

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.

Software Service and Hardware Interaction

When invoked, software services execute operations [1] as part of its modeled functionalities. Such operations utilize cpu cycles and memory for execution. In this context, the concept of computation load is the number of cpu cycles and amount of memory needed for an operation execution. Computation load consists of two parts 1. CPU load 2. Memory load. 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 Mbytes) used while the operation is being executed. The concept of communication load is the amount of data (e.g., 1500 bytes) that a software service sends as part of the executed operation semantics.

The concept of computation and communication load captures two important aspects of SBS. In a SBS, execution delay (e.g., computation delay at service host) associated with computation and the network delay (e.g., queuing delay, processing delay etc. at routers, switches) associated with message communication, both can impact the timing of service execution. For computation intensive systems, computation delay can be significant contributor to degradation os QoS. Whereas, for communication intensive systems, the network delay and available network bandwidth can significantly influence overall QoS in a heavily loaded network. Even if local systems has enough resource for service execution, the overhead in communication can impact the time dependent behavior of software services. As SBS system is message based, the use of communication load allows software services to impact the network dynamics important in the distributed software services interactions and executions. The communication load enables loading the network by consuming available network bandwidth and impacting timing of service interactions effected by network delays. Thus, a combination

[1] Refer to Software Layer in Section 4.2

of computation load and communication load capture the fundamental hardware dependency of distributed software services.

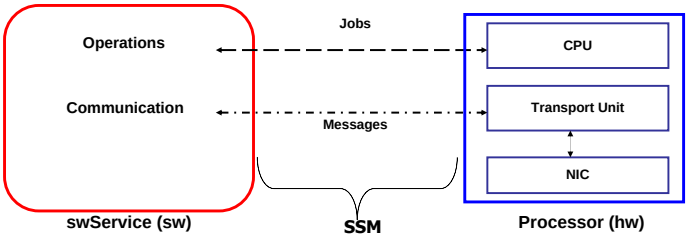


Figure 4.3: Basic software, hardware models and their interaction in SOC-DEVS

To represent the concept of computation and communication load, two SW/HW interaction entities 1.Job 2. Message are defined in SOC-DEVS, respectively. A job is defined as the required cpu cycles and memory consumption of an operation to complete execution in the CPU. Jobs are associated with operations and denote the computation load (i.e.,CPU load and memory load) of an operation. A message is defined as the unit of information that services can exchange as part of the message based service interactions and denote the communication load of an operation. Messages denote the payload data and the type information (e.g., request, response). As the definition implies, a job is parameterized with required cpu cycles & required memory size as in *Job<RequiredCpuCyles, RequiredMemory>*. Similarly, a message is parameterized with the payload size and type as in *Message<Type, PayloadSize>*. For example, if a software service operation requires 1200 cpu cycles and consumes 2048 bytes of system resource then an execution of the operation requires a *job<1200 cycles, 2048 bytes>* to

be completed in the CPU. Similarly, if a service operation needs to send 1MB of data, the *message<Generic, 1MB>* is created and sent to the recipient service.

Software Service 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 the hardware layer in purview. Based on the concept for service model in SOAD [53], the fundamental service elements of SOA (i.e., Broker, Publisher and Subscriber) are modeled in the software service layer.

Software Service

The primary abstraction in the software layer is the "software service" and it accounts for the basic 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 among software services. From the perspective of a single service, 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. In addition, communication band-

width can also effect the time and priority of service execution. So, the messages are parameterized with communication load. As a result, the software service is capable of modeling basic time dependent behavior of service execution with dependence on system resource status. Generated jobs and messages from the service model includes and carry this information to the hardware layer [2]

A service context maintains the state of the invoked operation and the message that requested the operation. Each service invocation is associated with a corresponding unique service context. The software service can support multiple operations and the requested operation is specified in the incoming message. An invocation of an operation sends a job parameterized with the CPU and the memory load to the hardware layer. In addition, 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. This way, a software service is capable of handling multiple invocation requests and the simultaneous operation invocations. Any job initiated from a software service is associated with a service context and operations completes when the final jobs associated with service contexts is complete. Once an operation is completed, the associated service context is removed. If operation completion semantic requires then a message is also sent to the requesting software service before the service context removal.

The software service is specified as an atomic model in Parallel DEVS [16, 81] named *swService* (see Listing-4.1, 4.2, 4.3). It is the basis for developing any service models in SOC-DEVS and as such is the template for modeling service functionality and I/O behavior with hardware components (i.e., Processor, Link, and Switch described in Section 4.2). The input and output for this model are defined as X={*(inMsgs* $\times$ *iMessages),(inJobs* $\times$ *iJobs)*} and Y={*(outMsgs* $\times$

[2]Refer to SW/HW interaction in section 4.2

oMessages),(outJobs $\times$ *oJobs)*$\}$ (see Figure 4.5. It maintains an outgoing message queue (FIFO), an outgoing job queue (FIFO) and a list of active service contexts. The model has two standard state variables $S = \{phase \times \sigma\}$ where phase can have values of *"passive"* or *"sending"* and $\sigma \in \Re^{+}_{[0,\infty]}$. Models are initialized to *phase* = "*passive*" and $\sigma = \infty$. The terms *Messages*, *Jobs* denote to a set of messages and a set of jobs, respectively. Service context denotes the context (e.g., the invocation message and operation requested) of the service when it is invoked.

Listing 4.1: swService Formal Specification - Input, Output, States

$$\text{swService} = \langle X, Y, S, \delta_{ext}, \delta_{int}, \delta_{conf}, ta, \lambda \rangle$$

$$\begin{aligned}
X &= \{(p, v)\} \text{ where } p \in IPorts,\ v \in X_P \text{ are the input port names and values} \\
IPorts &= \{\text{"}inMsgs\text{"}, \text{"}inJobs\text{"}\} \\
X_P &= \{iMessages, iJobs\} \text{ where } iMessages = \text{set of messages,} \\
&\quad iJobs = \text{set of jobs} \\
Y &= \{(p, v)\} \text{ where } p \in OPorts,\ v \in Y_P \text{ are the output port names and values} \\
OPorts &= \{\text{"}outMsgs\text{"}, \text{"}outJobs\text{"}\} \\
Y_P &= \{oMessages, oJobs\} \text{ where } oMessages = \text{set of messages,} \\
&\quad oJobs = \text{set of jobs} \\
S &= phase \times \sigma \times sendMsgQ \times sendJobQ \times scList \text{ where} \\
&\quad phase = \{\text{"}passive\text{"}, \text{"}sending\text{"}\},\ \text{and } \sigma = \Re^{+}_{[0,\infty]} \\
&\quad sendMsgQ = \text{queue of outgoing messages,} \\
&\quad sendJobQ = \text{queue of outgoing jobs,} \\
&\quad \text{and } scList = \text{list of acive service contexts}
\end{aligned}$$

When a *swService* receives a message through its *inMsgs* port, the δ_{ext} function is invoked to process the message. Since multiple *swServices* can be associated with the same port, the *swService* receiving the message checks whether the message is for itself. This is done by the *isMsgForSelf(message)* function and if so the *decode(message)* function is used to decode the incoming message to figure out the requested operation (Figure 4.8). A new service context is created and a job associated with the operation is queued in the *sendJobQ* queue. After processing all the incoming messages, the *swService* stays in *phase="sending"* for $\sigma = \epsilon$, a small non-zero time period. Then, the λ function generates output(s) (i.e., jobs) for "outJobs" port using *sendJob(job)* function and the model immediately goes into *phase="passive"* for $\sigma = +\infty$ due to the execution of the δ_{int} function. Similarly, when a *swService* receives a completed job through *inJobs* port, δ_{ext} function is invoked. The incoming job is then checked by *isMsgForSelf(job)* function to verify that the *swService* is the intended recipient. If *isMsgForSelf(job)* returns true then the *doOperation(job)* function is executed, otherwise *swService* ignores the job. If the operation semantic requires then a message is enqueued in the *sendMsgQ* queue.

Listing 4.2: swService Formal Specification - State Transitions

$$\text{swService} = \langle X, Y, S, \delta_{ext}, \delta_{int}, \delta_{conf}, ta, \lambda \rangle$$

$$\begin{aligned}
\delta_{ext}(s, e, x_p) \quad = \quad & (\text{“}sending\text{”}, \epsilon, sendMsgQ', sendJobQ', scList'), \\
& \text{if } isMsgForSelf(x_p) = true \\
& \text{or } isJobForSelf(x_p) = true, \text{ and } phase = \text{“}passive\text{”} \\
& \text{or “}sending\text{”}, \text{ where } x_p \in X_P, \epsilon = 0^+, \\
& sendMsgQ' = sendMsgQ \cup message,
\end{aligned}$$

$$
\begin{aligned}
& \text{if } inJobs \text{ has received a } job \in iJobs \\
& \text{such that } doOperation(job) = true, \\
& sendJobQ' = sendJobQ \cup job, \\
& \text{if } inMsgs \text{ has received a } message \in iMessages, \\
& \text{such that } decodeMsg(message) = true, \text{and} \\
& scList' = scList \cup conext, \\
& \text{if } inMsgs \text{ has received a } message \in iMessages \\
& \text{such that } existsInSCList(message) = false, \text{ or} \\
& scList' = scList - conext, \\
& \text{if } inJobs \text{ has received a } job \in iJob \\
& \text{such that } isComplete(job) = true \\
= \quad & (phase, \sigma - e, sendMsgQ, sendJobQ, scList); \\
& \text{otherwise (i.e., ignore } x_p) \\
\delta_{int}(s) \quad = \quad & (\text{“}passive\text{”}, \infty, \phi, \phi, scList)
\end{aligned}
$$

After processing all the incoming jobs, the *swService* stays in *phase=* *“sending”* for $\sigma = \epsilon$ period. Then, the λ function generates output(s) (i.e., messages) for the “outMsgs” port using *sendMsg(message)* function. Afterwards *swService* goes into *phase=“passive”* for $\sigma = +\infty$ due to the execution of the δ_{int} function. The associated service context for the completed job is also removed. Simultaneous I/O of messages & jobs (using $\sigma = 0$) are also allowed using the same semantics as two separate output queues for messages and jobs are maintained. In addition, the required time for job completion depends on the system load and system resource fluctuations in the hardware and the service operation completion time can vary accordingly.

Listing 4.3: swService Formal Specification - Time Advance, Readout Function

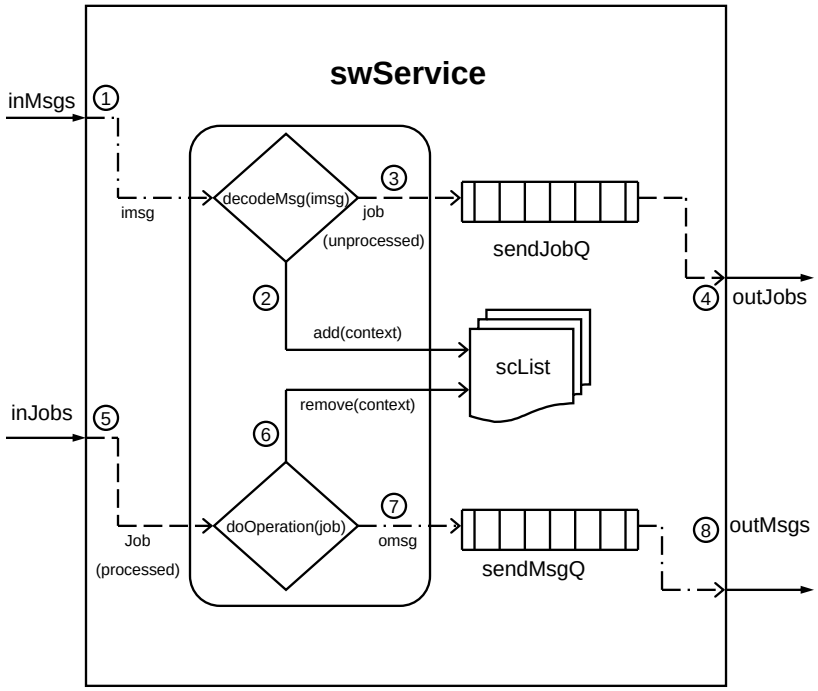
$$\text{swService} = \langle X, Y, S, \delta_{ext}, \delta_{int}, \delta_{conf}, ta, \lambda \rangle$$

$$\begin{aligned} \lambda(s) &= (outMsgs, oMessages) \text{ when } sendMsgQ \neq \phi \text{ or} \\ &\quad (outJobs, oJobs) \text{ when } sendJobQ \neq \phi \\ ta(s) &= \sigma, \text{ where } 0 \leq \sigma \leq \Re^{+}_{[0,\infty]}. \end{aligned}$$

The basic *message* and *job* processing scheme in the *swService* model is shown in Figure 4.4b. It illustrates a sequential ordering of a *swService* receiving a *request* message, using it to create an *unprocessed* job for *swService* assigned processor, and upon the receipt of the *processed* job creating and sending a *response* message. When messages and jobs are received simultaneously, they are processed in an arbitrary order which is facilitated by the two independent $sendJobQ$ and $sendMsgQ$ FIFO queues. There is no loss of generality since the ordering of messages and jobs has no side-effect. This is because some period of time must elapse – i.e., the time the assigned processor requires to process the job is nonzero (see Figure 4.5). Furthermore, when multiple *request* messages are received, no ordering is assumed among them. However, should ordering be necessary, it can be handled within δ_{ext} function and in coordination with the δ_{int} function. Statechart for swService is not provided for brevity; it can be derived from its DEVS specification [58].

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* (see Figure 4.10) in the context of the message interaction each model supports and the resultant message exchanges.



(a) Message & job processing in *swService*.

1. *imsg* (e.g., request) is received at the $inMsgs$ port.
2. service *context* is created and added to the *scList*.
3. *job* (e.g., computation load) associated with the service *context* is added to $outJobQ$.
4. output function sends *job* through $outJobs$ port.
5. completed *job* is received at the $inJobs$ port.
6. service *context* associated with the completed *job* is removed from *scList*.
7. *omsg* (e.g., response) is generated and sent to the $sendMsgQ$.
8. output function sends *omsg* through $outMsgs$ port.

(b) Basic processing steps for message and job inputs by *swService*.

Figure 4.4: *swService* internals.

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. Service invocation is initiated by a message exchange between a publisher and a subscriber. The service invocation message invokes a service operation associated with an 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.

The *swService* execution semantics are extended in Broker, Publisher & Subscriber by differentiating *response* message and the fundamental service operations required in each. For example, in any of the SOA components, if a message is received through the *inMsgs* port the δ_{ext} function is invoked, *decodeMsg(message)* is called, service context and jobs are sent and completed jobs result in removal of service contexts (i.e., same steps are involved as in a *swService* (see Section 4.2) semantics). However, *response* messages and the type of operation that is supported are different in Broker, Publisher and Subscriber. In Broker, a service information message will result in a service registration *registerService* invocation whereas a service lookup message will result in a service discovery *registryLookup* invocation and the returning of the discovery result message to the sender. In Publisher, a service call message would result in the requested service invocation and returning the service result to the requester. In Subscriber, a service result message will invoke service consumption *consumeService* and so on. Since, all messages use the same message I/O ports per *swService*, the logic of service interaction are incorporated in the *decodeMsg(message)* and *doOperation(job)* functions. The message is extended to contain the relevant informations as required to define such behaviors. As in *swService*, the timing behavior is dependent on

the system resources with the use of CPU, memory, communication loads and the resultant models capture the behavior as well as execution semantics of a generic Broker, Publisher and Subscriber.

It is important to note that 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 requests 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.

Broker

The Broker model responds to message of type *ServiceInfoMessage* and *ServiceLookupMessage*. When such messages are received, the *decodeMsg(message)* function evaluates the requested operation and sends a job associated with the operation to *sendJobQ*. When the completed job has returned, the *doOperation(job)* function completes which is either service registration or service lookup. In case of a service lookup operation a *ServiceInfoMessage* is sent to the *sendMsgQ*. Hence, the input $X_{inMsg} = \{ServiceLookupMessage,\ ServiceInfoMessage\}$, output $Y_{outMsg} = \{ServiceInfoMessage\}$ and the logic of *decodeMsg(msg)*, *doOperation(job)* define the Broker model.

$$\text{Broker} = \langle X, Y, S, \delta_{ext}, \delta_{int}, \delta_{conf}, ta, \lambda \rangle$$

$$\begin{aligned} X &= \{(p, v)\} \text{ where } p \in IPorts,\ v \in X_P \text{ are the input port names} \\ &\quad \text{and values} \\ IPorts &= \{\text{“}inMsgs\text{”}, \text{“}inJobs\text{”}\} \\ X_P &= \{iMessages, iJobs\} where \\ &\quad iMessages = ServiceLookupMessage,\ ServiceInfoMessage, \end{aligned}$$

$iJobs$ = set of jobs

$Y = \{(p, v)\}$ where $p \in OPorts$, $v \in Y_P$ are the output port names and values

$OPorts = \{\text{“}outMsgs\text{”}, \text{“}outJobs\text{”}\}$

$Y_P = \{oMessages, oJobs\}$

where $oMessages = ServiceInfoMessage$,

$oJobs$ = set of jobs

$S = phase \times \sigma \times sendMsgQ \times sendJobQ \times scList$ where

$phase = \{\text{“}passive\text{”}, \text{“}sending\text{”}\}$, and $\sigma = \Re^{+}_{[0,\infty]}$

$sendMsgQ$ = queue of outgoing messages, and

$sendJobQ$ = queue of outgoing jobs,

and $scList$ = list of acive service contexts

$\delta_{ext}(s, e, x_p) = (\text{“}sending\text{”}, \epsilon, sendMsgQ', sendJobQ', scList')$,

if $isMsgForSelf(x_p) = true$

or $isJobForSelf(x_p) = true$, and $phase = \text{“}passive\text{”}$

or “$sending$”, where $x_p \in X_P, \epsilon = 0^{+}$,

$sendMsgQ' = sendMsgQ \cup message$,

if $inJobs$ has received a $job \in iJobs$

such that $doOperation(job) = true$,

$sendJobQ' = sendJobQ \cup job$,

if $inMsgs$ has received a $message \in iMessages$,

such that $decodeMsg(message) = true$, and

$scList' = scList \cup conext$,

if $inMsgs$ has received a $message \in iMessages$

such that $existsInSCList(message) = false$, or

$scList' = scList - conext$,

if $inJobs$ has received a $job \in iJob$

such that $isComplete(job) = true$

$= (phase, \sigma - e, sendMsgQ, sendJobQ, scList)$;

otherwise (i.e., ignore x_p)

$\delta_{int}(s) = (\text{“}passive\text{”}, \infty, \phi, \phi, scList)$

$\lambda(s) = (outMsgs, oMessages)$ when $sendMsgQ \neq \phi$ or

$(outJobs, oJobs)$ when $sendJobQ \neq \phi$

$ta(s) = \sigma$, where $0 \leq \sigma \leq \Re^{+}_{[0,\infty]}$.

Publisher

Unlike the Broker model, the Publisher model responds to incoming message of type *ServiceCallMessage*. The *decodeMsg(message)* function evaluates the requested operation and sends a job associated with the operation to *sendJobQ*. When the completed job has returned, the *doOperation(job)* function completes which is either service publication or perform service operation. In case of perform service operation a *ServiceCallMessage* is sent to the *sendMsgQ*. The specifications for the input $X_{inMsg} = \{ServiceCallMessage\}$, output Y_{outMsg}= $\{ServiceCallMessage, ServiceInfoMessage\}$, and logic for *decodeMsg(message)* and *doOperation(job)* are defined differently from those given for the swService model.

$$\text{Publisher} = \langle X, Y, S, \delta_{ext}, \delta_{int}, \delta_{conf}, ta, \lambda \rangle$$

$X = \{(p, v)\}$ where $p \in IPorts$, $v \in X_P$ are the input port names and values

$$
\begin{aligned}
IPorts &= \{\text{“}inMsgs\text{”}, \text{“}inJobs\text{”}\} \\
X_P &= \{iMessages, iJobs\} \\
&\quad \text{where } iMessages = ServiceCallMessage, \\
&\quad \text{and } iJobs = \text{set of jobs} \\
Y &= \{(p, v)\} \\
&\quad \text{where } p \in OPorts,\ v \in Y_P \text{ are the output port names} \\
&\quad \text{and values} \\
OPorts &= \{\text{“}outMsgs\text{”}, \text{“}outJobs\text{”}\} \\
Y_P &= \{oMessages, oJobs\} \\
&\quad \text{where } oMessages = ServiceInfoMessage, \\
&\quad ServiceCallMessage\ oJobs = \text{set of jobs} \\
S &= phase \times \sigma \times sendMsgQ \times sendJobQ \times scList \text{ where} \\
&\quad phase = \{\text{“}passive\text{”}, \text{“}sending\text{”}\}, \text{ and } \sigma = \Re^{+}_{[0,\infty]} \\
&\quad sendMsgQ = \text{queue of outgoing messages,} \\
&\quad sendJobQ = \text{queue of outgoing jobs,} \\
&\quad \text{and } scList = \text{list of acive service contexts} \\
\delta_{ext}(s, e, x_p) &= (\text{“}sending\text{”}, \epsilon, sendMsgQ', sendJobQ', scList'), \\
&\quad \text{if } isMsgForSelf(x_p) = true \\
&\quad \text{or } isJobForSelf(x_p) = true, \\
&\quad \text{and } phase = \text{“}passive\text{”} \text{ or “}sending\text{”}, \text{ where} \\
&\quad x_p \in X_P, \epsilon = 0^{+}, \\
&\quad sendMsgQ' = sendMsgQ \cup message, \\
&\quad \text{if } inJobs \text{ has received a } job \in iJobs \\
&\quad \text{such that } doOperation(job) = true,
\end{aligned}
$$

$sendJobQ' = sendJobQ \cup job$,

if $inMsgs$ has received a $message \in iMessages$,

such that $decodeMsg(message) = true$, and

$scList' = scList \cup conext$,

if $inMsgs$ has received a $message \in iMessages$

such that $existsInSCList(message) = false$, or

$scList' = scList - conext$,

if $inJobs$ has received a $job \in iJob$

such that $isComplete(job) = true$

$= (phase, \sigma - e, sendMsgQ, sendJobQ, scList)$;

otherwise (i.e., ignore x_p)

$\delta_{int}(s) = (\text{“}passive\text{”}, \infty, \phi, \phi, scList)$

$\lambda(s) = (outMsgs, oMessages)$ when $sendMsgQ \neq \phi$ or

$(outJobs, oJobs)$ when $sendJobQ \neq \phi$

$ta(s) = \sigma$, where $0 \leq \sigma \leq \Re^{+}_{[0,\infty]}$.

Subscriber

The Subscrber model responds to incoming message of type *ServiceInfoMessage* and *ServiceCallMessage*. The *decodeMsg(message)* function evaluates the requested operation and sends a job associated with the operation to *sendJobQ*. When the completed job has returned, the *doOperation(job)* completes which is either lookup service or request service or consume service. In the first two cases a *ServiceLookupMessage* or a *ServiceCallMessage* is sent to *sendMsgQ*, respectively. In consume service operation, no message is sent. Here, the input $X_{inMsg} = \{ServiceCallMessage, ServiceInfoMessage\}$, output $Y_{outMsg} =$

{*ServiceLookupMessage*, *ServiceCallMessage*}, and the logic of *decodeMsg(message)*, *doOperation(job)* differs from other SOA component models.

$$\text{Subscriber} = \langle X, Y, S, \delta_{ext}, \delta_{int}, \delta_{conf}, ta, \lambda \rangle$$

$$\begin{aligned}
X &= \{(p, v)\} \text{ where } p \in IPorts,\ v \in X_P \text{ are the input port names and values} \\
IPorts &= \{\text{“}inMsgs\text{”}, \text{“}inJobs\text{”}\} \\
X_P &= \{iMessages, iJobs\}, \\
&\quad \text{where } iMessages = ServiceCallMessage, \\
&\quad ServiceInfoMessage\ ,\ iJobs = \text{set of jobs} \\
Y &= \{(p, v)\}, \\
&\quad \text{where } p \in OPorts,\ v \in Y_P \text{ are the output port names and values} \\
OPorts &= \{\text{“}outMsgs\text{”}, \text{“}outJobs\text{”}\} \\
Y_P &= \{oMessages, oJobs\}, \\
&\quad \text{where } oMessages = ServiceCallMessage, \\
&\quad ServiceLookupMessage\ oJobs = \text{set of jobs} \\
S &= phase \times \sigma \times sendMsgQ \times sendJobQ \times scList \text{ where} \\
&\quad phase = \{\text{“}passive\text{”}, \text{“}sending\text{”}\},\ \text{and } \sigma = \Re^{+}_{[0,\infty]} \\
&\quad sendMsgQ = \text{queue of outgoing messages}, \\
&\quad sendJobQ = \text{queue of outgoing jobs}, \\
&\quad \text{and } scList = \text{list of acive service contexts} \\
\delta_{ext}(s, e, x_p) &= (\text{“}sending\text{”}, \epsilon, sendMsgQ', sendJobQ', scList'), \\
&\quad \text{if } isMsgForSelf(x_p) = true
\end{aligned}$$

or $isJobForSelf(x_p) = true,$ and $phase =$ "$passive$"

or "$sending$", where

$x_p \in X_P, \epsilon = 0^+,$

$sendMsgQ' = sendMsgQ \cup message,$

if $inJobs$ has received a $job \in iJobs$

such that $doOperation(job) = true,$

$sendJobQ' = sendJobQ \cup job,$

if $inMsgs$ has received a $message \in iMessages,$

such that $decodeMsg(message) = true,$ and

$scList' = scList \cup conext,$

if $inMsgs$ has received a $message \in iMessages$

such that $existsInSCList(message) = false,$ or

$scList' = scList - conext,$

if $inJobs$ has received a $job \in iJob$

such that $isComplete(job) = true$

$= (phase, \sigma - e, sendMsgQ, sendJobQ, scList);$

otherwise (i.e., ignore x_p)

$\delta_{int}(s) = (\text{"}passive\text{"}, \infty, \phi, \phi, scList)$

$\lambda(s) = (outMsgs, oMessages),$ when $sendMsgQ \neq \phi$ or

$(outJobs, oJobs)$ when $sendJobQ \neq \phi$

$ta(s) = \sigma,$ where $0 \leq \sigma \leq \Re^+_{[0,\infty]}$

Hardware Layer

The hardware layer consists of abstraction of *single* hardware as well as *networked* hardware (refer to the discussion on *swService* behavior in Section 4.2). 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.

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 [28]. Each of the models is developed as DEVS atomic model.

The CPU is specified with one input port, one output port, and two parameters (i.e., CPU speed and memory size) [29]. The "processor.inJobs" input port accepts requests from software service to execute jobs. The completed 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 restricts 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.

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.

Network Switch

Network switch is developed as DEVS atomic model. It is used to route packets among multiple 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 [82], the modeler needs to specify and initialize static address lookup tables in network switches to represent a network topology under consideration.

Network Router

Network router is developed as DEVS atomic model. It is used to route packets among multiple processors. The network router 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. A neighbor connectivity based distributed route discovery algorithm is supported in the network router. The mapping is done using routing algorithm that automatically updates routing table. Based on the network connectivity and route discovery message propagation, the whole network route is discovered at system model initialization. The algorithm however does not support route rediscovery once the route has converged at initialization.

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.outMsgs" port of each software service component to the "processor.inMsgs" port of the processor. Similar couplings are made for "swService.outJobs" to "processor.inJobs". It also couples the "processor.outMsgs" port of the processor to the "swService.inMsgs" port for each software service component. Similar couplings are made, as well, for "processor.outJobs" to "swService.inJobs" (see Figure 4.5). 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 4.8). The behavior and performance of a SBS is dependent on both the software service and the hardware layers. Dynamic behavior is specified in the software service by computational load, communication load and service interactions. By mapping the software services onto processors,

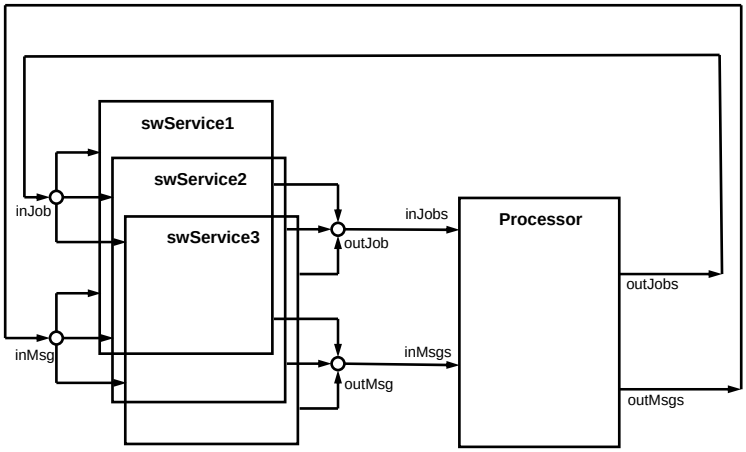


Figure 4.5: *swService* to processor assignment in SSM with I/O couplings.

the dynamics of the software service may be adversely affected given the capabilities and topology of the hardware layer. As the software services compete for processor resources, their dynamics are driven by the performance of the processor which in turn determines the performance of the software service to software service interaction which also drive the dynamics of the network performance and hence the QoS of the SBS.

Single Hardware Configuration

In single hardware configuration (see Figure 4.6), there is a single processor in the system and services get mapped to this single processor. As there is only one processor in the configured system, no network connectivity is possible and the services to service interaction is only via this single processor. Messages from services do not pass through the network card either. This is because, the TransportUnit alone suffices in ensuring communication among all services.

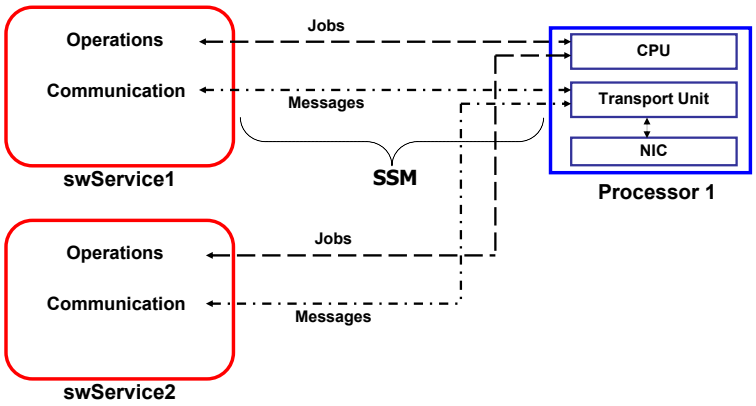


Figure 4.6: Service interactions in single hardware configuration

Networked Hardware Configuration

In networked hardware configuration (see Figure 4.7), services get mapped to multiple processors interconnected by network switches or links. In this configuration, distributed processors are interconnected by links, switches and routers. The services mapped on the processors can communicate via the network. Based on the processor interconnection, various network topology can be configured. The network connectivity and link bandwidth is an important factor in the service to service interactions in such configurations.

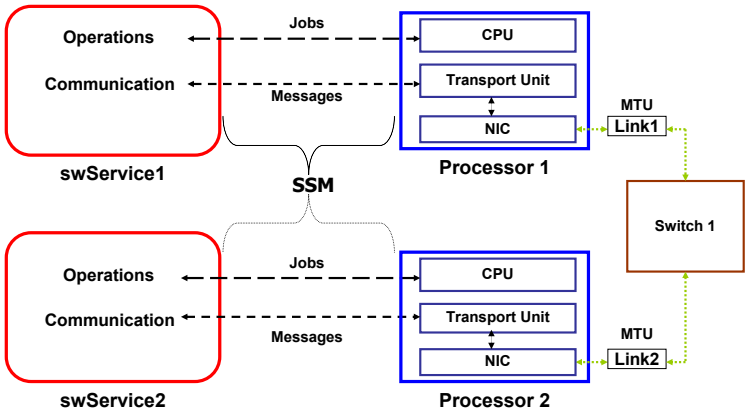


Figure 4.7: Service interactions in Networked Hardware

Software service layer consists of *swServices*. The UML design (attributes and methods) of the *swService* (see Figure 4.9) provides the basis for developing service simulation models in SOC-DEVS. The IHardwareLayer provides a generic interaction interface for the *swService* with the hardware layer. Also, hardware components conform to IHardwareLayer interface (CPU, TransportUnit implements IHardwareLayer). The use of interface concept ensures modularity for the software service layer; offering a level of independence from the internal details of hardware implementation (and vice-versa). However, the logical behavior of the *swService* and hardware models is specified in DEVS and the simulation models interact via DEVS messages. In this context, the System Service Map ensures the coupling of the software service layer and hardware layer models as illustrated in Figure 4.5 while the model implementation conforms to IHardwareLayer.

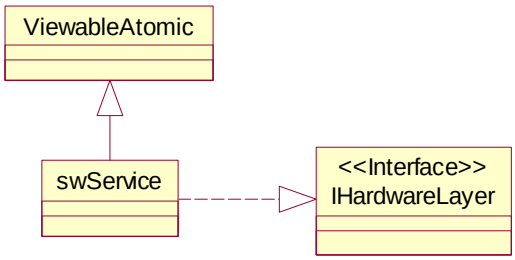


Figure 4.8: Design specification for *swService* simulation model components in DEVS-Suite simulator

The *swService* provides a generic design template to develop the SOA elementary components (i.e., Broker, Publisher, and Subscriber). Each of the Broker, Publisher, Subscriber models is developed by extending the *swService*

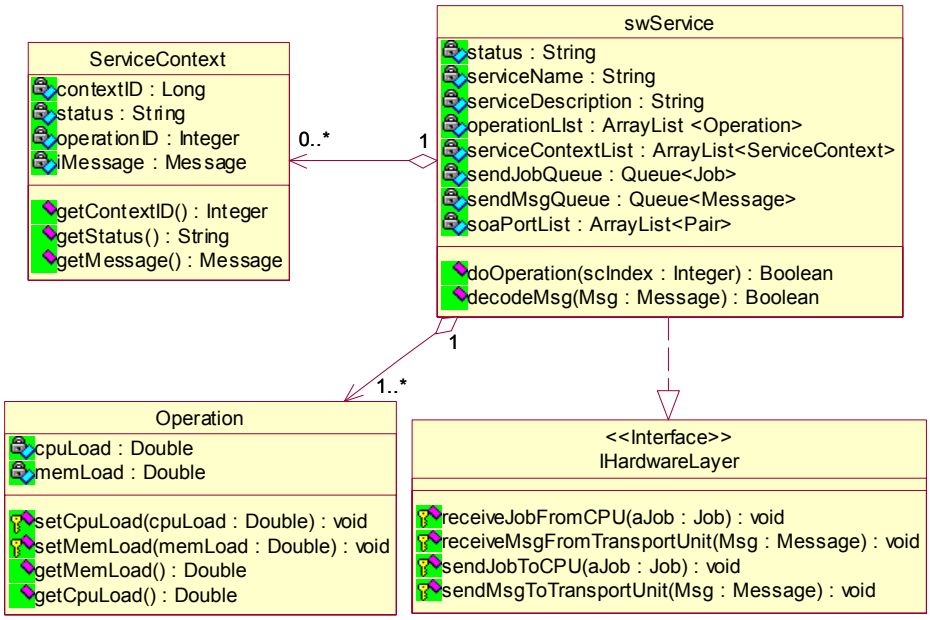


Figure 4.9: Design specification for *swService* model

(see Figure 4.10). As such, each conforms to the IHardwareLayer interface and extends the properties of *swService*. The behavior of the Broker, Publisher, and Subscriber model differs in the context of the supported message interactions and the type of request/response message exchanges (see the DEVS specifications in Section 4.2). The design of user defined SOA-Compliant service models can be developed based on the provided fundamental software service layer models by extending the base classes and implementing their DEVS specifications (i.e., Broker, Publisher, and Subscriber DEVS specifications).

As shown in the UML class diagram in Figure 4.9, *swService* contains a list of operations it supports. Each supported operation is associated with cpu load and memory load specifying the required cpu cycles and memory to complete the operation in the processor. The list of service context provides handle

to active service invocation context for the *swService* instance. This allows the *swService* model to account for multiple invocation handling capability. *swService* also maintains an outgoing Message and an outgoing job queue that hold the generated jobs and messages by the active service contexts associated with the *swService*. Messages generated can specify the recipient service's endpoint address (provided by the invocation message in the active context stored in the active context list). The *swService* can specify its own endpoint addresses in the port list and are associated with the operations. The IHardwareLayer methods are realized by the *swService* that accounts for DEVS message based interactions with the processor model. Thus, the *swService* accounts for the basic atomic service properties and with the concept of servce system mapping, provides a basis for service to service interaction modeling via the hardware layer.

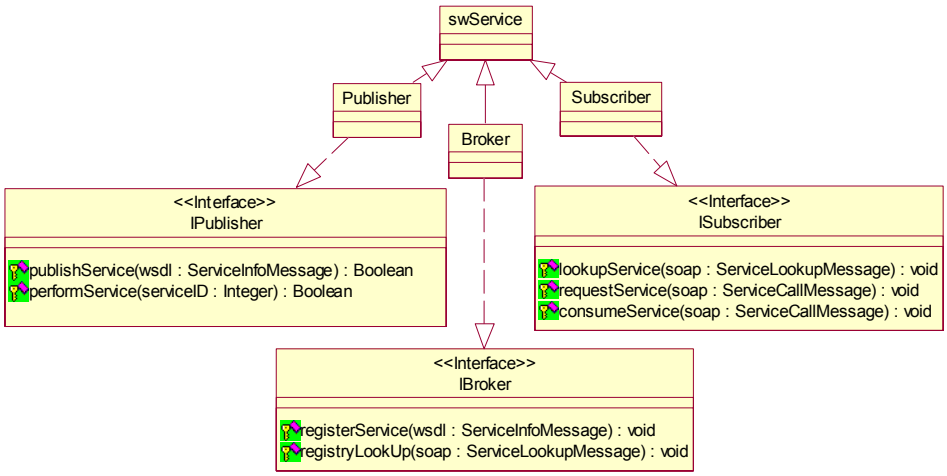


Figure 4.10: Design specifications for Broker, Publisher, and Subscriber

State Chart

swService abstracts basic service behavior and it is implemented as a state machine. As part of its state specification, some state transitions are dependent on interactions with CPU and TransportUnit model (in Processor) that are also state machines. In such cases, composite state chart can show the details of state transitions of the component models and simplify the understanding of complex system interactions.

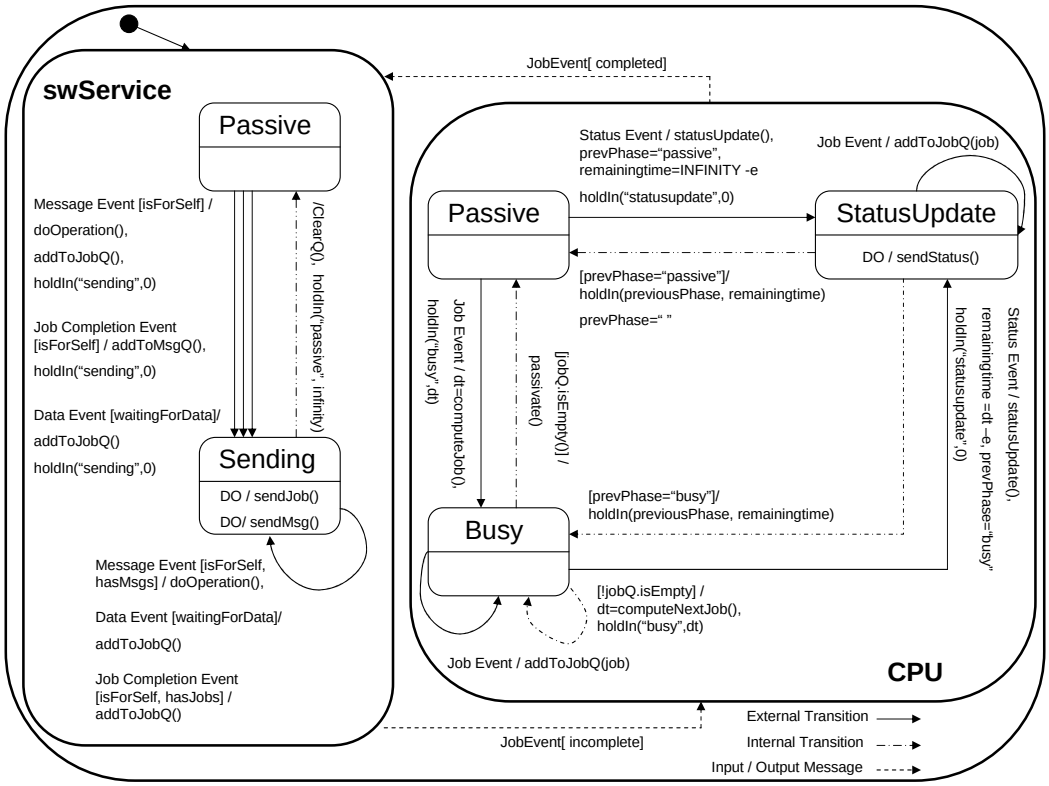


Figure 4.11: Composite state chart for swService and Processor's CPU

The composite state chart for *swService* and Processor components CPU and TransportUnit are illustrated in the Figures 4.11 and 4.12, respectively. The *swService* state machine is designed to respond to two external stimuli, messages and jobs. Similarly, the CPU stimulus is jobs and TransportUnit stimulus is messages. The logical behavior of service is modeled while making state transition

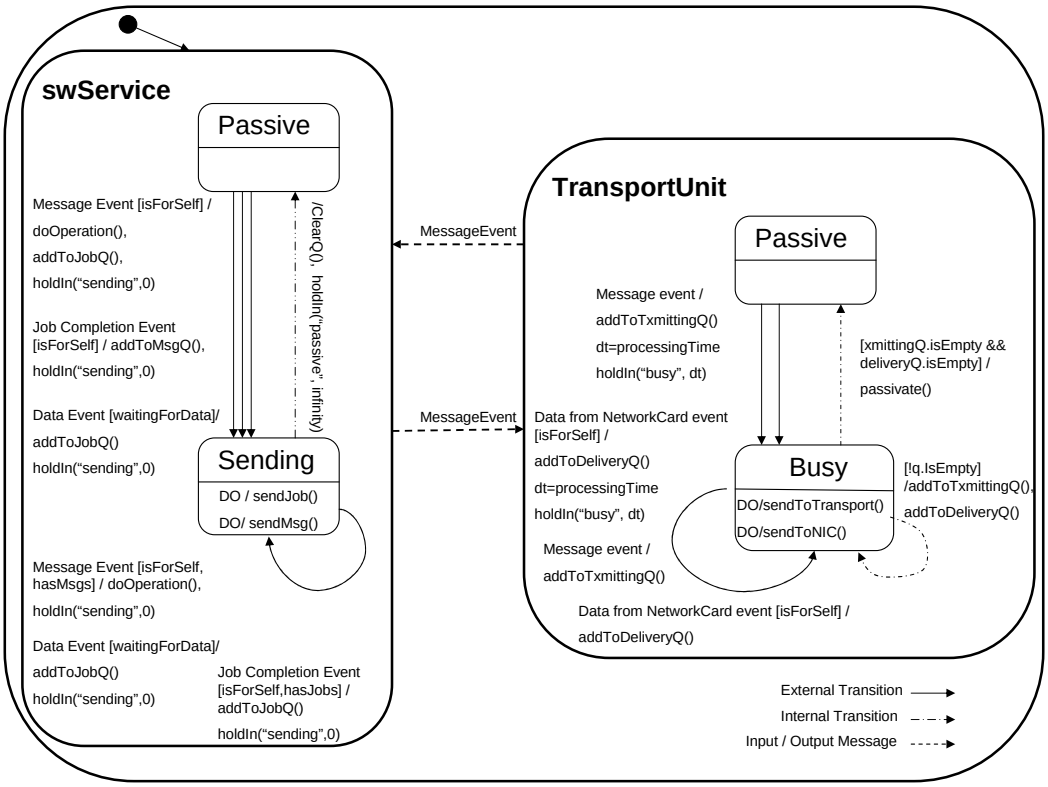


Figure 4.12: Composite state chart for swService and Processor's TransportUnit

dependency on CPU and TransportUnit state transitions. Thus, the request/response message based interactions among *swService* is agnostic of the underlying dependency.

When a message is received by a *swService*, the recipient *swService* state machine is initiated (see Figure 4.12). Each message reception results in a new service context creation and the *swService* stores the context of the invocation. Then a job is created and sent to the Processor to be executed in the CPU. The CPU receives the job and starts processing the job or enqueues it in a FIFO queue (if CPU is in busy phase, otherwise job is started to be processed immediately). The job takes a finite amount of logical time to be completed and during this logical time span the service execution for the associated service context do not progress. The time span between sending a job to the Processor and receiving

the processed job is nonzero and contributes to the service execution delay associated with the service context. Once the job is completed, it is sent back to the *swService*. The reception of a completed job associated with a service context is interpreted by the *swService* as completion of service operation and generates a response message to the invocation message. The *swService* model can also be defined by the modeler not to send any response messages. The response message is sent to the Processor in the TransportUnit. Once the TransportUnit receives the message, it processes it immediately or enqueues it if it is busy processing other messages. Once the message is processed, it is sent to the recipient *swService*. If multiple simultaneous messages or jobs are received, they are queued and processed in the same simulation cycle.

Any transition in the *swService* model has zero time advance. It means that the *swService* processes messages and jobs in zero logical time from a simulation timing perspective. However, jobs generated by the *swService* takes finite (and nonzero) time in the processor model and until the processed job returns to the *swService* the particular service context is waiting. Thus, the *swService* timing is dependent on the processing time at an external model (i.e., processor model) Modeling *swService* timing this way accounts for the effect of hardware processing time as the contributing factor to the *swServices* execution time. This is in contrast to a common approach where a model (in this case *swService*) would have explicitly specified the timing. Another important aspect of parallel DEVS is that it supports concurrent model execution from simulation timing perspective. For example, if *swService* and TransportUnit both have concurrently received (i.e., event times are same) messages then simulator will execute the models in a random order. However the simulation clock won't advance until all logical concurrent events are handled by each model. Thus, from logical timing perspective, both the models have executed at the same time.

4.4 Realization in DEVS-Suite Simulator

The SOC-DEVS models are developed in the DEVS-Suite simulation tool [18, 36], an integrated modeling and simulation tool that supports SOA compliant DEVS based software and hardware model development. SOC system models based on service models, hardware models and their syntheses can be simulated in DEVS-Suite (see Figure 4.13).

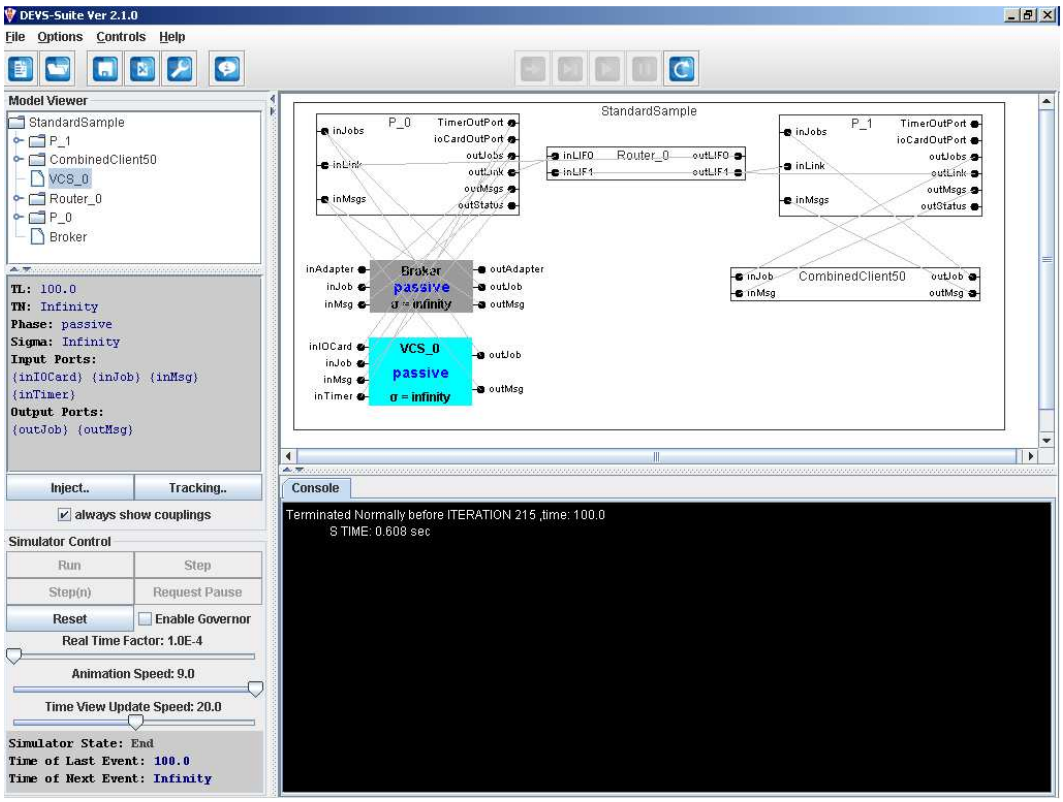


Figure 4.13: DEVS-Suite Simulator

DEVS-Suite provides "model.modeling" package which contains the classes that serves as the basis of modeling. The modeling module provides basic modeling elements including atomic and coupled models that can send and receive messages via input and output ports. The atomic class provides state assignment and functions to process external and internal events. All atomic models can run as a separate component with the ViewableAtomic class and all coupled models

can be executed as a separate system with ViewableDigraph class. The core models in SOC-DEVS are extensions of ViewableAtomic class. The system models are extensions of ViewableDigraph that contains all the system components models.

In the Figure 4.14, the SOC-DEVS package structure is shown. The Core package contains the service layer and hardware layer models. The ServiceModelInterfaces package defines the basic interfaces that must be implemented by the core service components defined in the ServiceModels package (i.e.seService, Publisher, Subscriber, Broker). The hardware layer models defined in the Hardwaremod package (i.e. Processor, CPU, Link, Router, Switch) must also implement the basic interfaces defined in the HardwareLayerInterface. The syntheses support of service and hardware models are provided the SSM class in the SystemServiceMap package. The SSM class allows service assignment to processors. The necessary service to processor couplings are also implemented in the SSM class.

The exemplar Voice Communication System (VCS) is in VCSystemModel package. The VCSPublisher class and the VCSSubscriber class extends the core publisher and subscriber class, respectively. The package structure is flexible for future extension to domain specific models for developing exemplar SOC system models.

4.5 Capabilities and Limitations of SOC-DEVS

The SOC-DEVS provides a basis for systematically building simulation models of SOC systems. From a modeling perspective, the SOC-DEVS provides modeling constructs of the software (i.e., Broker, Publisher and Subscriber) and hardware (i.e. router,switch, processor, transport unit, nic, link) building blocks of SOC systems. The methodology applies separation of service and hardware models with capability for their syntheses. The synthesis of service models with hardware models supported by the Service System Mapping (SSM) concept allows model-

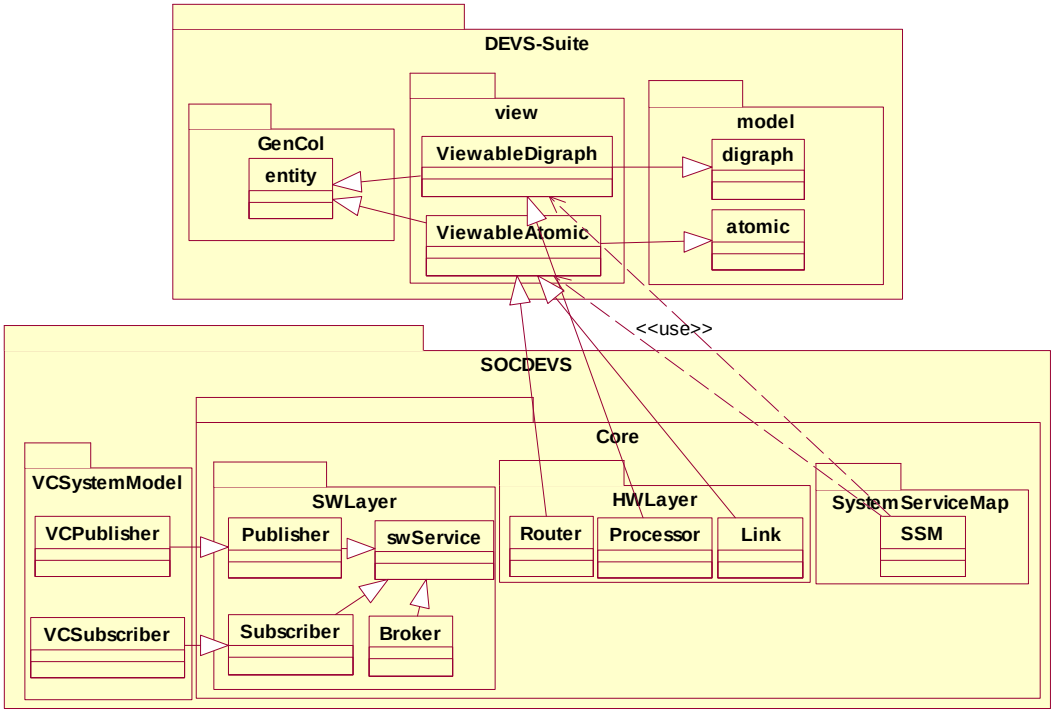


Figure 4.14: SOCDEVS package structure

ing services arbitrarily distributed over a network. The generic service models capture fundamental behavior of services as well as service to service interactions via the networked hardware. The generality of the methodology allows the modeler a structured approach to SOC system modeling. The model specification in SOC-DEVS is targeted for representing generic service models capturing computation and communication aspect. Domain specific models can be built on the foundation of the generic models. The realization of SOC-DEVS in DEVS-Suite provides a domain specific exemplar model of a Voice Communication System. The development of an exemplar model from generic service models and its ability to represent SOC system dynamics is considered the basis for demonstration of the usefulness of the modeling approach. The exemplar model's capabilities are illustrated in the simulation experiments and the comparisons with real system's behavior (refer to the Chapter 5).

The realization of SOC-DEVS support modeling simple sequential workflows with component services arbitrarily distributed in a network. However, mapping arbitrary workflows into composite services is challenging as it requires domain knowledge for creating abstractions. Current realization provides a single domain specific example. Thus, modelers interested in modeling other SOC systems need to extend the generic SOC-DEVS service models and hardware models. Apart from this, the SOC-DEVS simulator and the developed simulation testbed is capable of supporting larger scale systems (thousands of services and processors) with an almost linear simulation runtime (see Figure 5.9). However, a significant challenge lies in the verification and validation of simulation of such large scale systems (refer to Chapter 5, Section 5.3). This is because, the simulation models are verified and validated for a small scale system by comparing with the data collected from a real system. Hence in the absence of data from a real large scale system, the model's accuracy in capturing system dynamics at larger scale needs more careful consideration, i.e., the results of simulation at larger scale system can only be considered more as a prediction of the system modeled at that scale. This limitation is common to any simulation based approach. However, as the SOC-DEVS models, their compositions as well as the simulator are theoretically (DEVS formalism) grounded and the models built on first principles are validated to be capturing the important system dynamics for small scales, the simulation results at larger scales are expected to be the reflection of the real system (at the same scale and configuration). And in contrast to a black box approach where model internals are unknown, detailed analysis of the SOC-DEVS models at larger scale can be undertaken to have higher confidence in the simulation results.

Chapter 5

SIMULATION EXPERIMENTS

This chapter highlights simulation experiments on example SOC systems modeled in SOC-DEVS. Voice communication system, data encryption system and encrypted voice communication systems are modeled to represent exemplar SOC systems. Comparison with real system's data with similarity of trend is an important methodology evaluation criteria. Similarity in simulation and real system's data trend indicates the modeling methodology and abstractions can represent core system behavior. A simulation testbed with a prototype real system [57] is developed to support experimentation towards study and verification of the exemplar SOC systems. Data is collected from simulation and prototype real system for the same system configuration. Models are parameterized based on the the prototype real system. The simulation experiments are designed to observe the effect of system resource limitation on system QoS (e.g., delay and throughput). The simulation result is analyzed and compared with prototype real system data to observe the effectiveness of SOC-DEVS models in representing the exemplar systems.

5.1 Service Based Software System Example

From the perspective of system resource requirements, SOC systems can be largely categorized into the following types:

- Communication Intensive Systems
- Computation Intensive Systems
- Mixed Systems

A brief discussion along with some example systems of each type are provided in the following paragraphs

Communication Intensive System: This category of SOC systems consists of services where primary service functionalities are associated with high degree of data communication via network among services. The primary resource requirement in such systems is the communication bandwidth available towards meeting the end user's QoS requirements (e.g. data throughput, timeliness). In this type of systems, the dominant dynamics are the communications among services and the resource is the network bandwidth. Interactions among services are affected by the load at the network routers and links. Overhead due to computation (if any) is considered negligible in such systems.

Voice Communication System (VCS) is a good example of a communication intensive system. It provides streaming audio based on user requested sampling rate. The system primarily consists of server(s) hosting streaming services. The servers are connected via network communication devices (e.g. routers, switches) to the user's node. Encapsulated IP packets are used to stream data via the network. Streaming services can simultaneously stream data for multiple users. Sampling rate and streaming duration are the primary parameter the user provides. The sampling rate is an indicator of required QoS as it impacts the audio quality. In such systems, stream processing delay, network delay, available network bandwidth etc. along the route from the server to the client node contribute to the stream quality and hence the resultant QoS at the client end.

Computation Intensive System: This category of SOC systems are computation intensive and the primary resource requirements are the data processing resource (i.e., cpu cycles, memory size) available in the system. The dominant dynamics are the processing delay due to computation in services. Availability of system resources can impact service response time towards the clients. The communication overhead is considered negligible in such systems.

Data Encryption System is an example of computation intensive systems. It provides encryption services to end users. The system encrypts user requested data according to the user specified encryption parameters (e.g., encryption algorithm, key length, and percentage of encryption). The data encryption process is computation intensive and the resource requirement depends on the encryption algorithm, key length and size of user data. As such the performance of the system primarily depends on the system hardware capability. Generally longer key length increases encryption strength as well as the computation delay to process the data. So, the end user has to choose from a set of encryption parameters that suits the objective without experiencing significant service response delay [78].

Mixed System: This category of SOC systems has the property of both the computation and the communication intensive systems. The resource requirement consists of the data processing resource (i.e. cpu cycles, and memory size) available to the system in addition to the communication bandwidth requirements. The service performance measures (e.g., service response time, service throughput) are affected by a combination of factors depending on data processing and bandwidth resources.

Encrypted Voice Communication System provides capability to stream encrypted audio data. The user requests audio stream by specifying sampling rate and required encryption parameters. Based on the user specification , the system streams encrypted audio data. The data encryption process is computation intensive (i.e., cpu cycles and available memory) and the resource requirement depends on the encryption algorithm, key length and size of user data. On the other hand, the transmission of encrypted data consumes network bandwidth. As such the performance of the system depends on combinations of the system computation resource and network bandwidth availability. Generally longer key length increases encryption strength while the percentage of encryption increases

computation overhead to process the data and the sampling rate determines the network bandwidth requirement. The end user can choose from a set encryption parameters and sampling rate that suits the end user requirement without experiencing QoS degradation [78].

5.2 System Overview: Voice Communication System

Basic Attributes: The Voice Communication System (VCS) can stream various quality audio data specified by sampling rates (e.g., 44.1,88.2,132.3,176.4, and 220.5KHz) and the interested subscribers can subscribe to the audio data stream with QoS requirements on the data quality [76]. The higher sampling rates produce higher quality audio data as it contains more audio bits per sample. For example, sampling rate of 220.5KHz will produce superior quality audio data w.r.t. 44.1KHz sampling rate. The VCS under consideration supports a 2-channel (i.e., stereo) audio data that can be sampled at $44.1, 88.2, 136.4, 176.4$, and 220.5KHz rates. The subscribers request audio data stream for a specified amount of time over the network and expect the VCS to ensure subscriber's QoS (e.g., minimum throughput, maximum delay) requirements are met [78]. 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 audio data throughput provides a measure of the VCS performance and in general (i.e., under normal operating conditions), the VCS throughput is proportional to the aggregate of the individual audio streams being delivered. The atomic service can also be configured to encrypt audio data packets using Data Encryption Standard (DES) [17]. When data encryption is enabled, data packets can be encrypted using 64, 128 or 256 bit length keys and percentage of encryption (denoted as PE) can be specified. Encryption operation requires more computation resources and thus increasing percentage of encryption increases processing delay. However, packet processing

delay is considered negligible when encryption is disabled. Unless otherwise specified, Every audio data packets are encrypted (PE=100%) when encryption is disabled no packets are encrypted (PE=0%).

Basic Definitions

Sampling Rate: In digital signal processing, analog signals are first converted to digital signals through analogue-to-digital conversion process. Periodically, an instantaneous snapshot of the analogue signal is converted to digital equivalent data - the process is called Sampling [21]. The frequency at which the analogue signal is sampled is known as sampling rate and measured in Hz (i.e., samples per second.). If the sampling interval is T then the sampling rate fs is defined as the number of samples obtained in one second, or $fs = 1/T$. For example, if x(t) is a continuous signal, and the sampling is performed by measuring the value of the continuous signal every T seconds, the sampled signal $x[n]$ is given by: $x[n] = x(nT)$, with $n = 0, 1, \ldots, \infty$.

Stereo/Mono Sound: Stereo sound is the reproduction of sound using two independent audio channels through a symmetrical configuration of loudspeakers in such a way as to create the impression of sound heard similar to natural hearing with perception of depth, distance, and orientation w.r.t the listener. In contrast, mono sound is in the form of one channel, often centered in the sound field without any perception of depth and orientation of the sound source w.r.t. the listener. In voice-over-IP communication, stereo sound transmission requires both the left & right channel audio data to be encoded separately where as mono sound requires only a single channel (i.e., equivalent to linear combination of left & right channel audio data) to be encoded [27].

Measurements Of Interest

To quantify performance and QoS in the context of Voice Communication System, a set of measurements related to system resource status and system QoS are defined as follows

Processor Utilization: Processor utilization is defined as the ratio of the cpu cycles used in execution of a service w.r.t. the total cpu cycles generated over an observation period. It is also denoted as CPU utilization. For example, for a processor P, if N_s is the cpu cycles used in the execution of service s for an observation period T and total cpu cycles generated by the processor during the same observation period is N_T, then Processor Utilization $(P) = N_s/N_T \times 100$

Average Data Throughput: Average Data Throughput is defined as the amount of audio data bytes sent by the VoiceComm Service over the requested service period. It is measured at VoiceComm Service running at the server. All simultaneously active client's aggregated data bytes sent by the VoiceComm Service is considered in calculating the throughput.

Inter Data Frame Delay: Inter Data Frame Delay is defined as the time difference between events of two consecutive audio frame sending(or arrival) events from(or to) the VCS service at the server (or at VCS client). Due to data processing delay and delay at the network, the time delay between two consecutive data frame arrival can vary. At sender point, the inter frame delay is due to data processing delay at the processor and packet processing delay at the network card. However, at client end, the delay also includes the packet processing delay at network routers and switches.

5.3 Experiment Testbed

The experiment testbed consists of a server hosting VCS service and two client machines emulating multiple clients using threads (i.e. each thread generates a service request). The server and the two client machines are identical in hardware configurations (2.2GHz,1024 RAM, 100 Mbps NIC). The machines are connected by a 100Mbps router having 8 ports.

The testbed organization is designed to support basic VCS experiments under moderate scales to observe system behavior under normal and stressed system conditions. It allows simple and easy parameter modifications of service configurations and automates experiment runs and data collection process. .NET framework and C# language [19] is used in developing the prototype real systems. In addition, network packet level data is collected using NetMon 3.4 tracer [43]. Also, the simulation environment supports data collection using transducer models and trace files at each probe point of interest.

The testbed is developed to support basic experiments with the Voice Communication System to test system behavior under predefined parameter settings. The primary parameters with the VCS service is the sampling rates (44.1KHz to 220.5KHz) for any client request. When configured with encryption enabled [1], client can also specify key length (64, 128, 256 bits) as parameter in addition to sampling rate. Also, the number of simultaneous active clients (i.e. 5, 10, 15, 20, 40 etc.) is also considered. For comparative analysis, a simulation testbed is developed with simulated system representing the real testbed counterpart. A VCS service model is mapped to a processor model and subscribers are mapped to two other processor models. The processor models are interconnected by 100 Mbps network router model and each is configured (2.2GHz,1024 RAM, 100 Mbps

[1]Note: Unless otherwise specified, percentage of encryption is 100% whenever encryption is enabled

Table 5.1: Experiment setup configuration for Real & Simulated system

Category	**Real System**	**Simulated Systems**
Processor (CPU, Memory, NIC [2])	2.2 GHz,1 GB,100 Mbps	2.2 GHz,1 GB,100Mbps
Network Link Bandwidth	100 Mbps	100Mbps
Subscriber Number	1-40	1-40
Data Collection Duration	60 sec (wall clock)	60 sec (logical clock)

Network card) similar to the actual testbed.

All experiments are conducted in real prototype system as well as the simulated system. For any configuration, a set of 10 runs is collected and analyzed to have high confidence on the data and analysis results.

Simulation Parameter Estimation

SOC-DEVS software service layer models generic message based service interactions and the hardware layer is modeled to support such interactions. Together these model abstractions provide building blocks for simulation of Service-Based Software Systems. However, similar to other modeling approaches, domain knowledge serves a central role in developing useful simulation models. For example, service interactions through hardware layer requires data for computation load and communication load for different supported service operations. Data parameterization for model components are important for capturing resource dependent (which impacts time related behavior) system dynamics and needs to be estimated based on domain knowledge [76]. However, it is important to note that the modeler is ensured of capturing SOA-compliant service interactions and the resultant system dynamics on a case by case basis.

In the experiments, the real VCS system uses Data Encryption Standards (DES) [17] algorithm. The simulated VCS model does not implement the logic for DES rather accounts for the effect of encryption on the CPU load as a function of sampling rate. As such, the simulation setup needs CPU load factor approxi-

mation 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 rate S (i.e., $44.1, \ldots, 220.5$KHz) and the generated data sets are encrypted using the DES algorithm. If S denotes the Sampling rate (KHz), C is the Channel number (i.e., mono=1, stereo=2), B denotes bits per sample, then data generation rate, G = S×B×C bits/sec. 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 example, S=44.1KHz, C=2, B=16bits/sample, V=2.2GHz, U=5.7%, we get, LF =704cycles/byte. Also the average load on the CPU is, L=125.334Mcyles.

The processor parameters in the simulation are assigned based on real systems CPU configuration. The CPU speed for the processor is set at 2.2GHz and 1GByte of memory. For processor configuration test, the CPU speed of 3.2GHz and 1Gbyte of memory is used. The swap penalty is set for 0.1 logical second. The swap penalty activates only when a processor's available memory is less than required to process a job. The network bandwidth parameter is 100Mbps which is the bandwidth of I/O link at the network switches and network cards.

Environment Setup

The real experiments are exercised in a server hosting VCS service and two client machines emulating multiple clients using threads (i.e. each thread generates a service request). The server and the two client machines are identical in hardware configurations (2.2GHz,1024 RAM, 100 Mbps NIC). The machines are connected by a 100Mbps router having 8 ports. The real system is based on IIS server 5.0 running on Windows XP Service Pack 3, .NET version 3.5 and JDK version 1.6.

The simulation experiments are exercised in DEVS-Suite (ver 2.1) simulation environment. The SOC-DEVS simulator is part of DEVS-Suite. The simulator is setup in a machine with an Intel Core 2 Duo CPU E8200 at 2.66GHz and 3.23 GB of memory and Intel 82566DM Gigabit network card set at 100Mbs. The OS is Microsoft Windows XP Professional with Service Pack 3, .NET version 3.5 and JDK version 1.6. The network card is disabled while simulations are run and only the DEVS-Suite is the active user level application. For VCS model development in SOC-DEVS, eclipse (Helios) for Java (TM) Integrated Development Environment is used. During initial development the DEVS-Suite GUI mode is used. However for later experiments with multiple runs in each configuration, a console based streamlined version of the SOC-DEVS simulator is used.

Data Collection Process

The testbed is designed to support automated data collection process. Automated software modules are developed to drive the experiments with predefined parameter settings and configurations. Data collection is done both at the software code level as well as Windows Performance Object and network packet layer traces. The Figure 5.1 illustrates the flow of data collection. A special network packet trace utility (NetMon) and code level instrumentation technique is used.

The automated process has experiment coordinators running at the server and the client sides. The client side coordinators waits for execution command from the server side. Once the *start* command is received at the client end it starts clients. Once a run is completed it sends the *finished* command to the server side coordinator. Once the server side coordinator gets all the finished signal from client end coordinators - it continues with execution for other configurations or stops all data collection process and terminates.

The underlying block diagram for network packet capture and sound card

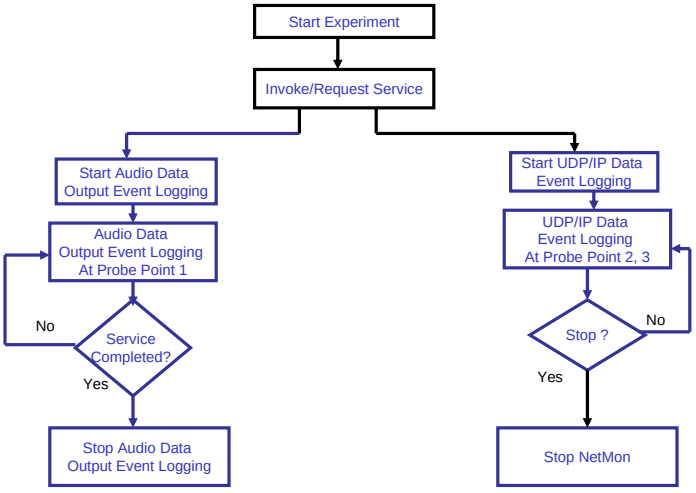


Figure 5.1: Data collection process

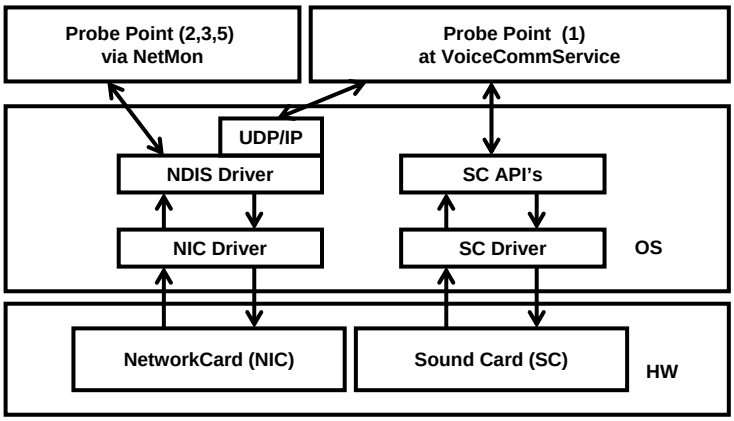


Figure 5.2: Data probe point layer view

interaction with the application layer is given in the Figure 5.2. Once the VoiceComm service gets a audio data event from the sound card it sends the data as UDP/IP packet which is captured by the netmon utility (outgoing and incoming packets).

Code instrumentation helps in collecting data at the application layer. The instrumented code region is minimal in nature to avoid unnecessary overheads or

```
private void Voice_Out (IntPtr data, int size)

{
      //add the current bytes to the total bytes

        bytesSent += (ulong)size;

      // Probe point 1 (at VoiceCommService StartStreaming Block)
      // Internally Logs Inter Frame Time

        traceLog.LogData(size,bytesSent);

      // for Recorder

        if (m_RecBuffer == null || m_RecBuffer.Length < size)
           m_RecBuffer = new byte[size];
        System.Runtime.InteropServices.Marshal.Copy(data, m_RecBuffer, 0, size);

      // Microphone ==> data ==> m_RecBuffer ==> m_Fifo

        r.SendTo(m_RecBuffer, new IPEndPoint(IPAddress.Parse(clientIP), clientPort));
}
```

Figure 5.3: Real VCS publisher code instrumentation

altering fundamental behavior of the services and clients. Since high precision (micro second granularity) is not needed for the evaluation purpose and the instrumented code has low overhead, the data and time stamps are accurate in millisecond range (i.e. ignoring instrumentation code overhead). Code instrumentation is done both at the service and the client side. For the service code, it is inserted inside the code block (see the Figure 5.3) that receives and sends audio data to the client end. Similarly in the client code, the instrumentation is inserted in the data receiving code block.

Windows operating system provides a performance profiling capability through a set of counters known as Windows Performance Object (WPO) [72]. These counters collect system status and performance information in the background as part of the OS itself. API is provided to use the counters to collect application as well as system specific data on the machine. Low level system in-

formation (for example: processor utilization, memory usage by processes etc.) is collected through WPO counters in the data collection process.

Network Packet Sniffing allows packet level data observation capability. The data collection process uses Microsoft Network Monitoring Tool NetMon 3.4. The captured network traces at server and client end is used to verify the service input/output and client input/output behaviors. The network packet traces allow correlating WPO counter data as well as instrumented code level data traces to more precisely relate data relation at different layers of abstractions. For example, WPO's FragmentsCreated/Second counter requires the MTU (i.e. Maximum Transmission Unit) value to relate to the network throughput. The packet traces provide the precise value of MTU to be 1500 bytes with payload size of 1480 bytes. In addition, UDP, IPv4 packet related information also provides important details of the system that aids in building improved abstractions for VCS.

5.4 Results and Analysis

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. In general, a VCS service is assigned to a processor and the subscribers are assigned to another processor interconnected via a switch. The background traffic generator is connected at the switch. The objective is to observe the VCS behavior and its QoS under varying load conditions by varying the number of simultaneously active subscribers. Unless otherwise specified, data analyzed is based on 10 runs for each of the real system and its counterpart simulation models. The observation time is 60 wall clock seconds for real system and 60 logical seconds for simulated system.

Effect of Background Traffic

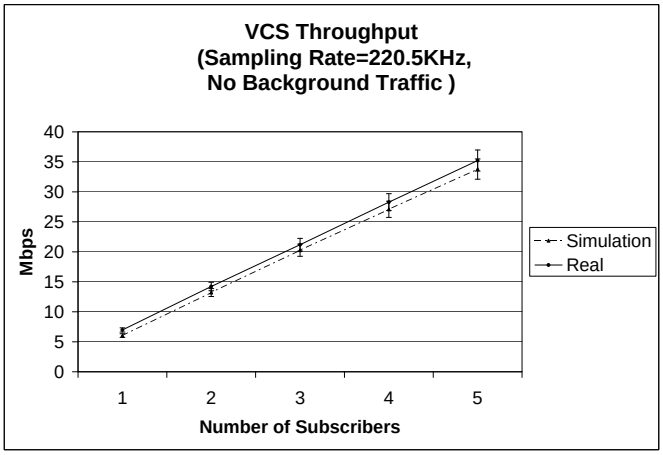
In Figure 5.4a, the simulated vs. real throughput of the VCS is shown. In this scenario the data encryption is disabled. A linear increase in VCS throughput with increasing number of subscribers is observed. This is comparable to the real system, where each active subscriber increases the streaming data throughput under normal operating conditions. Approximately, 35% of 100Mbs bandwidth is used by the active 5 subscribers.

In Figure 5.4b, 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 25Mbps (appx.) for 5 subscriber scenario due the background traffic consuming 75% bandwidth at the network switch.

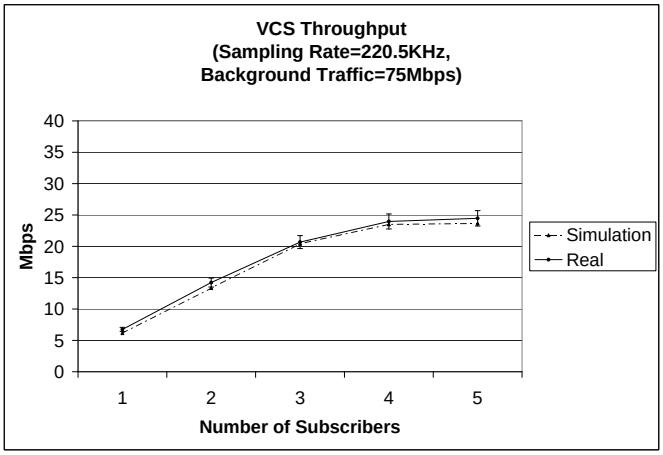
Effect of CPU Saturation

In Figures 5.5a and 5.5b, 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. The real and simulated VCS plots show throughput traces for 60 seconds (1 data point per second). The simulated VCS is parameterized based on real experiments. The real system is based on IIS server 5.0 running on Windows XP (SP3) machine which restricts the maximum subscriber number to 40 [39]. As such the data is collected up to 40 simultaneous subscribers.

In these experiments (refer to Figure 5.5b), the software services layer reaches a saturation point with increased CPU load (due to increased subscriber



(a) Average VCS throughput for Sampling rate=220.5KHz and no background traffic



(b) Average VCS throughput for Sampling rate=220.5KHz and background traffic=75Mbps

Figure 5.4: Effect of background network traffic on throughput of real and simulated VCS

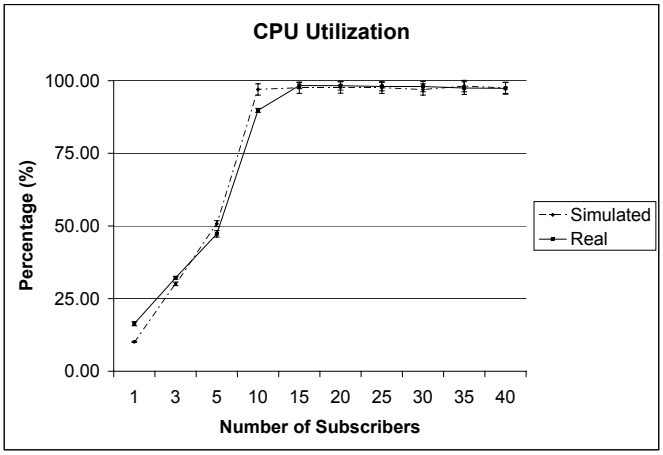
service requests) such that the effective throughput is not sufficient to use the available network bandwidth (note: no background traffic is used to load the network switch). It is evident that the CPU utilization increases with increasing

client number and the CPU becomes saturated which affects the VCS throughput. In both the real and simulated VCS, the throughput saturates around 10 clients. Though the saturation point differs with respect to the real system where various background threads interact in a complex manner, the trend line similarity shows the effect of the system dynamics as required of SOC-DEVS as an early architectural verification and validation tool for software-based software systems.

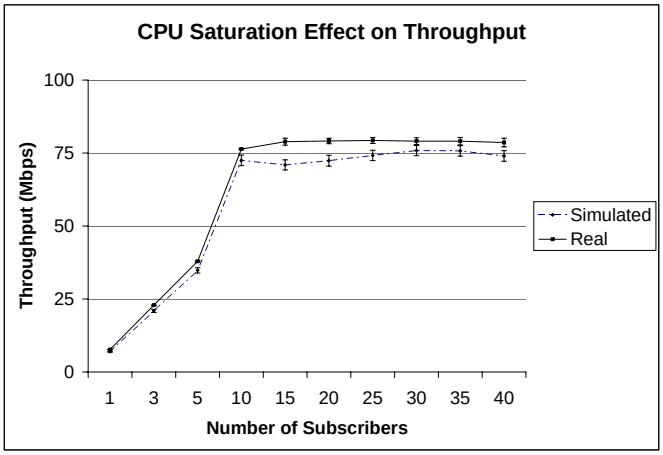
Effect of system configuration changes

In these experiments (refer to Figure 5.6, 5.7), we consider multiple QoS (i.e. namely - throughput and inter data frame delay). In VCS, the inter data frame delay time is defined as the time interval between any two consecutive data frames that are packetized and sent by the VoiceComm service. In Figure 5.6 and 5.7, the processor configuration is modified by changing the CPU speed (from (2.2GHz, 1GByte) to (3.2GHz, 1GByte)) to demonstrate how processor configuration can improve the throughput behavior and inter data frame delay.

With encryption enabled, each generated data packet is encrypted (CPU intensive) before it is sent to a subscriber. Increasing subscribers results in more data packets being generated and encrypted. As a result, CPU queue length and queue waiting delay increases (see Figure 5.6b) and this causes an increase in the mean inter data frame delay (as shown in Figure 5.7b). For the same load, the increase in CPU speed improves the software service layer execution speed with reduced CPU saturation and reduced CPU queue length and effective cap on the VCS throughput is increased (see Figure 5.7a) while inter data frame delay decreases (see Figure 5.7b). The changes are evident as shown in Figures 5.6b as the reduction in average CPU queue length denotes increased execution rate and hence the decreased inter data frame delay and increased VCS throughput.



(a) Impact of number of subscribers on CPU utilization

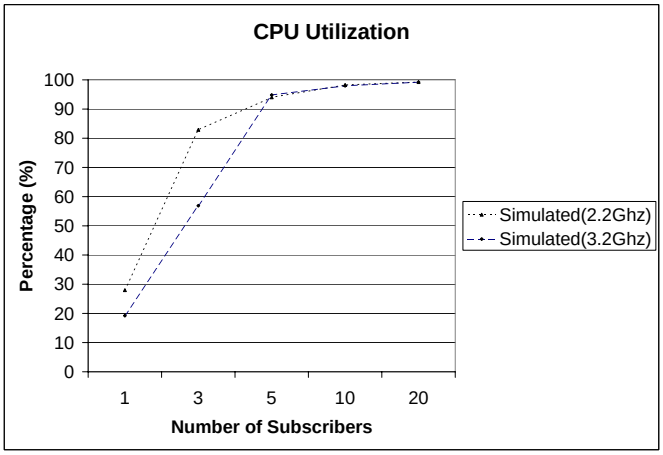


(b) CPU saturation effect on VCS throughput

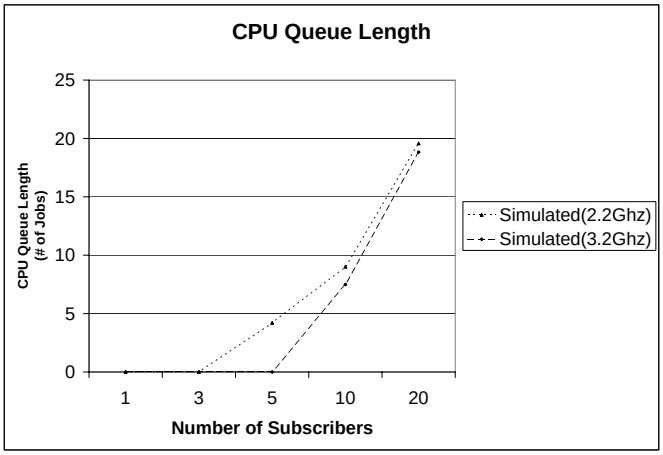
Figure 5.5: Effect of CPU saturation on real and simulated VCS throughput

5.5 Effect of System Scale

To observe system behavior at larger scales, a set of experiments are conducted with larger number of VCS publishers and subscriber. In his configuration, a VCS publisher is assigned to a processor and the subscriber are connected to the



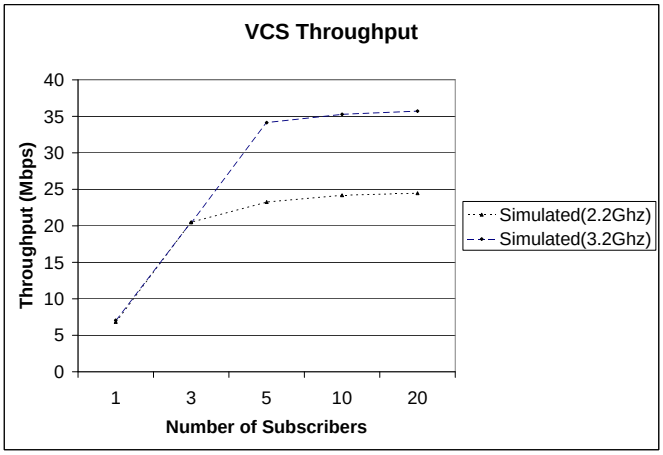
(a) Average CPU utilization on different CPU speeds



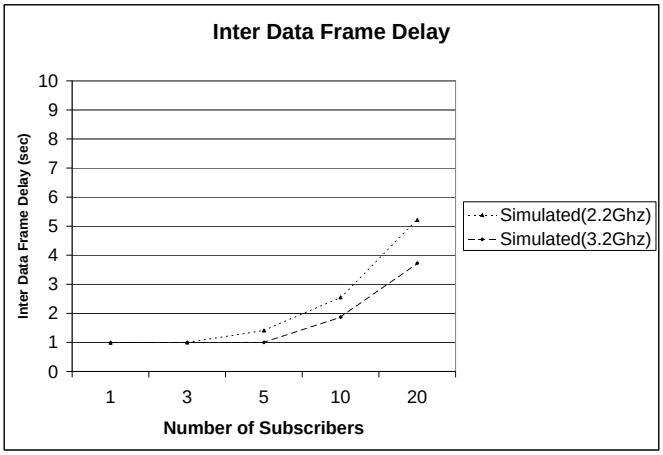
(b) Average CPU queue length on different CPU speeds

Figure 5.6: Effect of system configuration changes in simulated VCS

processor via a network router. The subscriber number is varied up to 1000 as the system scaling parameter. The QoS aspect consideration is the average VCS throughput and average inter frame delay over all subscribers. The scalability of the simulation runtime for each configuration is also measured. Simulation simulates 60 logical seconds and runtime is determined in wall clock time.



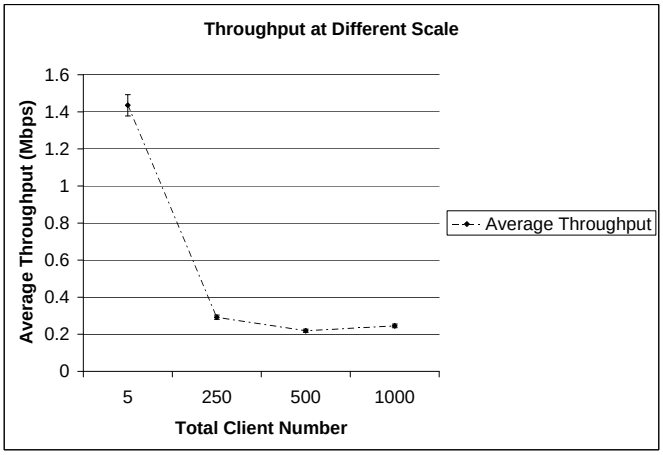
(a) Average VCS throughput on different CPU speeds



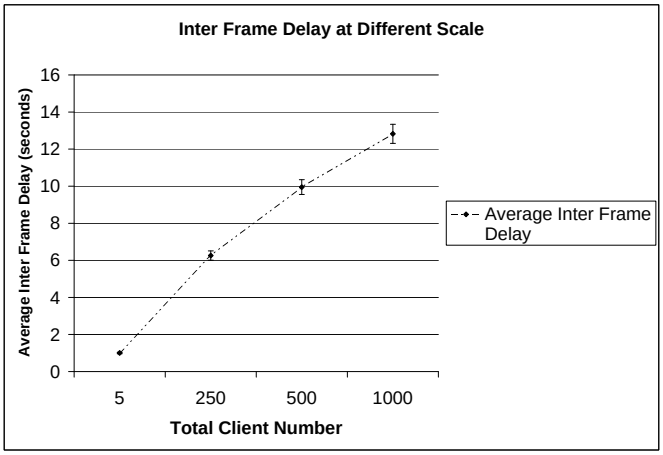
(b) Average inter data frame delay on different CPU speeds

Figure 5.7: Effect of system configuration changes on QoS of simulated VCS

As shown in the Figures 5.8, the QoS decreases with higher number of subscribers in the system. The simulated VCS has to serve a larger number of subscribers that consumes more system resources and requires more computation for data frame processing. Thus inter frame delay increases. In addition, larger number of streams contend for the link bandwidth this reduces available band-



(a) Average VCS throughput at different system scale



(b) Average Inter Frame Delay at different system scale

Figure 5.8: Effect of system scale on simlated VCS QoS

width for each stream. The net effect is a decreased throughput and increasing delay trend that is observed at larger system scales. From simulation execution performance perspective, the simulation runtime remains fairly linear as shown in the Figure 5.9 with the increasing simulated system model components.

It is important to note that systems at larger scale can behave unpre-

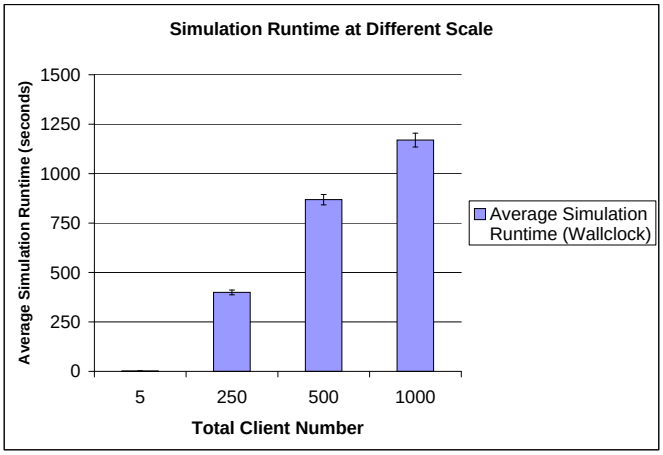


Figure 5.9: Effect of system scale on simulation runtime

dictably as behavior may become chaotic beyond certain boundary regions. As the models are validated and built based on first principle it is expected that based on the captured dynamics in the base models, the simulation prediction is capturing the system dynamics at larger scale.

Chapter 6

SIMULATION INTEGRATION FOR ADAPTIVE SBS DESIGN

In this chapter, the development of a prototype software system integrating SOC-DEVS simulator, with service monitoring & adaptation subsystem is discussed. The integration is facilitated using Knowledge Interchange Broker (KIB) [51, 52] which provides a skeletal basis for integrating such subsystems. In addition, an interaction scheme developed towards integration of simulation subsystem and monitoring & adaptation subsystem [75, 76] is provided [1].

6.1 Adaptive Service Based Software Systems

Design and configuration of Service Based Software System demands making trade-offs among multiple QoS features. Satisfying multiple quality of service features such as timeliness and throughput requires the capability not only to model the logical specifications of the services, but also being able to assess their dynamic behaviors. Runtime environment for Service Based Software Systems is very dynamic and poses new challenges to system design. 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. Due to fluctuations in system resource availability, the runtime QoS of the system is effected and thus, QoS fluctuations are expected. Hence, development of Service Based Software Systems demands the capability to monitor the changing system status, analyze and control trade-offs among multiple QoS features and adapt their service configurations accordingly. To maintain service level agreement on runtime QoS, services can be periodically monitored to detect degradation in QoS and adaptive measures can be taken to optimize system performance. Service Based Systems with such adaptation capability is known as Adaptive Service Based Software Systems (ASBS) [33, 76].

[1] Monitoring and adaptation subsystem is developed by Dazhi Huang

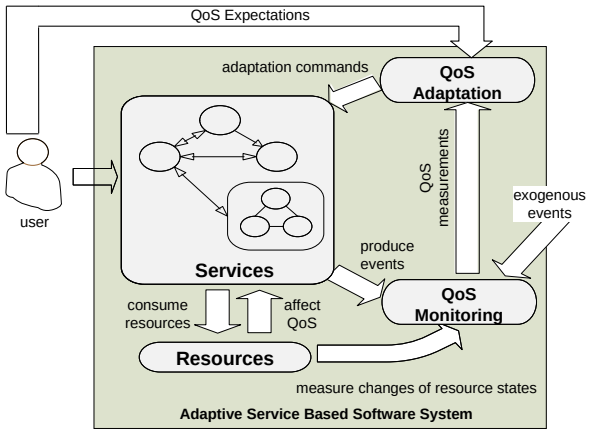


Figure 6.1: Conceptual view: An Adaptive Service Based Software System

The conceptual view of ASBS is depicted in Figure 6.1, 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.

As shown in the Figure 6.1, there are three fundamental parts to an ASBS - services, monitoring subsystem, and adaptation subsystem. User (i.e., service client) requests service specifying expected QoS (e.g., service delay, throughput). As services execute, they consume system resources which affects the service performance. Any such change in execution environment can cause system QoS to change dynamically. To detect such situation, the monitoring subsystems periodically sense the system QoS and resource status and notifies the adaptation subsystem of any detected QoS degradation. If any such degradation is detected, the adaptation subsystem can apply techniques (e.g., QoS optimization and/or policy enforcement) to adapt to changing system QoS or take corrective measures by allocating optimal system resources required to maintain overall system QoS

- thus completing the feedback loop. For example, under heavy system load scenario (e.g., a large number of active service clients), system QoS may degrade and either violate an in-session client's service level agreement or the system may become unable to meet a new client's expected QoS. As monitoring detects such degradation, the adaptation module can instruct services to reallocate system resources or enforce a policy of not granting further service requests to maintain expected system QoS.

6.2 Simulation Integration for ASBS Design

In ASBS design, the software design for monitoring and adaptation modules need rigorous experimentation under various system configurations. Due to the highly distributed nature of Adaptive Service Based Systems, an ASBS design testbed based on real service system is difficult to develop, configure and run experiments. In this context, simulation based design can improve experimentation capability with ease of repeatable and configurable experiments.Hence, an essential part of simulation based approach is a simulation environment that is capable of modeling Service Based Software Systems and support system model interactions to monitoring and adaptation subsystem software modules.

The monitoring and adaptation software modules under design and development can be driven by the dynamics of the simulated services and networked hardware under system configurations (e.g., topology) impractical to configure in real system deployment. The simulation subsystem can model complex systems consisting of service models and networked hardware models. Experiments under various system configurations (e.g., topology) that are infeasible or impractical in real systems (e.g., experiment management in geographically distributed systems) can be supported in simulation. The monitoring subsystem can collect system resource and QoS status from simulation subsystem, and adaptation subsystem can instruct the simulated system to account for the adaptation as specified by the

adaptation technique.

However, the integration of simulation subsystem with monitoring and adaptation subsystem requires a few challenges to be addressed. For example, services and hardware abstracted in simulation needs to provide system status and QoS data that conform to the structure and semantics the monitoring and adaptation modules expects. Interaction with monitoring and adaptation modules is complicated by the simulation subsystem's logical timing mechanism. Measurement of QoS or system status is time dependent and as such, it is logical time (as opposed to wall clock) based in simulation. Thus, the integrated system needs to develop a mechanism that allows a common time view based on simulation logical time. Since DEVS simulator executes DEVS simulation protocol, simulated system's interaction with monitoring and adaptation software modules needs to support the control and timing semantics required by DEVS simulation protocol. An important aspect is that the data flow between simulation and monitoring subsystem is former to the latter. In contrast, the data flow between simulation and adaptation latter to the former. Simulation must account for the bidirectional I/O and the timing aspect. In addition, simulation may provide data that needs to be aggregated to drive the monitoring subsystem. Hence, simulation, monitoring and adaptation subsystem interaction scheme needs to account for the following basic aspects

- Account for timing aspect of integrated system and time synchronization can be done from simulation side only
- Provide Flexible data transformation and I/O with monitoring and adaptation subsystem
- An interaction scheme that accounts for bidirectional I/O of the integrated system

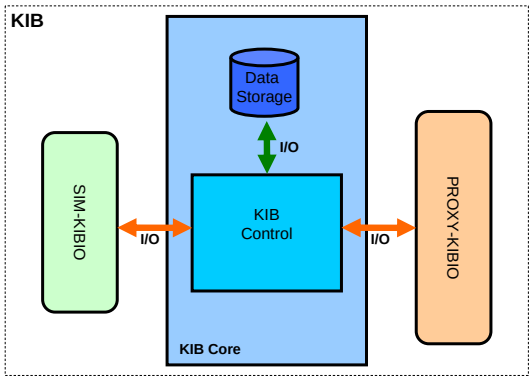


Figure 6.2: Knowledge Interchange Broker (KIB)

Knowledge Interchange Broker

Knowledge Interchange Broker [51, 52] is an approach that has been developed to enable composability of models of different formalisms. The conceptual basis for KIB multi-formalism modeling composition is the separation of model specification and its execution protocol. A modeling formalism can be considered as consisting of a model specification and an execution protocol with a unique syntactic and semantic characterization. KIB specifies event mapping, synchronization, concurrency,and timing properties that allow interaction of models of multi-formal specification. KIB can describe model composition and properties in terms of structure and behavior and it is intended to be used in model interoperability.

KIB provides generic mechanism and structured approach to integrating models [52]. Considering subsystems as models, the KIB approach can also be used in integrating multiple subsystems [50, 25, 31, 41]. Input and output mappings can support system integration of subsystems based on message types and interactions which are syntactically and semantically defined for the subsystem interaction. Hence, an important application of KIB is on characterizing how

different subsystems can interact with each other. Developing an integrated system using KIB requires accounting interaction semantics and message exchanges among subsystems. It supports simulation timing specification and controlling synchronization between simulation and other subsystems. In essence, KIB provides basic mechanisms important for system integration.

Integration Using KIB

The integration of SOC-DEVS simulator with monitoring and adaptation module is an important step towards simulation based design of ASBS. To integrate the subsystems, the KIB based approach is considered. The KIB has a core consisting of KIB engine and KIB data store. Bidirectional I/O through the KIB is supported by the KIB engine. It also supports data transformation for bidirectional I/O. KIB data store can buffer data from different subsystems and perform data transformations (e.g., aggregation, disaggregation). KIB also supports specifying timing of interaction among the simulation and other subsystems under integration consideration. KIB acts as the central component that ensures compatibility among interacting subsystems. However, the generic capability of KIB needs to be extended and developed based on the characteristics of the subsystems. Now, to develop an integrated system with KIB, the followings are considered

- Develop I/O for each subsystem with the KIB
- Develop data transforms and message mapping to ensure compatibility among subsystems
- Develop scheme of interaction with KIB for each subsystem

In this particular case, there are three subsystems namely the SOC-DEVS simulator, the monitoring subsystem, and the adaptation subsystem. Since simulator is the important subsystem to be integrated, the design decisions are fo-

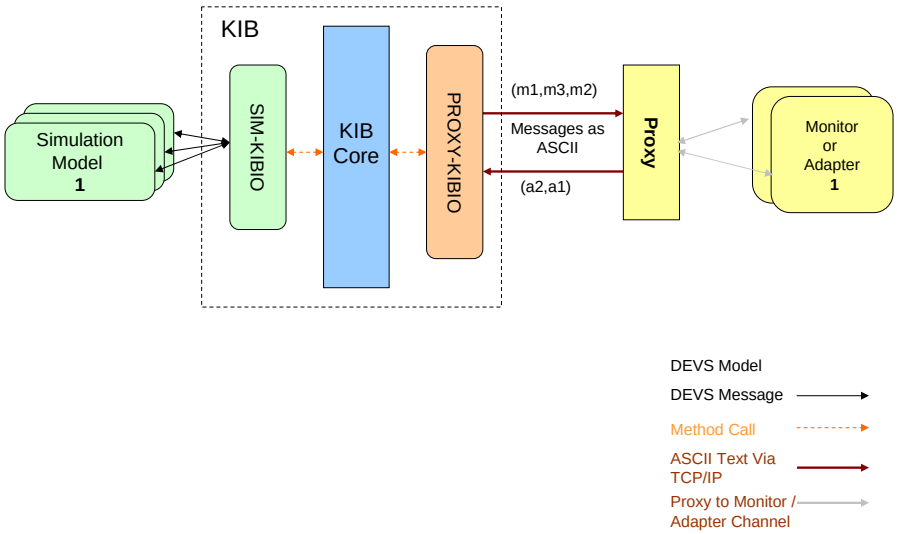


Figure 6.3: Simulator interaction with monitoring and adaptation subsystem

cused from simulation subsystem's perspective. As such simulation subsystem's I/O with monitoring and adaptation subsystems I/O's with KIB needs to be addressed. The architecture for the integrated system is given in the Figure 6.3.

The I/O is dependent on the specifics of the monitoring and adaptation subsystems. In this dissertation, the monitoring and adaptation subsystems are based on the proposal in [75, 76] . In this approach, ASQ (Activity, State, QoS) model is developed based on data analysis of experiments that collect QoS and resource status under various system activity. Then in runtime, the monitoring subsystem periodically collects system QoS and resources status that is used in conjunction with the ASQ model by the adaptation subsystem in inferring optimal system configuration. The resultant adapted configuration is used by the system to provide optimal QoS. The prototype realization of the approach uses client's service request parameter modification as a system configuration adapta-

Table 6.1: Inputs and outputs of M/A subsystems w.r.t simulation

Subsystem	**Input**	**Output**
Monitoring	Service QoS, Resource status	N/A
Adaptation	Service request, Service response	Service request, Service response

tion scheme. In this dissertation, the prototype realization is used towards this integration. Based on this realization, the monitoring subsystem requires service QoS and system resource status from simulation subsystem periodically and adaptation module optimizes service request parameters. Hence, the monitoring subsystem passively collects QoS and system resource status and the adaptation subsystem adapt service request parameters of any pending service requests based on status report from monitoring. To account for communication of monitors and adapters to the simulation via the KIB, a *ProxyKIBIO* module is developed (refer to Figure6.3). The ProxyKIBIO accounts for the message routing to/from distributed monitors, adapters and the KIB. In addition, both monitor and adapter subsystem each has a proxy developed that accounts for the message routings for each subsystem to/from ProxyKIBIO. For the case of the adaptation subsystem, the proxy provides message routing from the adapters to the KIB towards the simulation (i.e, simulated services and clients).

The table 6.1, shows the inputs and outputs for monitoring and adaptation subsystem that are relevant for the KIB based integration. Simulation subsystem needs to periodically (based on simulated logical time) send simulated service QoS and simulated hardware resource status to monitoring subsystem. Hence, the interaction between simulation subsystem and KIB is the sending of QoS, resource status messages to be forwarded to the monitoring subsystem. In addition, adaptation design choices require simulated service requests and responses to be also forwarded to and from adaptation subsystem. Data transformation in KIB consists of type changes to a type of message the monitoring and adapta-

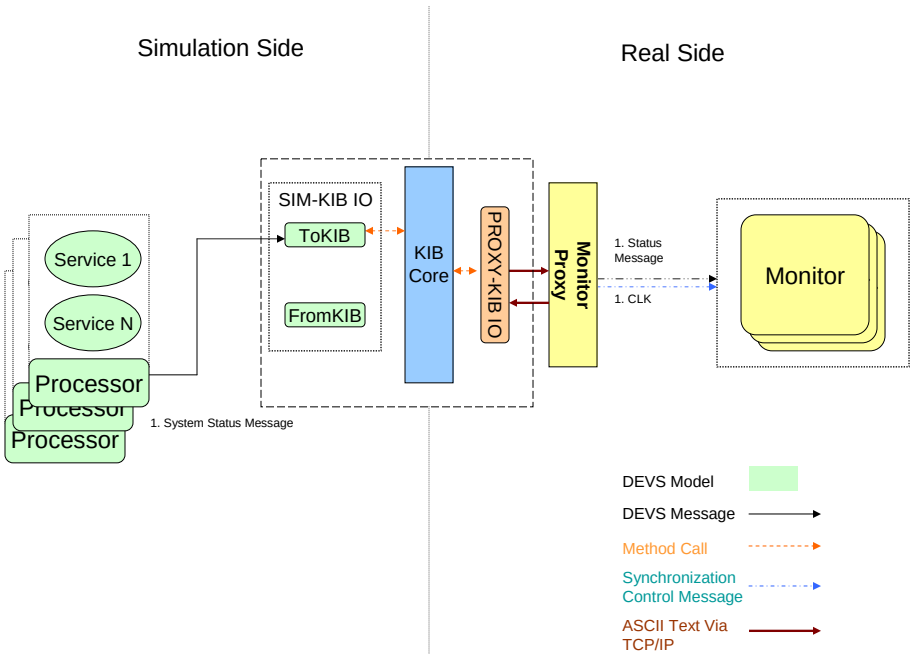


Figure 6.4: Simulation and monitoring subsystem interaction

tion subsystem expects. Time related aggregation or disaggregation feature in KIB is not used for the prototype. For the simualtion side, the DEVS models *ToKIB*, *FromKIB* models are developed that collect service QoS, resource status, service request/response messages from the simulation and KIB respectively. All data leaving for monitoring and adaptation subsystem are handled by *ToKIB*. Similary, all data for the simulation from monitoring and adaptation subsystem is handled by *FromKIB*. These two models are functionally similar to the proxy modules for the monitoring and adaptation subsystems.

The detail sequence of interaction via the KIB is shown in Figure 6.6. It shows the internal steps and sequences of actions that are part of KIB execution protocol. As shown in the Figure 6.4, the simulation subsystem generates system status messages that are sent to the KIB. KIB internally transforms the messages to types according to the monitor specification and forwards the messages to

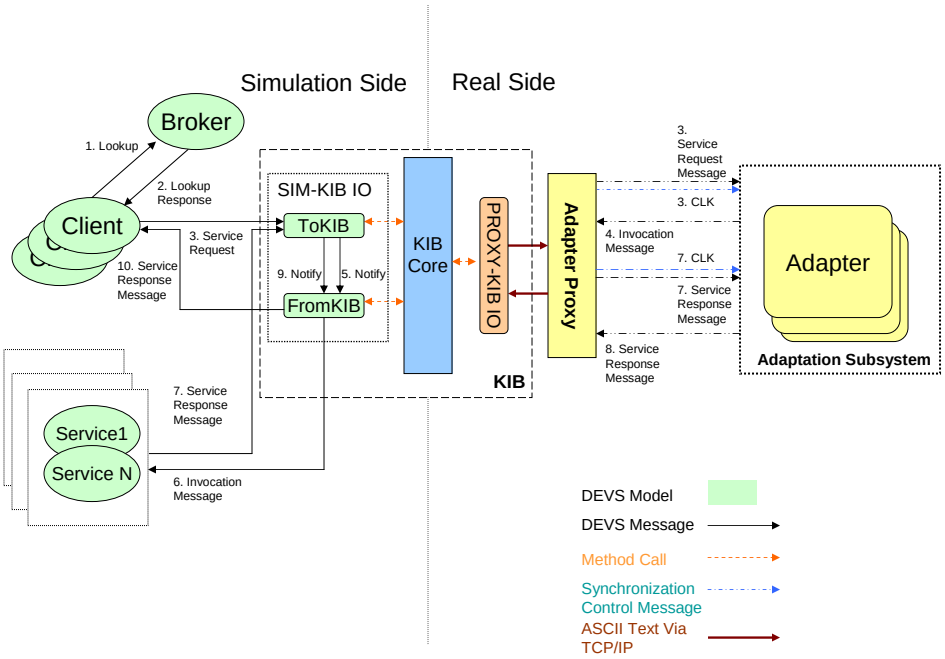


Figure 6.5: Simulation and adaptation subsystem interaction

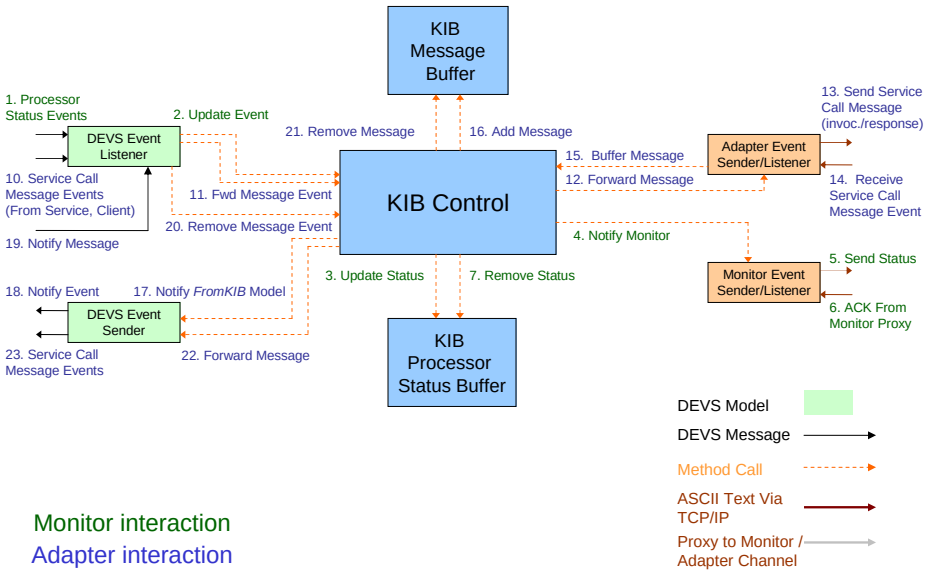


Figure 6.6: KIB Internal Sequence

the monitor proxy via the ProxyKIBIO module. The monitor proxy forwards the messages to the relevant monitors. Similarly on the adaptation subsystem interaction via KIB is shown in Figure 6.5. The service request, response messages are routed to the *ToKIB* model that forwards the messages to the adapters via the ProxyKIBIO and adapter proxies. If adapters have sent service requests with adapted parameters to the proxy, it sends the messages to the ProxyKIBIO and the messages are stored in the KIB data store and message available notification flag is set as part of *ToKIB* model behavior. The simulation side check for the flag periodically and if set then a notification message is sent to the *FromKIB* model from the *ToKIB* model. As such, *FromKIB* model retrieves all the stored messages from KIB and routes it to the appropriate service models. An important aspect of the integration with simulation is the timing aspect. To ensure a global view of time, the simulators logical time is considered to be the time base and used to time synchronization. As such, all interactions with the monitoring and adaptation modules are provided with the logical time stamp from the simulation so that real software modules can use the time base to asses time dependent QoS data in the simulation models.

Experimentation Support

An important aspect in the ASBS design using simulation is the QoS evaluation. The prototype integrated system can support experiments towards evaluation of system QoS. A set of measurements suitable for using the prototype monitoring and adaptation module for Voice Communication System are as shown in Table 6.2. Throughput and timeliness QoS aspects related to system performance is of particular interest in a network centric system like ASBS. Based on the adaptation and monitoring module requirements for the prototype Voice Communication System, the simulation provides QoS measurements of service data rate, service delay, network bandwidth, and network delay. In addition, the resource aspect for

the prototype system is computation and communication resources. Hence, the simulation provides CPU utilization, available memory and link bandwidth utilization on a per processor basis to the monitoring modules. The prototype design and implementation allows simulation, monitoring and adaptation subsystems to be physically distributed in a real network.

Various possible network topology and service configurations can be simulated and experimented with. For example, experiment setup can consist of multiple distributed processors interconnected by a network of links and switches. Services (publishers, subscribers and the broker) can be mapped to the distributed processors. Service interactions in the simulated system will effect the system resources and QoS. Based on system QoS status, the monitoring modules get notification and the adaptation modules can generate adapted service invocation parameters. The adapted service invocations are used in the simulated system. The system can thus adapt based on the adaptation schemes. Service delay, network bandwidth, CPU utilization etc. statistics observed for any configuration can be collected from the integrated system. Experiments for quantitative analysis of adaptation scheme performances can be designed. For example, monitoring and adaptation modules performance data like average length of deriving the optimal solution (i.e., monitor notification to solution generation) etc. can also be measured.

Followings are the definitions of the QoS and resource status measurements as used in the integrated system

- *Service Data Rate*: The rate of audio frame generation by service.
- *Network Bandwidth*: The end to end data transfer rate via the network.
- *Service Delay*: The elapsed time between user submitting a service request and receiving the service response.

Table 6.2: Example measurements using prototype integrated system

QoS Aspect	**Measurement**	**Unit**
Throughput	Service Data Rate	bits/sec
	Network Bandwidth	bits/sec
Timeliness	Service Delay	millisecond
	Network Delay	millisecond
Resource Aspect	**Measurements**	**Unit**
Computation Resource	CPU Utilization	Percentage(%)
	Available Memory	Bytes
Communication Resource	Link Utilization	Percentage(%)

- *Network Delay*: The elapsed time between sending a packet and its reception at the receiving end.

- *CPU Utilization*: The ratio of used cpu cycles towards computation w.r.t. total cpu cycles over the observation period.

- *Available Memory*: The amount of unallocated memory in the processor that are available for service executions.

- *Link Utilization*: The ratio of the used bandwidth w.r.t. the link capacity at the link in the processors.

However, based on the design of the adaptation and monitoring subsystem, the QoS and resource aspects and their measurements can be different from those mentioned above. In such cases, the system designer has to account for the type of system dynamics related data required to drive any particular monitoring and adaptation scheme of interest and design experiments based on the requirements. The developed prototype integrated system provides a foundation towards such simulation based design of ASBS using the capabilities in SOC-DEVS simulator in DEVS-Suite.

Chapter 7

CONCLUSIONS AND FUTURE WORK

In this chapter, a summary of the dissertation is provided. Future research challenges and potential research directions identified during the course of the dissertation work is also discussed.

7.1 Conclusion

In the early phase of the research, SOAD [53] is extended to support SOA-Compliant dynamic structure modeling. The resultant DSOAD modeling [40] is realized in DEVS-Suite simulator. Exemplar model of a Voice Communication System is developed in DSOAD. In addition, a real testbed is developed to study and experiment with dynamic addition and removal of services at runtime and observe the impact on Voice Communication System's QoS. The real testbed data analysis is then used in DSOAD model validation.

DSOAD [40] has been developed where 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 simulation to address QoS prediction and performance analysis of Service Based Software Systems [76]. Simulation in DSOAD can provide data that are closely related to the service layer. In addition, DSOAD support a very simplified representation of computational and communication hardware. However, it does not account for detailed hardware as DSOAD focuses on modeling abstractions for the software. 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 aggregate level view of the system resources. For example, Windows Performance Object [72] used in [76] is at the level of the system resource and

hardware, hence an ASQ [75, 76] model is developed to relate the service level behavior with the system hardware level data. Hence, a need for a detailed hardware resource and service representation in modeling SOC systems is identified.

This dissertation introduces the concept of HW/SW co-design into the SOA-Compliant DEVS to support detailed modeling for services and hardware parts of SOC systems. The developed SOA-Compliant co-design approach called SOC-DEVS enables better support for model verification and simulation validation by describing structure and dynamics of services and hardware. Modeling service behavior according to SOA has been emphasized in SOC-DEVS. SOC-DEVS provides primitive modeling concepts and constructs to support service modeling. In addition, computation and communication hardware models are represented separately from service models with capability to map services to hardware. Based on the idea of first principle, the service models & their interactions are derived from specifications in SOA. Based on networked software/hardware co-design concept, model specifications for software services as well as their interactions with hardware are developed. Service to service interactions through the hardware models are also accounted for. An implementation of SOC-DEVS approach has been realized in the DEVS-Suite simulator.

Exemplar model of a Voice Communication System and an Encryption System is developed in SOC-DEVS. A real testbed for Voice Communication System and Encryption System [1] experimentation is developed based on extension of an existing voice and encryption service [75, 76, 78]. The data collected in the real testbed experiments is used for parametrization of the exemplar models. Generic parameter values of computation and communication load could also be used if rigorous comparison with real testbed is not required. Further detailed experiments are conducted to validate the simulation models and verify the system

[1] Real VCS system was developed by Dazhi Huang. Refer to http://dpse.eas.asu.edu/sod

dynamics. The use of SOC-DEVS approach in architectural design of SBS is exemplified with the developed model and simulation experimentations.

An integrated prototype system consisting of SOC-DEVS simulator, prototype monitoring and adaptation subsystem [76] is also developed. Simulation integration is facilitated by the Knowledge Interchange Broker [51] such that simulated system are able to interact with the real monitoring and adaptation subsystem. Simulator sends processor status data and system QoS data to the monitoring subsystem while service requests are send to the adaptation subsystems. Adaptation is defined in the prototype as reconfiguring service requests as opposed to service and resource reallocation or reassignment. Based on the available system resources and QoS requirements, the adaptation modules reconfigures the service request message with optimal parameter values (which can vary from what the client had originally requested). Simulated services are invoked in the simulation using the adapted service requests. The design of prototype integrated system is flexible in supporting interactions with different monitoring and adaptation module implementations. Thus, the integrated testbed provides a foundation (refer to Chaper 4, Section 4.5) for simulation based design and development of SBS.

7.2 Future Work

The existing integrated prototype system is capable of basic experimentation with monitoring and adaptation subsystems. Use of simulated services in place of real services allows verification and validation of monitoring and adaptation modules that are infeasible with experimentation using real distributed services. As part of future work, the developed prototype system can be extended to a comprehensive testbed towards co-design modeling and simulation experimentation to support larger scale ASBS design. An important effort in this context is 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 executing near real time will be a very important in simulation based SBS design.

In the current implementation, computation and/or communication intensive SOC system models have been developed with support for simple sequential composition of services. However, domain knowledge is necessary for arbitrary workflow composition in the current implementation. Future research can address on ways to improve support for arbitrary workflow composition using a visual modeler (e.g. CoSMos)[69, 13]. In additions, model abstractions are targeted for representing generic SOC components (e.g. Publsher, Subscriber, Broker, Router, NIC, Links). The base models may need to be extended to support concepts for domain specific model development. For example, the model of TranportUnit may need extension based on system specific (e.g. wireless communication protocol) requirements. Future research can focus into ways of extending SOC-DEVS models for such cases.

There are other interesting future research that can be undertaken. For example, ontology based service compositions and application-specific workflow patterns can enrich and simplify design and evaluation of Service Based Software Systems. For simulation execution perspective, one research direction is to support distributed simulation using web services. For example, the DEVS/SOA environment [54] can be used with DEVS-Suite simulator. The results of such research efforts can lead to more efficient simulations - thus, enabling design and development of arbitrary Service Based Software Systems.

[1] Architecture Description Language, http://www.cs.cmu.edu/ able/, 2011.

[2] 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, Buenos Aires, Argentina, 78-89.

[3] Asimow, M., 1962, "Introduction to Design", First edition, Prentice-Hall, 394.

[4] Azeez, A., S. Perera, D. Gamage R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, P. Fremantle, 2010, "Multi-tenant SOA Middleware for Cloud Computing", IEEE 3rd International Conference on Cloud Computing, 458-465.

[5] Barros, F.J., 1997, "Modeling Formalisms for Dynamic Structure Systems", ACM Transactions on Modeling and Computer Simulation, Vol. 7, No. 4, 501-515.

[6] Bass, L., Clements, Kazman, 2003, Software Architecture in Practice, Second edition, Addison-Wesley.

[7] 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.

[8] Benini, L., G.D. Micheli, 2002, "Networks on Chips: A New SoC Paradigm" IEEE Computer, Vol. 35, No. 1, 70-78.

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

[10] Buede, D.M., 2000, "The Engineering Design of Systems: Models and Methods", 1st Edition, John Wiley & Sons, 488.

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

[12] Buyya, R., R. Ranjan, R.N. Calheiros, 2009, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges

and opportunities", Seventh High Performance Computing and Simulation, Leipzig, Germany, 1-11.

[13] CoSMoS Modeler, 2009, http://cosmosim.sourceforge.net/, accessed November, 2011

[14] Concatto, C., D. Matos, L. Carro, F. Kastensmidt, A. Susin, M. Kreutz, 2009, "NoC Power Optimization Using a Reconfigurable Router," IEEE Computer Society Annual Symposium on VLSI, 235-240.

[15] Chapman, W.L., A.T. Bahill, A.W. Wymore, 1992, "Engineering Modeling and Design", CRC Press, 400.

[16] Chow, A.C., 1996, "Parallel DEVS: a parallel, hierarchical, modular modeling formalism and its distributed simulator", Transactions of the Society for Computer Simulation International, Vol. 13, No. 2, 55-67.

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

[18] DEVS-Suite Simulator, 2009, http://devs-suitesim.sourceforge.net/, accessed June, 2011.

[19] Microsoft Dot Net Framework, http://www.microsoft.com/net, 2011.

[20] Schmidt, D.C., 1997, "Distributed Object Computing", IEEE Communications Magazine, Vol. 35, No. 2, 42-44.

[21] Oppenheim, Alan V., Ronald W. Schafer, John R. Buck, 1999, Discrete-Time Signal Processing, Second Edition, Prentice Hall.

[22] Edwards, S., L. Lavagno, E.A. Lee, A. Sangiovanni-Vincentelli, 1997, "Design of Embedded Systems: Formal Models, Validation, and Synthesis", Proceedings of the IEEE, Vol. 85, No. 3, 366-390.

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

[24] Frankel, D.S., 2003, "Model Driven Architecture: Applying MDA to Enterprise Computing", 1st. Edition, Wiley Publishing Inc., 352.

[25] Godding, G., H.S. Sarjoughian, K.E. Kempf, 2003, "Semiconductor Supply Network Simulation", Winter Simulation Conference, New Orleans, Alabama, .

[26] 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.

[27] Gold, B., N. Morgan, 1999, Speech and Audio Signal Processing, Wiley Press.

[28] 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.

[29] Hild, D.R., 2000, "Discrete Event System Specification (DEVS) / Distributed Object Computing (DOC) Modeling and Simulation", PhD Dissertation, Electrical and Computer Engineering Department, University of Arizona.

[30] IEEE Standard for Modeling and Simulation High Level Architecture (HLA) Framework and Rules, 2010, IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000),1-38.

[31] Huang, D., H.S. Sarjoughian, W. Wang, G. Godding, D. Rivera, K. Kempf, H. Mittelmann, 2009, "Simulation of Semiconductor Manufacturing Supply-Chain Systems with DEVS, MPC, and KIB", IEEE Transactions on Semiconductor Manufacturing, Vol. 22, No. 1, 165-174.

[32] Hu, X., B.P. Zeigler, S. Mittal, 2005, "Variable Structure in DEVS Component-Based Modeling and Simulation", Simulation Transactions, Vol. 81, No. 2, 91-102.

[33] Jiang, C.H., H. Hu, K.Y. Cai, D. Huang, S.S. Yau, 2009, "An Intelligent Control Architecture for Adaptive Service-based Software Systems", International Journal of Software Engineering and Knowledge Engineering, Vol. 19, No. 5, 653-678.

[34] Karangelen, N.E., N.D.T. Hoang, 1994, "Simulation based design for large complex computer based systems", Systems Engineering of Computer-Based Systems,116-123.

[35] Kim, T., M.H. Hwang, D. Kim, 2008, "DEVS/NS-2 Environment: An Integrated Tool for Efficient Networks Modeling and Simulation", The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, Vol. 5, No. 1 33-60.

[36] 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.

[37] Kumar, S., J.H. Aylor, B.W. Johnson, W.A. Wulf, 1993, "A framework for hardware/software codesign" IEEE Computer, Vol. 26, No. 12, 39-45.

[38] Krafzig, D., K. Banke, D. Slama, 2004, Enterprise SOA: Service-Oriented Architecture Best Practices, Prentice Hall.

[39] Inbound connections limit in Windows XP, Microsoft Online Support, http://support.microsoft.com/kb/314882, 2010.

[40] Muqsith, M.A., H.S. Sarjoughian, D. Huang, S.S. Yau, 2010, "Simulating adaptive service-oriented software systems", Transaction of the Society for Modeling and Simulation, doi: 10.1177/0037549710382431.

[41] Mayer, G.R., H.S. Sarjoughian, 2009, "Composable Cellular Automata", Simulation Transactions, Vol. 85, No. 11-12, 735-749.

[42] NS-2, network simulator, http://www.isi.edu/nsnam/ns/, 2011.

[43] NetMon 3.4, Microsoft Network Monitoring Tool, http://www.microsoft.com/download/en/details.aspx?id=4865, 2011.

[44] OMNeT++, http://www.omnetpp.org/, 2011.

[45] OPNET, http://www.opnet.com/, 2011.

[46] Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F., 2007, "Service-Oriented Computing: State of the Art and Research Challenges", IEEE Computer , Vol. 40, No. 11, 38-45.

[47] Sage, A.P., J. Armstrong, 2000. Introduction to Systems Engineering, 1st. ed., John Wiley & Sons, Inc., 568.

[48] Sarjoughian, H.S., B.P. Zeigler, 1997, "Object-Oriented DEVS", Proceedings of the 11th SPIE confernce, Orlando, Florida, 100-111.

[49] Sarjoughian, H.S., D.R. Hild, B.P. Zeigler, 2000, "DEVS-DOC: A co-design modeling and simulation environment", IEEE Computer, Vol. 3, No. 33, 110-113.

[50] Sarjoughian, H.S., J. Plummer, 2002, "Design and implementation of a bridge between RAP and DEVS", Computer Science and Engineering, Arizona State University, Tempe, AZ.

[51] Sarjoughian, H.S., D. Huang, 2005, "A multi-formalism modeling composability framework: agent and discrete-event models", IEEE International Symposium on Distributed Simulation and Real-Time Applications Proceedings, 249-256.

[52] Sarjoughian, H.S., 2006, "Model Composability", Winter Simulation Conference, Monterey, CA, 149- 158.

[53] 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.

[54] Mittal, S., J.L. Risco-Martn, B.P. Zeigler, 2007, "DEVS/SOA: A Cross-Platform Framework for Net-Centric Modeling and Simulation in DEVS Unified Process", Simulation Transactions, Vol. 85, No. 7, 419-450.

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

[56] Howell, F., R. McNab, 1998, "SimJava: A discrete event simulation package for Java with applications in computer systems modelling", 1st International Conference on Web-based Modeling and Simulation, San Diego, CA, 51 - 56.

[57] Muqsith, M.A., H.S. Sarjoughian, 2010, "A Simulator for Service Based Software System Co-design", Proceedings of the International Conference on Simulation Tools and Techniques, 1-9, Torremolinos, Malaga, Spain.

[58] Schulz, S., T.C. Ewing, J.W. Rozenblit, 2000, "Discrete event system specification (DEVS) and StateMate StateCharts equivalence for embedded systems modeling", Proceedings of the 7th IEEE International Conference and

Workshop on the Engineering of Computer Based Systems, Edinburgh , UK, 308-316.

[59] Tsai, W.T., R. Paul, B. Xiao, Z. Cao, Y. Chen, 2005, "PSML-S: A Process Specification and Modeling Language for Service-Oriented Computing", International Conference on Software Engineering and Applications, 160-167.

[60] Tsai, W.T., C. Fan, Y. Chen, R. Paul, 2006, "DDSOS: a dynamic distributed service-oriented simulation framework", Simulation Symposium, doi:10.1109/ANSS.2006.17.

[61] Tsai, W.T., B. Xiao, Q. Huang, Y. Chen, R.A. Paul, 2006, "SOA Collaboration Modeling, Analysis, and Simulation in PSML-C", IEEE International Conference on e-Business Engineering, 639-646.

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

[63] Tsai, W.T., X. Wei, Z. Cao, R. Paul, Y. Chen, J. Xu, 2007, "Process Specification and Modeling Language for Service-Oriented Software Development", Future Trends of Distributed Computing Systems, 181-188.

[64] Tsai, W.T., X. Zhou, Y, Chen, 2008, "SOA Simulation and Verification by Event-Driven Policy Enforcement", 41st Annual Simulation Symposium,165-172.

[65] Uhrmacher, A.M., 2001, "Dynamic Structures in Modeling and Simulation: a Reflective Approach", ACM Transactions on Modeling and Computer Simulation, Vol. 11, No. 2, 206-232.

[66] Unified Modeling Language, http://www.uml.org/, 2011

[67] Van Vorst, N., M. Erazo, J. Liu, 2011, "PrimoGENI: Integrating Real-Time Network Simulation and Emulation in GENI", IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS), 1-9, doi: 10.1109/PADS.2011.5936747

[68] Weiss, A., "Computing in the Clouds", netWorker, Vol. 11, No. 4, 17-25.

[69] Hu, W., 2007, "Visual and persistent co-design modeling for network systems", PhD Dissertation, Computer Science and Engineering Department, Arizona State University.

[70] Woodside, C.M., J.E. Neilson, D.C. Petriu, S. Majumdar, 1995, "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Serverlike Distributed Software", IEEE Transaction on Computers, Vol. 44, No. 1, 20-34.

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

[72] Windows Performance Object,http://www.microsoft.com/resources/documentation/, 2011.

[73] Xiong, P.C., Y.S. Fan, M.C. Zhou, 2008, "QoS-Aware Web Service Configuration", IEEE Transactions on Systems, Man and Cybernatics : Part A, Vol. 38, No. 4, 888-895.

[74] Xiong, P.C., Y.S. Fan, M.C. Zhou, 2010, "A Petri Net Approach to Analysis and Composition of Web Services", IEEE Transactions on Systems, Man and Cybernatics : Part A, Vol. 40, No. 2, 376-387.

[75] Yau, S.S., N. Ye, H.S. Sarjoughian, D. Huang, 2008, "Developing Service-based Software Systems with QoS Monitoring and Adaptation", IEEE International Workshop on Future Trends on Distributed Computing Systems, 74-80.

[76] Yau, S.S., N. Ye, H.S. Sarjoughian, D. Huang, A. Roontiva, M. Baydogan, M.A. Muqsith, 2009, "Toward Development of Adaptive Service-Based Software Systems", IEEE Transactions on Services Computing,Vol. 2, No. 3, 247-260.

[77] Yau, S.S., H.G. An, 2009, "Adaptive Resource Allocation for Service-Based Systems", International Journal of Software Informatics, Vol. 3, No. 4, 483-499.

[78] Yau, S.S., Y. Yin, H.G. An, 2011, "An Adaptive Approach to Optimizing Trade-off Between Service Performance and Security in Service-Based Systems", International Journal of Web Service Research, Vol. 8, No. 2, 74-91.

[79] Ye, N., X. Xu, P. Hurley, 2008, "QoS protocols for end-to-end delay guarantee of instantaneous jobs on computer networks", Information-Knowledge-Systems Management, Vol. 7, No. 4, 429-451.

[80] Zeigler, B.P., 1986, "Toward a Simulation Methodology for Variable Structure Modeling", Editors, M.S. Elzas, B.P. Zeigler, T.I. Oren, Modeling and Simulation of Methodology in the Artificial Intelligence Era, 185-210, North Holland, Amsterdam.

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

[82] 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.