Simulating Service-Oriented Systems: A Survey and the Services-Aware Simulation Framework

Michael Smit and Eleni Stroulia
Department of Computing Science
University of Alberta
Edmonton, Canada
Email: msmit,stroulia@cs.ualberta.ca

***Abstract*—The service-oriented architecture style supports desirable qualities, including distributed, loosely-coupled systems spanning organizational boundaries. Such systems and their configurations are challenging to understand, reason about, and test. Improved understanding of these systems will support activities such as autonomic run-time configuration, application deployment, and development/testing. Simulation is one way to understand and test service systems. This paper describes a literature survey of simulation frameworks for service-oriented systems, examining simulation software, systems, approaches, and frameworks used to simulate service-oriented systems. We identify a set of dimensions for describing the various approaches, considering their modeling methodology, their functionalities, their underlying infrastructure, and their evaluation. We then introduce the Services-Aware Simulation Framework (SASF), a simulation framework for predicting the behavior of service-oriented systems under different configurations and loads, and discuss the unique features that distinguish it from other systems in the literature. We demonstrate its use in simulating two service-oriented systems.**

***Index Terms*—simulation, web services, SOA, survey**

I. Introduction

Software-based simulation methods have been used for modeling and analyzing a broad range of phenomena and a variety of languages and tools have been developed to support the simulation of particular domains of interest. We define a *simulation framework* as a set of software components and tools intended to specify, execute, visualize, and analyze simulation implementations reflecting a range of systems within a domain. In this paper, our domain of interest is service-oriented (SOA) software systems, i.e., systems built using services (distinct units of functionality) composed together to form more complex systems. Any given SOA system usually involves many physical resources (network, storage and processing components) and logical organizations (each providing one or more services). Understanding and reasoning about such systems is challenging, as is deploying and testing them, and simulation is a promising approach for this problem.

This paper makes two primary contributions to furthering simulation as an approach to reasoning about service-oriented systems. The first contribution is a systematic survey of existing simulation frameworks, based on a novel formulation of the key characteristics of such frameworks. The second contribution is a services-aware simulation framework (SASF) that improves on the state of the art established in the survey.

The **survey** offers an overview of six simulation frameworks for service systems, describing their approach to areas of interest such as what modeling approaches are employed/supported, what tools are available to assist the creator of a simulation, what data capturing and visualization methods are employed, and how the framework has been validated. We conducted a systematic search for frameworks using the ACM Portal, IEEE Xplore, Google Scholar, Google Search, and Springer Link. The search terms were "services simulat(e|ion|ing|or)", "software component simulation", and "service-oriented simulat(e|ion|ing|or)". These keywords were chosen (and the results were filtered) based on a defined scope, namely "sets of software components and/or tools that are designed to support the creation of simulations that create virtual models of service-oriented systems, are capable of relating a "narrative" of system performance at each moment of a simulation, and are capable of running those models without integrating with real-world systems". The primary specification is simulation frameworks intended to support the simulation of service-oriented or componentized software systems. We excluded from our survey a) approaches to modeling software systems that are related to simulation – in particular, the queuing networks approach and Markov models, and b) the construction of emulation environments and testbeds. Though both are related and important, they are out of scope. There has been work on using general, non SOA-specific, frameworks and tools to simulate SOA systems; for example, see [1], [2], [3], [4]. In this paper, we focus our discussion on frameworks that have been explicitly designed to support the simulation of service-oriented or component-based software systems, and will refer to these as *frameworks*.

Our **SASF** framework makes three important contributions to the state of the art in this area. First it supports the automatic generation of executable simulations, based on existing data about the service. Second, it enables users to interact with a running simulation, through a user interface (a functionality typical of many other systems), but also through an API, so both simulation components and external applications can modify the simulation parameters during its execution. Finally, it includes a flexible metrics-gathering component, which, through its own API, can also be further extended to collect and visualize new metrics. It is the extendibility of SASF, through these two APIS, that makes it rather unique among the other frameworks we have reviewed, in that it supports

further development of automatically generated simulations, as required to represent more complex behaviors. Additional features include stochastic support where required and true replication of service performance. SASF has been used to replicate the behavior of two real-world service-oriented applications, and validated to faithfully replicate their performance. It has been used to generate data that informs other tools that reason about the system.

The remainder of this paper is organized as follows. We describe the six frameworks we found in our survey (Section II), then describe our own approach in detail (Section III). We compare all seven approaches based on the important characteristics of simulation frameworks in a taxonomy we define in Section IV. We illustrate our improvement on the state of the art through a case study, describing using SASF to simulate a service-oriented application (Section V). Finally, we summarize the contributions of this paper in Section VI.

II. Simulating Services

In this section, we describe six frameworks for simulating service-oriented systems. These frameworks will later be reviewed in terms of a set of characteristics, described in detail in Section IV. The frameworks and their characteristics are summarized in Tables I and II.

A. SOAD and DEVS

The Discrete Event System Specification (DEVS) [5] is a formal method for modeling the structure and behavior of complex systems. Its behavioral model defines states with lifespans, events (inputs), and outputs. Transitions between states occur at the end of a state's lifetime, or upon receiving external input. Transitions may optionally generate output. A component modeled in this fashion is considered atomic, but can be hierarchically coupled together with other atomic components. These hierarchical DEVS models can be simulated [6]. Since it was first introduced, various extensions, variations and simulation-engine implementations have been developed, including DEVSJAVA [7] and DEVS-suite [8]. The latter offers visual construction of simulations, as well as observation of and interaction with the simulation using a GUI.

Sarjoughian *et al.* propose SOAD [9], an extended version of DEVS which maps service-oriented concepts to the component-based model of DEVS, and describe a general simulator based on that model. The objective of SOAD is to verify logical correctness and certain performance elements of the service composition in terms of its throughput (no bottlenecks), timeliness, accuracy and quality. Most of the modeling and simulation features in SOAD are based on the existing DEVS formalism and tool sets, with some extensions for SOA concepts for which there is no direct translation; for example, composition is modeled as a series of nested hierarchies. They build their simulator on DEVS-suite.

At this point they consider a "simple" web-services scenario, with composed, loosely-coupled services that communicate using SOAP over a "simple" network model, which includes delay and transmission volume. The model can, in principle, be extended to accommodate more complex scenarios (*e.g.*runtime composition, network routing). Currently, it uses three atomic components: producers, subscribers, and brokers, with communication links between them. QoS metrics are recorded and visualized by DEVS-suite, in an animated time series. The scenario through which SOAD has been validated[9] is rather simple. It involves a fictional travel-planning service composition, with hard-coded timings; the output metrics whose expected value is obvious based on the fixed input metrics have the correct value; the fact that the composition produces the expected output is offered as evidence of the validity of the composition. There is no integration with real-world implementations, and no metrics are generated.

A related framework, called DEVS4WS by Seo *et al.* [10], used a similar translation to a DEVS model and implemented a simulation in DEVSJAVA (which includes a visual simulation builder). The focus of this work is on BPEL service compositions: given a set of services, the simulation produces valid BPEL compositions as sequential service invocations, based on common data types in the WSDLs of two or more services. Functions of BPEL other than message sequences are not implemented. The recognition of data types that services have in common is based on the name of the parameter in the WSDL message specification or the specification of complex data types. The simulation is based on the flow of data through the composition and not on performance or QoS metrics. Its use has been demonstrated through a proof-of-concept composition of services related to purchasing and registering a car.

B. DDSOS

The Dynamic Distributed Service-Oriented Simulation Framework (DDSOS) is the work of Tsai *et al.* [11], [12]. Their goal is to support the development of service systems by testing compositions in simulation. The approach begins at design-time, when a service-oriented system is modeled using PSML-S (Process Specification and Modeling Language for Services) [13], a language intended to provide the same features for SOA design that UML provides for object-oriented design (there is no translation support between the two). At the composition-level, relationships among services are visually specified; the actual service details and specifications are implemented separately. The PSML-S model allows for model checking of an SOA design, as well as the generation of test suites. From the PSML-S model and a simulation configuration file, code is automatically generated and deployed to a distributed, multi-agent simulation engine, where it can be validated using the test suites. [12] is a explanation of how SOA concepts map to PSML concepts (to produce PSML-S), and how the result is simulated.

DDSOS models are simulated by an extensible distributed, discrete-event engine. The simulation engine and all other framework components are all implemented as services, and are deployed as a set of federated components at run-time. A simulation federation client looks up deployed engines to which a generated simulation can be uploaded and run. The

results are recorded by each agent involved in the simulation and retrieved once the execution is completed. The simulation runs as quickly as the engines can process the event queues. No emulation is supported; the service-oriented architecture would allow it theoretically, just as it would theoretically support programmatic interaction with a running simulation, but neither are available "out of the box". Similarly, the potential for visualization is there but is limited in support.

The authors describe [12] the process and results of simulating a service-oriented solution to an escape problem: a runner and a bait entity collaborate at run-time to get past a guard. There is no real-world service for comparison, however they do identify several issues with their collaboration strategy from iteratively running simulations, analyzing results, and changing their general collaboration strategy.

C. MaramaMTE

Grundy *et al.* used an existing performance test-bed generator (MaramaMTE) [14] to generate stress tests for static service compositions. Their goal was to enable the architects of service systems to explore potential interactions among the composed systems that might impact the non-functional requirements (particularly QoS) of their composition. They use BPMN or ViTABaL-WS [15], their own custom visual WS composition modeling language, to model the high-level service composition. A custom software architecture notation can be used within the tool to model lower-level service interface details. Performance requests can be sent to the actual services or to generated stub services (in the paper, the service stubs are SQL queries executed on a database; there does not appear to be an attempt to replicate actual service behavior).

The composed services are stress-tested with load intended to emulate a remote client. Loads are created by hand (using time delays between requests of services in a composition) and by a method based on the Form Charts formalism by [16]. The intent is to emulate the actions of a user issuing a series of service requests by navigating a web site front-end.

They modeled and implemented a service composition based on searching for flights, choosing a flight, and choosing a seat. They identify this implementation as "simplistic", but were able to identify potential resource contention issues in concurrent requests. The results are accurate when assessing composition but do not predict or emulate real-world service performance. The strengths of their approach are suited to authoring simulations exploring design-time performance considerations.

D. SOPM

The Service-Oriented Performance Modeling (SOPM) framework, developed by Brebner *et al.* [17], [18], [19], offers a visual tool to define a service-oriented system in terms of its participating services and their composition. A deterministic performance model is constructed for each constituent service by sending a series of single requests to identify single-request response time, or by instrumenting the service implementation to report performance metrics. The model is made of building blocks like services, servers, workloads (workflows annotated with timing details), and metrics. The building blocks are not intended to be extensible; only simulations with a one-to-one mapping from real to simulated components are possible. From this hand-crafted model, a simulation is automatically generated to be run on their custom simulation engine (discrete-event, serial, local). The simulation allows for interaction at simulation-time to modify parameters dynamically. The simulation runs produce metrics for each simulated component.

SOPM has been used to model substantial (i.e. not toy) services and many-service workflows for service systems currently being developed. The authors describe using SOPM to model an Enterprise Service Bus called Mule [17] and calculate the total throughput. Their validation compared simulation-generated metrics to real-world metrics in identical configuration. They found a margin of error of up to 15%, which is explained by elements in the real world not included in the simulation model. They were able to make performance predictions based on the scenarios they simulated.

E. DAML

Narayanan *et al.* [20] described a knowledge-representation formalism that can be used for simulation (among other things) for Web service compositions. They do not mention SOA or most of the WS standards, but their composition-oriented approach is in line with SOA principles. They model services using a markup language intended for the semantic web, the DARPA agent markup language for services (DAML-S), which has since been superseded by OWL-S[1]. DAML-S is a process modeling language. This model is mapped to a first-order logic language to allow for situation calculus-based reasoning.

To simulate the modeled system, they translate the DAML-S representation to Petri nets. Petri nets have visual representations, yet allow mathematical analysis of processes; they support stochastic modeling; and they are commonly used to model distributed systems. They include places, transitions, and arcs: transitions are essentially events, places are roughly states, and a set of input arcs to a transition are roughly pre-conditions on the occurrence of events. Though Petri nets support continuous values, they choose to model a discrete-event system in their approach.

Their implementation translates DAML-S representations to Petri nets that can be executed in a simulator called KarmaSIM [21]. This visual tool shows the path of "execution" through the network. Since each transition has a probability of actually being taken even when pre-conditions are met, a variety of analyses can be conducted - detecting deadlocks, for example. Interactive simulations can be used to test compositions. A related benefit is the ability to automatically compose services to achieve a provider's goal. [20] describes modeling a book-buying web service; the analysis identified a potential deadlock in user account creation workflow.

F. Service Testing Tools

This set of services testing and development tools offers the ability to generate a service stub from a WSDL (emulation,

[1] http://www.w3.org/Submission/OWL-S/

which they refer to as simulation), to generate and send SOAP messages based on a WSDL (collecting performance metrics from the responses), and to perform similar functions for other communications protocols. The stated purpose is typically testing, including composition testing (using service stubs in place of outside services), automated testing (with a variety of messages), and load testing (with a large number of messages sent in parallel). Raw numbers can be displayed in a line graph; the metrics are focused on response time and bytes transferred.

These tools meet the inclusion criteria as they offer simulated (emulated) services and metrics gathering, but the simulated services are based on a model of the interface (*i.e.*, the WSDL) and not on the behavior or performance of the service being simulated. They do not improve the accuracy of the results over simulation, and do not offer the speed and flexibility of a simulated test. Though performance metrics can be gathered from the simulated services, their use would be limited to identifying bottle-necks or incompatible compositions. The purpose of the performance testing tools is to gather metrics from real services, which is useful but outside the scope of this survey.

Though an exhaustive list of such tools is outside the scope of this survey, commercial and closed-source offerings offering these features include soapUI[2], GH Tester[3], and CloudPort[4]. JOpera[5] is an open-source Eclipse plugin that allows the visual specification of service compositions, and visualizes the execution of that composition as the real services or service stubs are invoked. Genesis [22] is an open-source tool for generating SOA testbeds (including services, clients, registries, etc.) from defined models of a SOA environment.

The focus on individual services means that support for composition requires labor-intensive piecing together of a workflow, often in the form of scripting or perhaps in a GUI. Other tools, like Oracle BPEL Process Manager[6] offer superior support for load testing compositions, but offer fewer options at the individual service level.

III. Services-Aware Simulation Framework (SASF)

The services-aware simulation framework (*SASF*, *sass-if*) is a suite of tools supporting the modeling of a service-oriented system and the execution of this model in a virtual environment. The intended use is the generation and capturing of data predicting the performance of the modeled system, under different resource configurations and work loads. The advantage of time-based simulation over an analytic model is the ability to, not only predict end results, but also to produce an ongoing narrative of system performance at each moment of a simulation. The models built using this methodology, and the prototype tools, are focused on a set of metrics related to capacity planning (sometimes called performance metrics). An early version of the framework was described in [23]; the complete description below reflects the current state of the framework, including several extensions. SASF was created to meet a particular use case, one where a narrative could be created by understanding the current state of the system through all stages of the simulation, where real-time interaction was possible, with a focus on generating and visualizing metrics. As discussed in Section IV, it does surpass existing systems in these areas.

Fig. 1. The primary components and extensions of *SASF*.

The goal of this simulation framework is to reduce the development time required to create a simulation capable of accurately predicting the behavior of a software system. In terms of the lifecycle of an application, we do not expect SASF to replace existing commercial testing tools for testing or even planning/design stages, though it could in theory support such activities. The first key area of intended use is the *pre-deployment* stage. Once the services are implemented, a simulation can be created and used to design an appropriate deployment for the service system, supporting the process of better understanding, and systematically analyzing, an implemented service system. The second key area is *post-deployment and run-time*. After deployment, the simulation can be improved using the SASF tools by capturing more accurate performance profiles and logs of actual requests. Alternative configurations can be attempted in simulation to improve some aspect of the service system (here, primarily the performance). Off-line simulation can reason about the service system without interfering with it.

The framework includes four main components and two extensions (Figure 1), described further in the following sections.

1) The *Simulation Engine* provides functionality for running simulations (clock management, *etc.*) and a set of extensible libraries that implement common functionality (§III-A).
2) The *wsdl2sim* component takes as input the document describing the service interface and generates a simulation (§III-B).
3) *JIMMIE* is a tool that can be used to systematically generate alternative deployment configurations and to re-run the simulation to test a variety of scenarios (§III-C).
4) The *Metrics Engine* facilitates collecting, storing, and visualizing metrics generated by the simulation or from recordings of simulations, and creating a dashboard for presenting the visualizations (§III-D).
5) The *Emulation* extension enables integrating simulated components with real-world services (§III-E).
6) Finally, the *Service Testing Modules* extension provides request generators and monitors requests/responses to produce metrics (§III-F).

A. Simulation Engine

The discrete-event, time-driven simulation engine is implemented in Java, using objects to represent and simulate the behavior of their real-world counterparts, in some cases at

[2]http://www.soapui.org/
[3]http://www.greenhat.com/ghtester/
[4]http://www.crosschecknet.com/products/cloudport.php
[5]http://www.jopera.org/
[6]http://www.oracle.com/technetwork/middleware/bpel/index.html

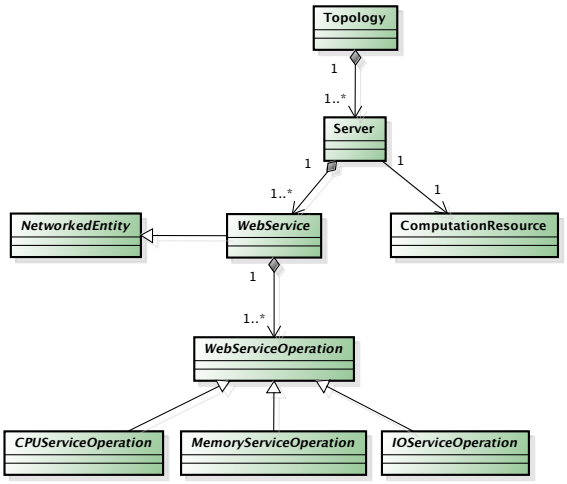


Fig. 2. A class diagram of selected extensible service objects.

a higher level of abstraction[7]. The natural mapping between simulated entities and real-world entities simplifies the task of a simulation author, one of the main goals of the engine. An extensible library of existing functionality reduces the "rote" code writing to create simulated entities, allowing the author to focus on implementing functionality.

The engine considers services and groups of services as entities on interconnected networks. These entities are linked by the exchange of messages over the network. By default, each entity is run in its own thread and only communicates with other entities over the network. The engine provides a set of Java classes implementing web services at different levels of granularity; simulation authors extend this base set of classes and override key methods with their own functionality (these implementations can be automatically generated). Several useful functionalities and sets of Java libraries are provided by the engine, described below. By employing these libraries, sophisticated topologies can be constructed - multiple tiers, replication, distribution, *etc.*

a) Timing and clock control: All entities in the simulation deal with time in actual time units, and the engine handles conversion to simulation time. The number of real seconds that pass in a simulated-environment second is configurable, based on the complexity of the simulation and the computation power available. The clock also offers timed events (*e.g.*, the author of the simulation can request to be notified in n minutes with a specific message or function call).

b) Network transfers: Simulated services need only identify the intended recipient and the message to send. The required network bandwidth limitations, routing, and actual transmission over time are simulated by the engine. Capacity limits are enforced based on configuration properties.

c) Basic service behavior: The meta model in terms of which simulated applications are specified is shown in Figure 2. This simplified UML diagram shows classes and their relationships; additional support such as interfaces and utility classes are not shown. Each application has a topology, describing its servers, networks, and what runs on those servers. Simulated entities are assumed to be resident on a particular server, which has a fixed amount of computation resources. Default implementations for topologies and servers with their computation resources are provided (and are extensible and integrated with the metrics engine).

A simulation author can start by extending any of the pre-defined elements: for example, one may choose to define a new type of an entity that is network capable by extending the `NetworkedEntity` element, or by implementing the individual operations that comprise a particular service. Thus, systems can be modeled and simulations can be developed at different levels of abstraction. All levels of abstraction offer certain core functionality: for example, a `NetworkedEntity` is capable of joining a network, packaging messages, sending messages and listening for messages. These operations are thread-safe. For this most abstract, least detailed representation, the simulation author needs to implement the message creation and message processing entirely. This offers flexibility, and more abstraction, but can be extended to offer more specific functionality.

Alternatively, services can also be simulated at a slightly less abstract level, but with more default functionality (`WebService`), where each individual service is simulated. In addition to the features of the most abstract representation, it can delegate to lower levels of abstraction, it provides metrics gathering/reporting functions that integrate with the metrics engine, and a local interface to timing/scheduling features. It manages queues of waiting requests and enforces service-level limits on member operations. This is particularly useful if the simulation author does not want to simulate at the level of individual operations, and would prefer to have a single implementation simulate the behavior of all the operations.

At the lowest level, operation-level simulation, operation templates are available in a variety of flavours (*e.g.*, for CPU-bound or memory-bound operations, *etc.*). A simulation author need only extend these stubs and initialize a set of parameters for the equations used to calculate total processing time and resource consumption. The existing implementation will perform calculations based on the performance profile specified and imitate the behavior of an operation with that performance profile.

d) Web Service messages: Simulated web services communicate using simulated SOAP or REST messages. An interface is provided and can be extended by simulation authors, or a default message can be populated and used. The *wsdl2sim* component generates compliant messages automatically (§III-B).

e) Dynamic configuration: A simulation is configured and run based on XML configuration files, allowing simulation authors to easily move configurable parameters to external files outside the code base. The *JIMMIE* module (§III-C) can systematically modify these configuration files. Default files include a generic name-value file and a topology file which defines servers, services, and operations. Each topology item specifies the Java class that implements that layer.

[7]For example, a simulated message is "transmitted" by the simulated network in the amount of time it would take the real-world network, but the process of creating packets and IP headers and so forth is abstracted away.

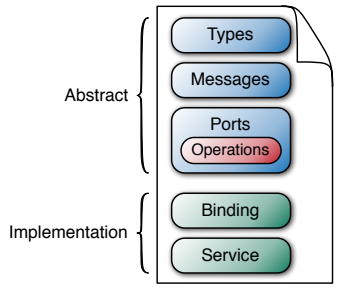


Fig. 3. The high-level structure of a WSDL document. Each component can be characterized as a more abstract description or as being more about implementation.

f) Service composition: is emulated by crafting special `CompositeMessages` that specify the business process. The simulation engine automatically handles composition by producing a series of messages to the services specified by the business process.

```
<?xml version="1.0" encoding="UTF-8"?>
<p:experiment count="1000" saveOutput="true" xmlns:p=".."
  xmlns:xsi=".." xsi:schemaLocation="..">
 <configDocList>
 <use URI="configure.xml" name="config"/>
 <use URI="./topology[0].xml" name="topology"/>
 </configDocList>
 <do>
   <range from="1" to="1000">
      <document id="topology">
         <changeElement name="server" withId="main">
            <addAttribute name="threads" to="15"/>
            <changeAttribute action="increment" by="100"
               name="memory" />
         </changeElement>
         <addElement name="service" parent="main"
            withId="service_id">
         <removeElement id="some_id" name="useless"
            withId="123"/>
      </document>
   </range>
 </do>
</p:experiment>
```

Fig. 4. A simple JIML document, which increments by 100 the `memory` attribute of the `<server>` tag that has id "main" for each of the 1000 experiments, add an attribute `threads` to the same element, add a new element `service`, and removes the element `useless` with id 123.

B. *wsdl2sim*

This framework component generates a simulation from the WSDL[8] specification of the service under consideration. That is, it creates Java source code that uses the framework libraries to replicate the service described in the WSDL (both in structure and function). The remainder of this section is organized in line with the supported top-level tags in a WSDL document (Figure 3).

The types section is for defining complex message types used by the service operations. The outer `<types>` tag wraps a series of tags based on the XML Schema Document syntax (XSD). A simulation data container object is created with fields for each simple data type listed there (based on the translation from standard XML types to appropriate Java data types). A constructor and getter and setter methods are also provided. This data container provides a mapping from the types defined in the XML schema to types in the simulation.

The messages sent and received by the web service are generated from the `<messages>` section. Each message stub class includes the necessary constructor, fields, getters, and setters to create and use message objects that represent actual messages. The author can then add or remove code to customize the representation. Code is also generated to instantiate a message object populated with syntactically correct but semantically meaningless data.

The operations of the web service are implemented in a series of classes that can also include performance profiles. These profiles, at the moment, must be dominated by one metric (CPU-bound, IO-bound, etc.) and are expressed as a function of input size. The current implementation uses linear equations fit to raw data; in principle, any equating and fitting mechanism may be implemented. Web services that are CPU-bound and in $O(n)$ are common, so this performance profile is the focus here. The equation expresses CPU time used per byte of input, plus some constant initialization time, plus some error factor representing the distance between the linear performance profile and the actual service performance. Thus, the CPU time used is $y = mx + b + e$, where m is CPU time per byte, b is initialization time, e is the error, for input of size x bytes. The CPU time is adjusted by a scale factor that represents how much faster/slower the test CPU is than a defined base processor. The performance profile also includes a similar equation for calculating the size of the returned response. The current implementation uses linear regression on metrics gathered automatically by sending requests of varying size to the operation under various service configurations and recording the results. This metrics generation is done using a tool that is functionally equivalent to WSUnit (https://wsunit.dev.java.net), with the addition of templates: an automated way to populate the content of a series of SOAP messages with semantically meaningful information.

In cases where no performance profile is available for the simulated service, the generated code is more abstract, receiving incoming requests and returning an empty response immediately.

The web service described by the WSDL is represented by a class capable of receiving requests and sending responses over the network. The automatically generated code instantiates operation objects and handles routing of incoming requests to the appropriate operation. It then transmits the response received from the operation. A corresponding testing class is also generated, which simply sends a WSDL-compliant message to the simulated service for each operation and reports what response is received.

The generated classes use the metrics engine to track metrics such as CPU utilization, response time, and number of messages received over time. The classes produced by *wsdl2sim* can be directly used by the simulation engine but they can also be further extended by the simulation author to improve the accuracy with which they reflect the corresponding services.

[8]Web Services Description Language, http://www.w3.org/TR/wsdl

C. JIMMIE: Systematic Simulation Configuration

JIMMIE[9] is the framework component responsible for automatically and systematically modifying the simulation configurations, represented in an XML-based syntax. This enables the use of SASF to test a system under systematically varied configurations and workloads. JIMMIE is essentially an XML transformation tool that makes potentially thousands or millions of transformations of a configuration file based on a set of user-defined rules, saves each iteration, uses it to run an experiment, and manages the output produced by the dashboard/metrics module to allow for easy retrieval. Once configured it runs autonomously; the experimenter can view and compare the results as they appear in the database. JIMMIE is designed to work on any XML document with any schema and to handle any number of configuration files.

JIMMIE takes its directions from an XML document that conforms to the JIML (JIMmie Language) schema (*e.g.*, Figure 4). JIML defines a number of experiments, and includes a list of rules which apply to all or certain subsets of those experiments. Each rule identifies the experiments to which it applies (start to end, 1 to 500, *etc.*). The rule then identifies the tags in the simulation configuration XML document to which it applies, and describes the desired values and attribute values for those tags. The values can be specified as an enumerated list of items, as a range with step values, as a loop, as an equation, or "hard-coded". Hard-coding is based on the experiment ID: [on the k^{th} experiment, set value of v to y]. A JIMMIE plugin based on an existing templating library[10] allows for the use of templates, which allows XML documents to be systematically modified with variable text content (replacing special template tags with defined blocks of text). A sample JIML document is provided in Figure 4.

The modification of XML may remind the reader of XSLT[11]. JIMMIE offers features akin to that of an XSLT engine specialized to systematically modify an XML file in a variety of ways producing potentially thousands of modified XML files. In addition, it manages invoking the simulation and tracking the location of the results. Because it is specific to the domain of simulation, it is easier to learn and use (instead of being general and broad like XSLT). Furthermore, it can be integrated into an existing codebase, accessed via its library API, or run as a separate application.

D. Metrics Engine

The metrics engine is a general library for recording, storing, and visualizing metrics. A simulation author can register any component of the simulation as a reporter to record metrics and report them via one of several API calls. The engine then stores and distributes the metrics to registered observers. Metrics are represented as triples, consisting of a time stamp, a name for the metric, and a value (number or text). The metrics are visualized in one of several default visualizations available, called *dashboard modules*.

[9]JIMMIE stands for Just Imagine Many Many Interesting Experiments.
[10]http://code.google.com/p/hapax/
[11]XML Stylesheet Transformations, http://www.w3.org/TR/xslt

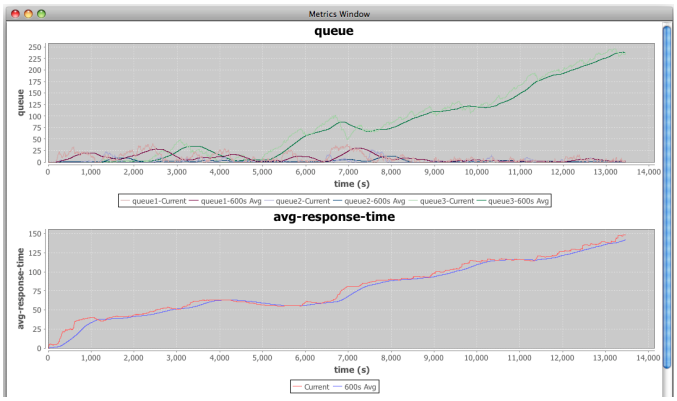


Fig. 5. The visualization metrics listener displaying two of the metrics being captured by a simulation.

The metrics module specifies *metrics listeners* which are notified each time a metric is added. Two listeners are provided by default. The database metrics listener caches metrics and periodically stores them in a database where a new table is created for each simulation. The visualizing metrics listener generates graphs for each type of metric it receives and updates the graph as more metrics arrive. It also creates a 600 second rolling average line by default. Figure 5 shows the two graphs automatically constructed by the visualization metrics listener.

Beyond the default functionality, a simulation author can define their own metrics listener to handle metrics in other ways. For example, Figure 6 shows a screenshot of a special-purpose metrics listener. It uses a pie graph, a line graph, and a custom visualization of client state (green = served, yellow = waiting, blue = currently being served). The sliders on the right can be used to change the server capacity and the network capacity. Another example, extending the dashboard to allow for run-time modification of the simulation, is shown in Figure 9. For another application, a listener capable of modifying the simulation was implemented: an autonomic computing module that monitors metrics, makes decisions, and implements changes.

An additional feature of the metrics engine is the simulation player, which allows one to select an experiment stored in a database. Using the existing visualization metric listener, metrics recorded during the original simulation can be "played back" (rewound, fast-forwarded, at various speeds) to review the results.

E. Emulation Extension

The role of the Emulation extension is to integrate the simulated environment with a real-world environment. Simulated applications allow for inexpensive performance testing and enable the implementation of features that may be easier to develop and test in a simulated environment than in the real world. Validating these testing results or new features in simulation is a first step, and implementing them in the real world is the final step. In between lies the process of testing a simulated feature in the real world.

The extension includes a translation layer that converts simulation `Messages` to corresponding SOAP messages. The

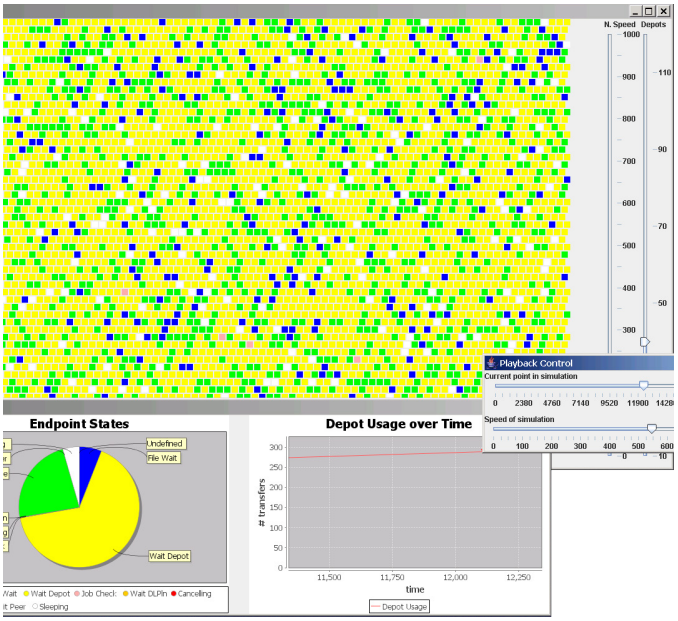


Fig. 6. A custom metrics listener and the playback control.

author specifies a mapping between simulated messages and the SOAP messages described in a WSDL, and a mapping between the WSDL operations and the simulated operations. The emulation layer converts simulation-generated messages into SOAP messages, and vice versa. This allows the use of simulation request generators, a mix of real and simulated services, and the interaction of simulated components with real-world services. A second translation layer aggregates metrics from real-world entities and includes them in the metrics engine; the existing metrics listeners, including the visualization dashboard, function as they do in simulation. The time-driven clock of the simulation engine is governed by actual time.

For example, this extension was employed to test load balancing strategies and an autonomic management feature, both implemented in simulation, using simulation-generated requests. The use of this extension requires a one-to-one mapping between simulated entities and real-world entities. Simulated servers cannot host real services, and real servers cannot host simulated services.

F. Service Testing Modules (STM) Extension

The purpose of the STM extension is the generation, transmission, reception, and metrics reporting of simulated service requests to enable testing (*i.e.*, it is a workload generator for simulated services). Request generation for web servers itself is well understood; however, request generation for services, and in particular for services in simulation is under-developed. This extension provides an extensible `ServiceTester` class, which employs a `Generator` to create requests. The base testing class is notified on each "tick" of the simulation engine clock, and requests from the `Generator` a set of requests to send for that particular point in the simulation. A set of `Generator` implementations is provided with different strategies for producing this set.

The generated incoming requests vary based on several parameters: the arrival rate of requests, the type of each request (*i.e.*, the operation being invoked, as each has a different performance profile), and the size of each request. These values depend on the following generators; each records and reports metrics on how many requests were generated and of what size.

- **Fixed**: The type of all requests, the size of each request, and the number of requests are described in a pre-defined configuration. These requests are generated as quickly as they can be processed. This is useful for performing tests that stress the application. This type of generation is used when validating the simulation.
- **Reading**: The generator reads from a log file that specifies a series of requests. For each request, the arrival time, type, and size are defined. This is useful when generating traffic based on logs of the existing service, or to generate identical traffic every time when conducting specific tests.
- **Stochastic**: Requests are generated based on a probabilistic distribution. The number of requests for any given second are determined by a Poisson distribution, where λ is a configurable job arrival rate. The type of each request is based on a weighted random distribution derived from logs of the real-world service. For the size of each type, the request sizes were extracted from the logs, then curve-fitting was used to identify and parameterize an appropriate distribution (usually exponential or Guassian). This is useful for generating roughly consistent but randomly varying traffic that can be used directly or recorded and used as input to the reading generator.
- **Modifiable Stochastic**: Requests are generated as in the stochastic generator, but some parameters can be modified at run-time: the job arrival rate and the weights for the type distribution (Figure 9). This is useful for creating variable loads for the same purposes as the stochastic generator.

IV. Characteristics of Surveyed Simulation Frameworks

The six frameworks discussed in the survey and SASF have similar overarching goals (to support reasoning about service-oriented systems), but diverse designs and implementations. In order to characterize and compare these rather diverse systems, we have defined key dimensions and characteristics involved in simulating a service-oriented system (see Figure 7 for an overview). In this summary, we describe each characteristic and then review the results of the survey for the various frameworks (SOAD, DDSOS, MarmaMTE, SOPM, DAML, Testing Tools, and SASF) with respect to that characteristic. The properties of each framework are summarized in Tables I and II. There are varied reasons for simulating service-oriented systems, and each simulation framework will have strengths best-suited to certain scenarios.

1. **Objective**. This characteristic captures the specific goal of the creators. We found that framework objectives can be classified into two general categories: (a) behavior analysis, in order to identify possibly undesirable interactions in the composition of the system's constituent services, and (b) performance analysis.

	Objective	Modeling	Evaluation
SOAD	verify logical correctness of the service composition pre-development	DEVS (SOAD)	Simple proof-of-concept
DDSOS	support the development of service systems by testing compositions pre-development	PSML-S	Simple composed problem, identified composition problems
MaramaMTE	help meet non-functional requirements in service composition before and during development	BPMN, ViTABaL-WS	"Simplistic" composed service, identified resource contention issues
SOPM	understand performance and scalability pre-deployment	Visual, JMeter for performance	Numerous uses; sample app within 15%
DAML	"enable markup and automated reasoning technology to describe, simulate, automatically compose, test and verify Web service compositions"	DAML-S	Modeled example network, found deadlock
Testing Tools	composition and load testing automatically from WSDLs	WSDL	–
SASF	create a virtual model of a componentized software application... focused on capacity planning	WSDL + perf. profiles	Modeled two applications; statistically validated metrics

TABLE I
THE OBJECTIVES, AND THE MODELING AND EVALUATION PROPERTIES.

	Basics	Interaction	Emulation	Visual
SOAD	Deterministic, Serial, Discrete-event, Event/Time-driven	–	–	Animated, Graphs
DDSOS	Deterministic, Parallel/Serial, Discrete-event, Event-driven	–	??	–
MaramaMTE	Deterministic, Serial, Discrete-event, Time-driven	–	Optional	–
SOPM	Limited stochastic, Serial, Discrete-event, ??	Manual	–	Animated, Graphs
DAML	Stochastic, Serial, Discrete-event, ??	Manual	–	Animated
Testing Tools	Emulation	–	Required	Graphs
SASF	Stochastic, Serial, Discrete-event, Time-driven	Manual, Autonomic, API	–	Animated, Graphs

	Authoring	Metrics	Speed	
SOAD	Visual, code generation from model	Targeted	Real-time	
DDSOS	Partial visual, code generation from model	Targeted	Accelerated	
MaramaMTE	Partial visual, code generation from model	Targeted	Real-time	
SOPM	Partial visual, code generation from model	Comprehensive, extensible	Accelerated	
DAML	Code generation from model	–	Accelerated	
Testing Tools	Automatic from WSDL (editable)	Targeted	Real-time	
SASF	Automatic from WSDL and performance profile	Comprehensive	Accelerated	

TABLE II
SIMULATION ENVIRONMENT PROPERTIES.

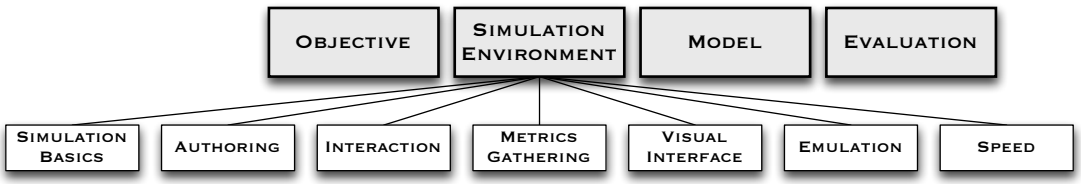


Fig. 7. A high-level view of the overall characteristics of simulation frameworks.

Survey results: The most common task supported by the frameworks is to test compositions before full-scale deployment. SOPM and SASF were the exceptions, focusing instead on capacity planning and performance testing. Both SOPM and SASF offer library support for composing services; rather than the main goal, it is a means to enable capacity planning based on simulated performance. Testing Tool solutions are focused on performance testing of individual services, though composition can be hand-scripted into the load testing. □

2. **Simulation environment**[12]. This characteristic, organized into 7 sub-categories, describes the set of features available to simulation authors.

a) Basics: Based loosely on Sulistio *et al.* [24], this covers the fundamentals of the simulation. Continuous simulations are capable of tracking a system through any point of time, usually based on differential calculus. Discrete-event simulations track a system based on specific moments in time (time-based), as events occur (event-driven), or based on traces from real-world applications. A deterministic simulation will produce the same output for a given set of inputs every time it is run; stochastic or probabilistic simulations use probability distributions and will produce output that varies based on the given distributions. Simulations can run in parallel, serially, or distributed (perhaps in a service-oriented architecture).

Survey results: A surprising number of frameworks offer only deterministic simulations: given the same set of input data, they will return the same result every time. Only DAML and SASF offer completely stochastic simulators; of these, SASF best supports the option to deterministically fix otherwise stochastic elements for systematic and repeatable testing. All of the frameworks examined, including SASF, offer or employ discrete-event simulation engines (this is the more common simulation method in other domains, as well). DAML offers the greatest potential for a continuous simulator, as the

[12]The various Testing Tool solutions are disregarded here; their environment is determined by the real-world and not a simulation engine.

underlying model is petri nets, which have been extended in a number of ways to enable continuous simulations (*e.g.*, [25], [26]). The majority are time-driven simulations; only SOAD and DDSOS offer event-driven simulations, which make them a better solution for simulating systems where events are sparse. The simulation engines all run individual simulations serially, except DDSOS which can optionally be run in a parallel mode. Only DDSOS offers the ability to run a distributed simulation. □

b) Authoring: Typically a simulation is implemented using a high-level domain-specific language or by programming using an API or library. Some simulation frameworks aim to ease the process of simulating a system by offering authoring assistance such as visual authoring tools and automated code generation based on the model or on the service itself (e.g. from WSDLs). Some simulation development environments offer tools intended to debug simulations.

Survey results: Only two frameworks offer automatic generation of a simulation, each with its own caveat. The Testing Tools generate a running service from the WSDL, but this version makes no effort to replicate the performance or behavior of the model. SASF generates simulated services from the WSDL, but uses performance testing results obtained semi-automatically: the SOAP requests need human input to be parameterized before being sent to the real-world system to generate a performance profile. Both approaches work for SOAP-based services, but not REST. The other tools offer visual environments to help build models of the system; simulations can be generated automatically from these models. In the absence of WSDL specifications, a visual tool for modeling is a better option. Some frameworks (like DAML) use the same approach to model the service at design time and guide implementation, which gives dual-purpose to the modeling stage. □

c) Interaction: Once a simulation is running, some frameworks allow run-time changes to the configuration of the simulated entities, while others do not. The former category encompasses several further variations, with some frameworks allowing for modifications by a user through a tool or programmatically through an API and others supporting autonomic modifications driven by the system itself.

Survey results: Here SASF is quite distinct, as compared to the other approaches. Only SOPM, DEVS, and DAML offer the basic ability to interact with a running simulation; all others adopt a "launch-and-forget" interaction style. SASF goes further, offering both an API and usable, extendable code for programmatically modifying the parameters of a running simulation. Properties of both the environment and the simulated application can be modified at runtime, thus enabling the simulation of systems which evolve at run time (either autonomically or by a system manager), a fundamental property of interest in service-oriented systems. □

d) Metrics gathering: Real-world systems can be instrumented to produce a variety of data. While simulations can be created to generate many types of data, often the focus is on a particular type of data (targeted). Others collect every metric generated by the simulation (comprehensive). Frameworks may be extensible to collect other metrics (*e.g.*, via an API).

Survey results: Again SASF is a standout; the features and APIs enabling the collection and dissemination of metrics require minimum effort by the simulation author. SOPM offers customizable metrics gathering. □

e) Visual Interface: Simulation results are usually reported in some textual representation during the simulation run, or upon its completion. In addition, some frameworks include a visual component to visualize the simulation or the metrics with either static images (default) or animated images. Also important is the ease with which the framework can be extended to augment or annotate the visualizing functions.

Survey results: The standard here is the generation of visual depictions of metrics or of the system architecture, often animated. Only DDSOS and MarmaMTE fall short in this category. SASF was designed to require less developer effort when introducing additional metrics to the visualizations. □

f) Emulation: Some simulations offer the ability to interact with real-world services or entities. Others require it as they cannot function on their own (for example, a simulation of a service broker might rely on real-world services).

Survey results: Testing Tools offer total integration with real-world services, where simulated services can even participate in compositions with already implemented services (with syntactically correct but semantically meaningless data). Only SASF and MarmaMTE offer any ability to integrate real-world services with simulated services that accurately reflect the behavior of the service. □

g) Speed: Simulations can run faster than real time, in real time, or slower than real time (due to computation required).

Survey results: As a consequence of their emulation abilities, the Testing Tools and MarmaMTE framework cannot run faster than a real-world service. SOAD also has this constraint. SASF runs at an accelerated pace, except in emulation mode. □

3. **Model**. Different simulation frameworks adopt different modelling languages, abstract representations, or formalisms for specifying the systems under examination. Each of these languages may make explicit (or alternatively, ignore) different aspects of the system's constituent services, their composition, the underlying infrastructure, the behavior, *etc.*

   Survey results: SASF and the Testing Tools rely on the existing model of a service as expressed in WSDL. This is advantageous if the service already exists and is described using a WSDL; if not, the increased expressiveness of other models may be superior. The others use typically more rigorous formalisms that allow the use of formal methods and proofs. SOPM offers a purely visual model. All of the frameworks support composition to some extent. MarmaMTE is notable for its use of BPMN as a modelling language. □

4. **Evaluation**. This characteristic describes the evaluation methodology, particularly whether the frameworks have been used to simulate "toy" pedagogical problems only or have been applied to real-world systems.

   Survey results: SASF is noteworthy as it has been used to model full-size real-world componentized software systems and statistically validated to accurately predict service performance, then used to reason about the system. SOPM has been used to address research problems. The remainder used simple proof-of-concept problems. □

V. Services Simulation Case Study

Having described the benefits of SASF over the state of the art, we now illustrate these benefits by describing the process of modeling and simulating a service-oriented application using SASF. TAPoRware is a text-analysis tool that provides a public web services interface. Its primary focus is on data processing (CPU-bound), with the movement of data being a secondary but still important concern. We use this case study to demonstrate a one-to-one relationship between real-life objects (services, operations) and simulated objects. This is made possible by the use of wsdl2sim. We have generated significant amounts of performance data about the real-world application that we use for a statistical evaluation of the similarity between the real and simulated application. Our simulation of TAPoRware was briefly described in 2009 [27] and has been used to illustrate simulation-based reasoning decisions [28].

A. Example Application: TAPoRware

TAPoRware is a single web service with 44 operations, implemented in Ruby using the SOAP4R libraries[13]. Each operation runs in $O(n)$[14]. The tools covered in this paper are listing the words with their counts, generating word clouds, finding the use of a word in context (concordance), and finding two words located near each other in text (co-occurrence). These operations are described in (Table III). To create more interesting examples, sometimes the single service is split into several services with varying distributions of the 44 operations among them.

A typical usage scenario begins with the end user identifying a piece of text to analyze with a given tool. Then, via a web-services client or the web front-end, the user inputs the relevant parameters for the analysis. Common options include word stemming, excluding stop words, and the number of words/sentences/paragraphs to display results in-context. In addition, each operation has its own configuration options. The text to be analyzed and the configuration options are encoded in a SOAP request and transmitted to the web service.

One challenge facing TAPoRware is performance. The service performs CPU-intensive operations on potentially large text corpora. Though it is capable of handling a small number of users, performance challenges may critically throttle its adoption.

Our simulation of TAPoRware tools included three steps as described in the following sections.

B. Building Performance Profiles

To determine the performance of a TAPoR operation, our performance tool sends repeated requests of varying size and with varying levels of concurrency (request concurrency measures how many requests are sent simultaneously). These requests are automatically generated using a combination of soapUI and a template engine. In most cases, all input options except the text to analyze were hard-coded to the default values. For the remaining cases, special tags were used to indicate where variable data was to be inserted. A Java template engine (Hapax[15]) was used to generate the final SOAP messages, populating the fields where variable data was expected with text of several fixed lengths (from 10,000 up to 175,000 words, or 57,000 to 1,000,000 bytes).

We used the Apache Benchmark (ab) tool[16] to send the SOAP requests. The requests sent for analysis were English-language novels from Project Gutenberg (most frequently Mary Shelley's *Frankenstein* or portions thereof).

First, we calculated the amount of time required to process a single request of a given size. Requests of several varying sizes were sent repeatedly. Next, we calculated the time required when concurrent requests were sent; this determines the effect of multiple concurrent requests on performance. Performance scaled linearly: when request concurrency was doubled, so did response time. This indicates that a single request was sufficient to completely use the resources of the test machine. From this data, we computed a linear regression and the error in the regression (the distance of the line from the actual data). This linear regression represents the performance profile as described in §III-B. Finally, stress testing was used to identify the limit of the service: that is, the point at which the service attempted to handle so many requests that it failed. The data for the List Words operation with request concurrency of 1 is shown in Table IV. Results for the other operations of the TAPoRware service are omitted but proceeded similarly.

Bytes	Response Time (ms)	Time per Byte
250	62	0.2499
56469	3086	0.0547
217012	8010	0.0369
416818	13537	0.0325
1000732	29235	0.0292

TABLE IV
List Words operation response times for input of varying size.

C. Generating Code

Using wsdl2sim (§III-B) and the performance profiles, we generated a simulator for TAPoRware to be run on the simulation engine. We used the automatically generated code without modification to validate this simulation. 1,145 source lines of code (SLOC) were created for a 6-operation sample simulation. In comparison, the framework itself is 7,500. These numbers, along with the SLOC counts for subsequent additions to the simulation, are shown in Figure 8.

We then extended the simulation to support additional reasoning and manipulation features. By using the features of the metrics engine, we extended the code to create a metrics visualization dashboard for TaporSim that included using the real-time configuration modification features (Figure 9). We created a load balancer in simulation based on existing load balancers by extending the simulation engine Services libraries. We extended the configuration files to allow for

[13] http://rubyforge.org/projects/soap4r/

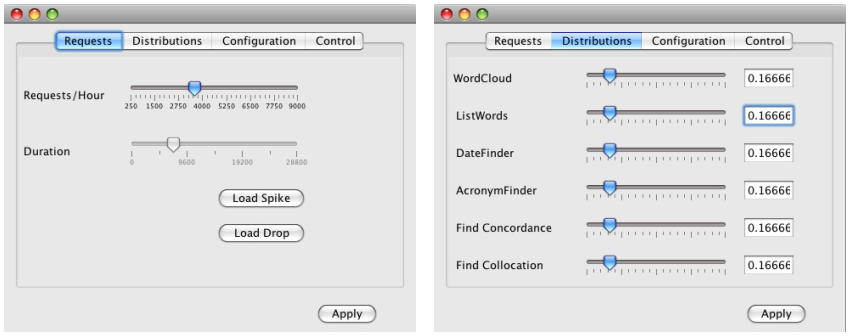
[14] In practice for this application; there is no claim regarding new algorithmic complexity for these operations.

[15] http://code.google.com/p/hapax/

[16] http://httpd.apache.org/docs/2.0/programs/ab.html

Operation	Description
List Words	Count, sort and list words of the source text in different orders. It can strip words user specified from the list, or common stop words.
Find Concordance	Find user specified word/pattern anywhere in the text with its context. Display the result in KWIC format.
Find Co-occurrence	Find user specified pattern and co-pattern in a given length of context. Highlight the patterns in the result.
Word Cloud	Using the top k frequent words reported by List Words, generate a visualization of those words and their relative frequency.

TABLE III
TAPOR OPERATIONS (FROM *http://taporware.mcmaster.ca/~taporware/textTools/*)



(a) Modifying request arrival rate (b) Modifying request type distributions

Fig. 9. The modification tool for modifying parameters of request-generation in real time.

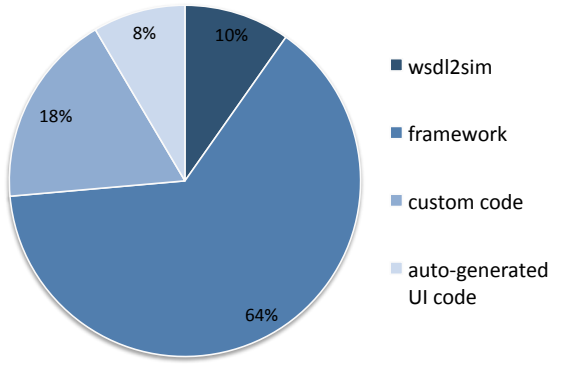


Fig. 8. The proportion of hand-written code versus code provided by a library or automatically-generated.

JIMMIE integration. The total custom code written for this simulation was 2,100 SLOC; roughly $\frac{1}{3}$ UI and metrics, $\frac{2}{3}$ request generating and various testing strategies. An additional 1,000 SLOC were generated by a Java UI creator that was used to create portions of the new user interface (Figure 8). We expect to be able to integrate one-half of the custom code back into the framework as general tools. The extended version of TaporSim was used to generate data we use to reason about the application.

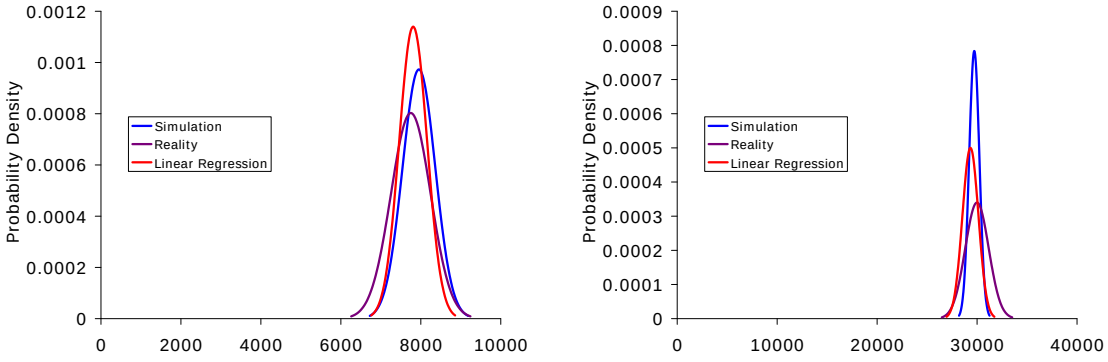
D. Validation

To evaluate the accuracy of our simulation of individual operations, we compared simulation-generated metrics to real-world metrics for each operation. Using Apache Benchmark, we sent requests of various fixed sizes to the real-world operation and recorded the response time (ms). The test requests consisted of blocks of requests. Each block had a varied request size and request concurrency and included 50 requests.

We replicated the real-world environment in simulation and gathered data using the same procedure. For brevity, details are provided only for the results of evaluating the List Words operation. The results do not differ substantially across all operations. In all cases, the automatically generated code was used without modification. The real and simulated environments were both configured with a single server having with 1 processor, 512 MB of memory, and a 100Mbps network connection. Each would process up to two requests simultaneously, queuing any other requests. We ran a two-way ANOVA (response time in milliseconds versus real/simulated and size) and found that the two sets of results were not statistically different (response time in milliseconds, $p < .05$).

Recall that the simulation is built on performance profiles generated from linear regressions. We also compared the predicted response time based on the linear regression (excluding the expected error information) to the actual response time as measured empirically. The linear regression only predicts performance with single requests, so only three test request blocks are applicable. For each block, we calculated the Mean Absolute Error of the two values, and expressed this as a percentage.The mean absolute error over all three applicable test blocks is 8.3%. Visually, Figure 10 shows the normalized distribution of response times for the List Words service under two input sizes for the real service, the simulated service, and for the response times predicted by the linear regression model.

Our performance model was based on real-world service performance for single concurrent requests. We made assumptions in the simulation about how it would scale to higher concurrency. Increasing request concurrency beyond the approximate capacity of the server results in increased response times for clients, but may reduce overall time spent processing



(a) Input size 217 KB

(b) Input size 1,000 KB

Fig. 10. Normalized distributions of service response times to single concurrent requests, comparing reality to the theoretic model (linear regression) and the implemented model (simulation).

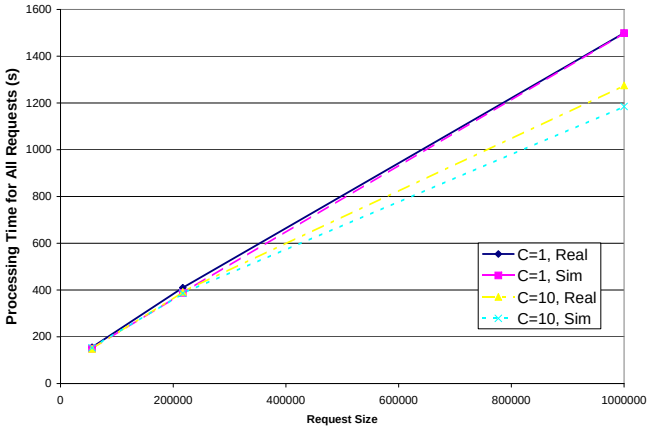


Fig. 11. The total time required for the real service and the simulated service to respond to all 50 requests for varying input sizes.

a fixed number of jobs. We compared total processing time in the real-world service with the same metric in our simulation, for request concurrencies of 1 and 10. In Figure V-D, it appears that our simulation accurately predicts total processing time. However, for request concurrency of 10, the simulation is overly optimistic about the improvement in overall processing time. Further improvements to the model may be required to decrease this margin of error.

VI. Conclusion

In this paper, we discussed SASF a novel framework for simulating services-oriented software systems. To place SASF in the context of related work, we systematically surveyed the literature for simulation frameworks and their approaches to supporting the simulation of service-oriented software systems. To guide this survey, we introduced a set of dimensions for characterizing simulation frameworks based on properties of relevance to those who developed these frameworks.

Each of the reviewed frameworks offers features and strengths appropriate for different tasks. SASF excels in

- its support for recording and visualizing behavioral metrics of interest, supporting special-purpose metrics and visualizations;
- its support for automatically generating executable service simulations, based on existing data about the services of interest, including their WSDL specifications and their implementations;
- its ability to receive interactive input from users and external programs, thus enabling the simulation of systems evolving at run time and the integration of real and simulated services; and
- its extensibility, both in terms of the implementations of the simulated components and the data collected during simulation.

SASF is built around extensible libraries for common web service functionalities that offer substantially more features than the current state of the art. Development effort is reduced using automatic generation of basic simulations from existing data about the service: from the services WSDL and a performance profile, an executable simulation is generated automatically, ready for extension and enhancement. The basic generated simulation can accurately replicate the performance characteristics of a real service; existing solutions focus on composition issues and not on a minute-by-minute replication of performance metrics. A narrative of the predicted performance of a simulation can be produced using the extensible and flexible metrics gathering support system with its own API, which out-of-the-box is capable of visualizing, recording, and playing back metrics generated during a simulation.

SASF also offers the ability to interact with a running simulation, including not just the usual user interface, but also an API so both the simulation components and external applications can modify configuration parameters during simulation execution. It can integrate simulated components with real-world components by translating requests and messages from the simulated environment to a real-world environment. It can generate requests based on real request logs or stochastic distributions modelled on known request patterns. Finally, JIMMIE is an innovative language and tool to run systematically modified simulations in parallel to generate data.

SASF was used to generate and test a simulation in a real-world case study. The simulated version was validated to produce the same performance results.

ACKNOWLEDGEMENTS

The authors received funding from AITF (formerly iCore), NSERC, and IBM. Andrew Nisbet was very helpful while implementing some components of SASF. The comments of the anonymous reviewers were particularly useful; thank you.

REFERENCES

[1] R. E. Nance, "Simulation programming languages: an abridged history," in *Proceedings of the 27th conference on Winter simulation*, ser. WSC '95. Washington, DC: IEEE Computer Society, 1995, pp. 1307–1313.

[2] L. F. Pollacia, "A survey of discrete event simulation and state-of-the-art discrete event languages," *SIGSIM Simul. Dig.*, vol. 20, pp. 8–25, September 1989.

[3] J. J. Swain, "Simulation software survey: To boldly go," *OR/MS Today*, vol. 36, no. 5, October 2009.

[4] Y.-H. Low, C.-C. Lim, W. Cai, S.-Y. Huang, W.-J. Hsu, S. Jain, and S. J. Turner, "Survey of languages and runtime libraries for parallel discrete-event simulation," *SIMULATION*, vol. 72, no. 3, pp. 170–186, 1999.

[5] B. Zeigler, *Theory of Modeling and Simulation*, 1st ed. Wiley Interscience, New York, 1976.

[6] ——, "Hierarchical, modular discrete-event modelling in an object-oriented environment," *Simulation*, vol. 49, no. 5, pp. 219–230, 1987.

[7] H. S. Sarjoughian and B. Zeigler, "DEVSJAVA: Basis for a DEVS-based collaborative M&S environment," in *Proceedings of the International Conference on Web-Based Modeling and Simulation*, 1998, pp. 29–36.

[8] S. Kim, H. S. Sarjoughian, and V. Elamvazhuthi, "Devs-suite: a simulator supporting visual experimentation design and behavior monitoring," in *SpringSim '09: Proceedings of the 2009 Spring Simulation Multiconference*. San Diego, CA, USA: Society for Computer Simulation International, 2009, pp. 1–7.

[9] H. Sarjoughian, S. Kim, M. Ramaswamy, and S. Yau, "A simulation framework for service-oriented computing systems," in *WSC '08: Proceedings of the 40th Conference on Winter Simulation*. Winter Simulation Conference, 2008, pp. 845–853.

[10] C. Seo and B. P. Zeigler, "Automating the DEVS modeling and simulation interface to web services," in *SpringSim '09: Proceedings of the 2009 Spring Simulation Multiconference*. San Diego, CA, USA: Society for Computer Simulation International, 2009, pp. 1–8.

[11] W. Tsai, C. Fan, Y. Chen, and R. Paul, "DDSOS: a dynamic distributed service-oriented simulation framework," *Simulation Symposium*, 2006.

[12] W. Tsai, Z. Cao, X. Wei, R. Paul, Q. Huang, and X. Sun, "Modeling and simulation in service-oriented software development," *Simulation*, vol. 83, no. 1, pp. 7–32, 2007.

[13] W. Tsai, R. Paul, B. Xiao, Z. Cao, and Y. Chen, "PSML-S: A process specification and modeling language for service oriented computing," in *The 9th IASTED international conference on software engineering and applications (SEA), Phoenix*, 2005, pp. 160–167.

[14] J. Grundy, J. Hosking, L. Li, and N. Liu, "Performance engineering of service compositions," in *SOSE '06: Proceedings of the 2006 international workshop on Service-oriented software engineering*. New York, NY, USA: ACM, 2006, pp. 26–32.

[15] N. Liu, J. Grundy, and J. Hosking, "A visual language and environment for composing web services," in *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. New York, NY, USA: ACM, 2005, pp. 321–324.

[16] D. Draheim, J. Grundy, J. Hosking, C. Lutteroth, and G. Weber, "Realistic load testing of web applications," in *Proceedings of the 10th European Conference on Software Maintenance and Reengineering, 2006. CSMR 2006.*, March 2006, pp. 56–67.

[17] P. Brebner, "Service-oriented performance modeling the MULE enterprise service bus (ESB) loan broker application," in *Software Engineering and Advanced Applications*, 2009, pp. 404–411.

[18] P. C. Brebner, "Performance modeling for service oriented architectures," in *Companion of the 30th international conference on Software engineering*. New York, NY, USA: ACM, 2008, pp. 953–954.

[19] P. Brebner, L. O'Brien, and J. Gray, "Performance modeling for e-government service oriented architectures (SOAs)," in *ASWEC*, S. R. Ashley Aitken, Ed. Australia: ACS, March 2008, pp. 130–138.

[20] S. Narayanan and S. A. McIlraith, "Simulation, verification and automated composition of web services," in *WWW '02: Proceedings of the 11th international conference on World Wide Web*. New York, NY, USA: ACM, 2002, pp. 77–88.

[21] S. Narayanan, "Reasoning about actions in narrative understanding," in *IJCAI'99: Proceedings of the 16th international joint conference on Artifical intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 350–355.

[22] L. Juszczyk and S. Dustdar, "Script-based generation of dynamic testbeds for soa," in *Web Services (ICWS), 2010 IEEE International Conference on*, july 2010, pp. 195 –202.

[23] M. Smit, A. Nisbet, E. Stroulia, A. Edgar, G. Iszlai, and M. Litoiu, "Capacity planning for service-oriented architectures," in *CASCON '08: Proceedings of the 2008 conference of the Center for Advanced Studies on collaborative research*. New York, NY, USA: ACM, 2008, pp. 144–156.

[24] A. Sulistio, C. Yeo, and R. Buyya, "A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools," *Software-Practice and Experience*, Jan 2004.

[25] R. David and H. Alla, *Discrete, Continuous, and Hybrid Petri Nets*. Springer, 2010.

[26] H. Alla and R. David, "Continuous and hybrid petri nets," *Journal of Circuits Systems Computers*, vol. 8, no. 1, pp. 159–88, February 1998.

[27] M. Smit, A. Nisbet, E. Stroulia, G. Iszlai, and A. Edgar, "Toward a simulation-generated knowledge base of service performance," *International Workshop on Middleware for Service-oriented Computing*, 2009.

[28] M. Smit and E. Stroulia, "Exploring simulation-based configuration decisions," in *International Conference on Service-Oriented Computing*, vol. 6470. Springer Berlin-Heidelberg, 2010, pp. 684–685.

Michael Smit is a province of Ontario postdoctoral fellow at York University in Toronto, Canada, and a member of the Adaptive Systems Research Lab led by Marin Litoiu. He completed his PhD with Eleni Stroulia at the University of Alberta in 2011. His research interests include autonomic/adaptive computing, cloud technologies, and service-oriented applications.

Eleni Stroulia is a Professor and NSERC/AITF Industrial Research Chair on Service Systems Management (w. support from IBM) with the Department of Computing Science at the University of Alberta. Her research addresses industrially relevant software-engineering problems with automated methods, based on artificial-intelligence techniques. Her team has produced automated methods for migrating legacy interfaces to web-based front ends, and for analyzing and supporting the design evolution of object-oriented software. She has more recently been working on the development, composition, run-time monitoring and adaptation of service-oriented applications, and on examining the role of web 2.0 tools and virtual worlds for innovative service delivery.