12th ICCRTS

"Adapting C2 to the 21st Century"

Modeling Service Oriented Architectures in a Command and Control Application Context

Topics: Networks and Networking, Modeling and Simulation, Network-Centric Experimentation and Applications, C2 Technologies and Systems

James Solderitsch

Gestalt, LLC

680 American Ave.

Suite 302

King of Prussia, PA 19406

610-994-2860

jsolderitsch@gestalt-llc.com

Abstract

Service Oriented Architectures (SOAs) hold promise for use in Command and Control (C2) application domains and contexts but this promise remains essentially un-tested and un-validated. This un-tested and largely un-realized promise is particularly true in the context of systems on the Edge where system connectivity is problematic. Gestalt and Villanova University are partners in the Applied Research for Computing Enterprise Services (ARCES) program under sponsorship of the Air Force Electronic Systems Group (ELSG/KI) at Hanscom Air Force Base. ARCES has spent significant effort in trying to understand aspects and features of both SOAs and their applications to C2. As part of this effort, we developed executable models to help evaluate SOA-based approaches and architectures. This paper presents ARCES' findings to-date, introduces several SOA models, discusses how these models are verified and validated, and shows the direct application of these models to investigating the utility and practicality of SOA solutions to C2 problems.

Classification

The material in this paper is unclassified.

1 Introduction

Service Oriented Architectures (SOAs) hold promise for use in Command and Control (C2) application domains and contexts but this promise remains essentially un-tested and un-validated. This un-tested and un-tried promise is particularly true in the context of systems on the Edge where system and participant connectivity is problematic.

Gestalt and Villanova University are partners in the Applied Research for Computing Enterprise Services (ARCES) program under sponsorship of the Air Force Electronic Systems Group (ELSG/KI) at Hanscom Air Force Base. We (ARCES) have expended significant effort in trying to understand aspects and features of both SOAs and their applications to C2. To that end, we developed executable models to help evaluate SOA-based approaches and architectures. One important modeling technology we employ is called MESA: Modeling Environment for SOA Analysis. The Mitre Corporation created MESA under DISA (Defense Information Systems Agency) sponsorship. MESA allows the creation of executable architectural models through which the stressful conditions, under which C2 applications (realized in a SOA) are expected to be deployed and operate, can be examined and experimented with.

This paper presents some of the research work that we have performed and discusses several models and laboratory experiments conducted to validate the models. In these models, ARCES attempts to define as-is and to-be architectures and their interplay in a large-scale communication network that we call a C2 fabric. The fragileness and unpredictable nature of this fabric is of particular importance and modeling constraints such as low bandwidth and intermittent communication links is an important feature of these models. The use of compression technologies to overcome some of these constraints appears to be a fruitful avenue of research – in this paper we show how our models support this claim.

Along with modeling, ARCES is also concerned with evaluating and experimenting with particular ESB (Enterprise Service Bus) implementations ranging from commercial tools such as BEA's AquaLogic to open source technologies such as the Apache ServiceMix project. In fact, using these technologies we built a laboratory test bed in which the ideas and approaches incorporated in the SOA models can be demonstrated and used to verify and validate the models themselves. At the same time, the suitability and maturity of the ESB products are examined to see how well they are equipped to support C2 SOA needs and concerns.

Recently, ARCES has started to examine the difficult problem of service discovery from a Net-Centric Warfare viewpoint and we are beginning to develop models to help understand and document this problem and potentially outline solutions whose effectiveness can be demonstrated using our models.

2 Overview of MESA Technology

As mentioned in the introduction, the SOA models created by ARCES are based on MESA. MESA, developed by the Mitre Corporation, is primarily a set of library components for a commercial discrete event simulation tool called Extend. MESA requires the Industry version of Extend because it utilizes a database approach to capture information about computer and network resources. In order to execute MESA models, you must first acquire the free Extend Player tool and install the MESA libraries. The ARCES project has interacted with Patrick Van Metre at Mitre who was a primary developer of MESA to obtain the MESA libraries and to suggest some extensions to MESA functionality.

A screen snapshot of our Compression ESB Fabric model running in the Extend application is shown in Figure 1.

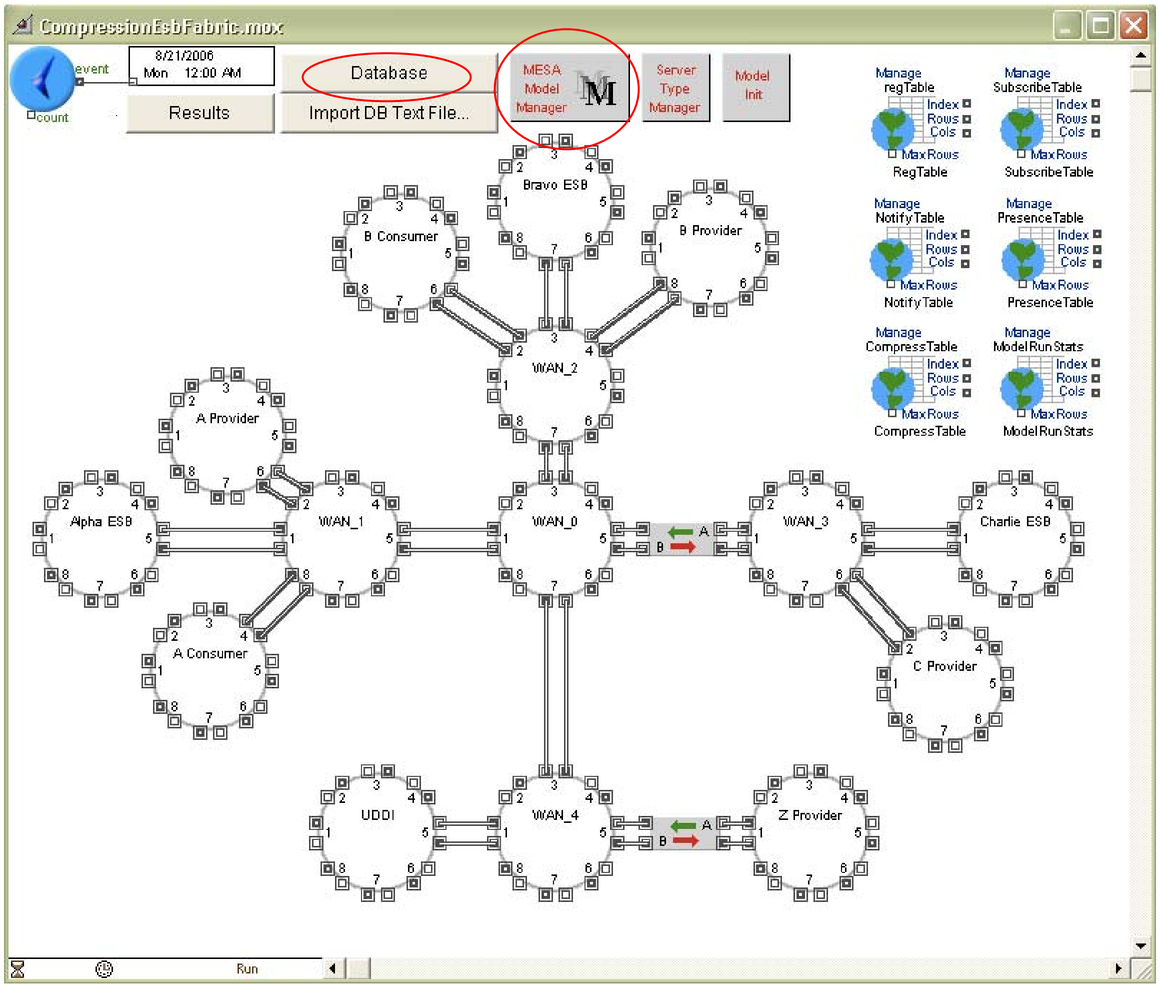


Figure 1. MESA Model in Extend

Each round node shown in Figure 1 represents a computational resource (most often a computer or network component) on which programs can execute and through which messages (requests and responses) can be sent. Web services in a Service Oriented Architecture (SOA) are instances of such programs. With MESA it is possible to define rather complex logical behavior that models the actions of web services including the invocation of one service by another. Web services can be mapped to individual nodes in the model where their execution will be simulated. Each node is defined with computational characteristics including CPU horsepower and number of resident processors so that execution statistics can be gathered and measured. Connections between nodes are also modeled as shown by the rectangular box between the WAN_4 and Z Provider nodes in Figure 1. These connections or links can have characteristics of their own that define their behavior including properties such as link speed (in bits per second: bps), link latency and link background utilization. In the case where nodes are directly connected to one another, there is no explicit link behavior imposed on the connection – MESA manages network traffic using default behaviors built into MESA.

It is beyond the scope of this design document to fully explain the usage of MESA. A MESA Users Guide is included in the MESA distribution that describes how to use MESA and understand MESA models. The process of creating a MESA model conceptually consists of the following steps:

- Define node names
- Define service names
- Create model layout from MESA node library
- Define service logic definitions

Some of these steps are facilitated by MESA user interface extensions to Extend and some of them are performed using Extend model creation actions. For example, double clicking the MESA Model Manager button as shown in Figure 1 brings up the dialog panel shown in each half of Figure 2

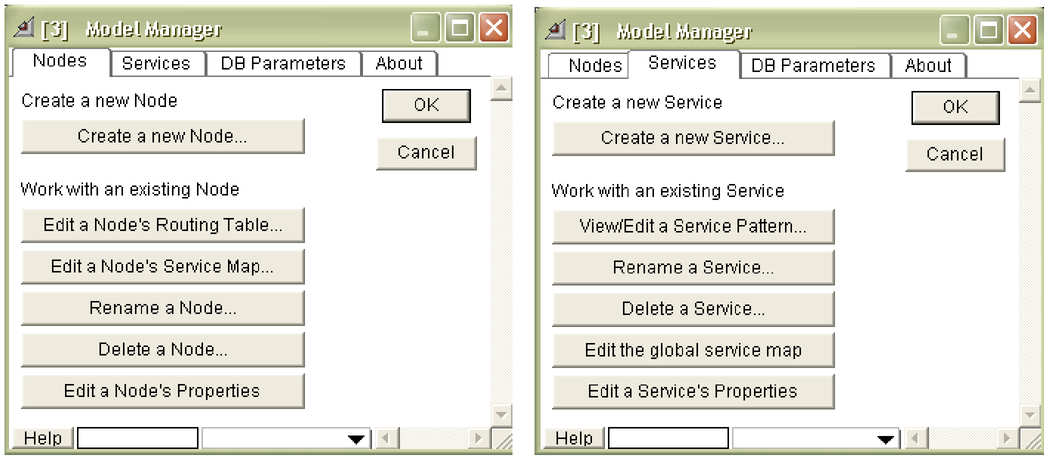
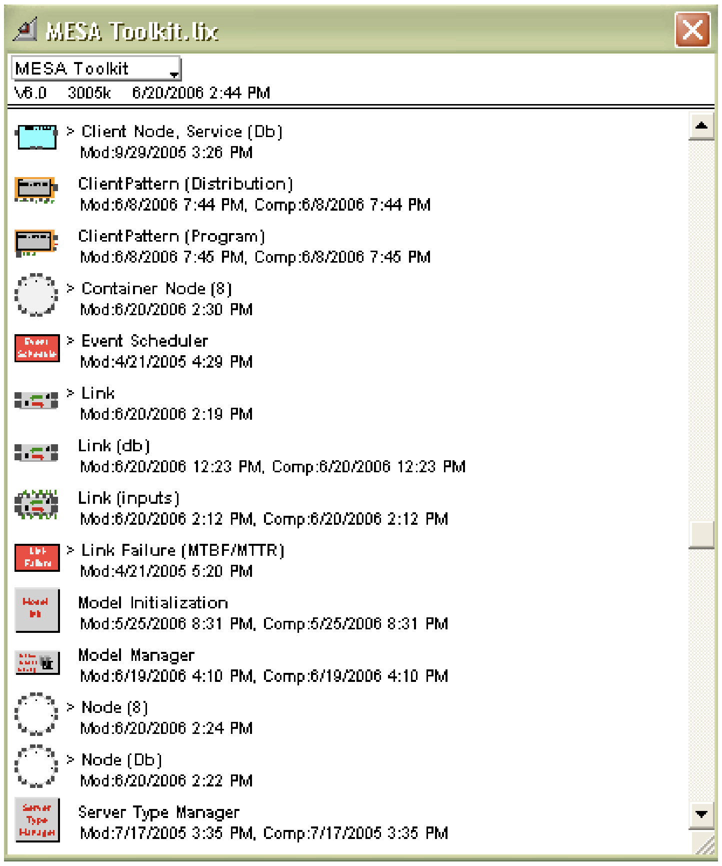


Figure 2. Node and Service Editing Panels

From this panel, model nodes are named by clicking the Create a New Node… button and completing the subsequent dialog. Similarly, you begin to define the set of services to be modeled by first clicking the Services tab in the right side of Figure 2 to bring up the panel shown in the left side of the figure and then clicking the Create a new Service… button and completing the subsequent dialog for each service to include in the model.

You define MESA nodes and their interconnections by choosing items from the MESA ToolKit library shown in Figure 3 and then linking them together by dragging the mouse from one item’s input to another’s output. Each connection is shown as having two elements to capture both incoming and outgoing message flow. In particular, the model shown in Figure 1 is primarily composed of the Link (DB) and Node (DB) items found in the MESA Toolkit library. MESA supports DB items with properties defined in database tables. A database inspector is available for viewing and editing these properties – you double click the Database button in Figure 1 to see this inspector.



MESA Toolkit.lix

MESA Toolkit

V6.0 3005k 6/20/2006 2:44 PM

> Client Node, Service (Db)
Mod:9/29/2005 3:26 PM

ClientPattern (Distribution)
Mod:6/8/2006 7:44 PM, Comp:6/8/2006 7:44 PM

ClientPattern (Program)
Mod:6/8/2006 7:45 PM, Comp:6/8/2006 7:45 PM

> Container Node (8)
Mod:6/20/2006 2:30 PM

> Event Scheduler
Mod:4/21/2005 4:29 PM

> Link
Mod:6/20/2006 2:19 PM

Link (db)
Mod:6/20/2006 12:23 PM, Comp:6/20/2006 12:23 PM

Link (inputs)
Mod:6/20/2006 2:12 PM, Comp:6/20/2006 2:12 PM

> Link Failure (MTBF/MTTR)
Mod:4/21/2005 5:20 PM

Model Initialization
Mod:5/25/2006 8:31 PM, Comp:5/25/2006 8:31 PM

Model Manager
Mod:6/19/2006 4:10 PM, Comp:6/19/2006 4:10 PM

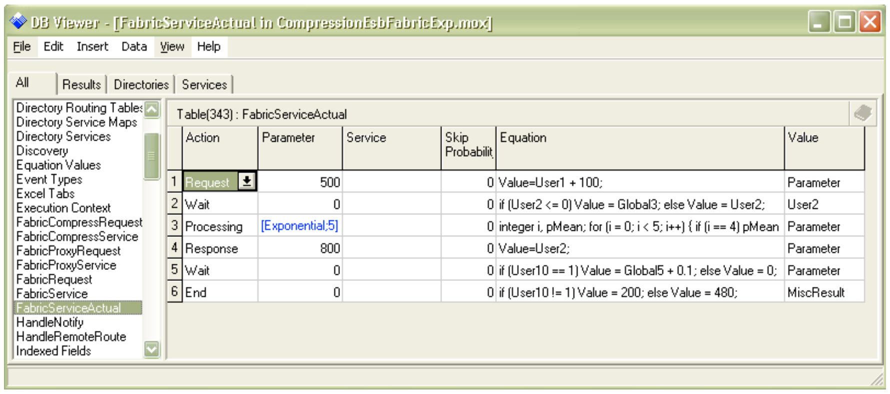
> Node (8)
Mod:6/20/2006 2:24 PM

> Node (Db)
Mod:6/20/2006 2:22 PM

Server Type Manager
Mod:7/17/2005 3:35 PM, Comp:7/17/2005 3:35 PM

Figure 3. MESA Toolkit Library

Each named service is fully defined in its own MESA database table. For example: Figure 4 shows the table defining the FabricServiceActual service that models the provider’s implemented service. A service in MESA is composed of a number of Action steps – FabricServiceActual is defined in 6 steps. Each action step may involve reference to a number of values obtained from the other columns in the service table. For example, the Value computed in the Equation column of step 1 results in the request size parameter being set to this value (User1 + 100). In step 3, the Value of the equation (logic partially hidden) is used to set the processing delay. Equation column values allow computations to be defined and Value column values allow computation results to be used either in the current step or in other steps.



Table(343) : FabricServiceActual

	Action	Parameter	Service	Skip Probabilit	Equation	Value
1	Request	500		0	Value=User1 + 100;	Parameter
2	Wait	0		0	if (User2 <= 0) Value = Global3; else Value = User2;	User2
3	Processing	[Exponential;5]		0	integer i, pMean; for (i = 0; i < 5; i++) { if (i == 4) pMean	Parameter
4	Response	800		0	Value=User2;	Parameter
5	Wait	0		0	if (User10 == 1) Value = Global5 + 0.1; else Value = 0;	Parameter
6	End	0		0	if (User10 != 1) Value = 200; else Value = 480;	MiscResult

Figure 4. Service Logic Definition

Additional information regarding the structure and usage of MESA is provided in subsequent sections of this document.

3 Model Validation

We begin this section with some definitions of model validation from the literature. In particular, we consider these quotes from the paper by Robert Sargent [1].

- "Model validation is usually defined to mean "substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model" (Schlesinger et al. 1979 [2])."
- "A related topic is model credibility. Model credibility is concerned with developing in (potential) users the confidence they require in order to use a model and in the information derived from that model."
- "A model should be developed for a specific purpose (or application) and its validity determined with respect to that purpose. If the purpose of a model is to answer a variety of questions, the validity of the model needs to be determined with respect to each question."
- "A model is considered valid for a set of experimental conditions if the model's accuracy is within its acceptable range, which is the amount of accuracy required for the model's intended purpose. The amount of accuracy required should be specified prior to starting the development of the model or very early in the model development process."
- "The substantiation that a model is valid, i.e., performing model verification and validation, is generally considered to be a process and is usually part of the model development process."

As software engineers we find it useful to consider familiar software quality assurance (SQA) practices and compare them to similar practices for ensuring model quality. SQA practices include:

- Unit Testing
- Integration Testing (Better: Automated Continuous Integration)
- Acceptance Testing (by third party?)
- Usability/Behavioral Analysis Performed
- A test-driven (preferably automated) software development process – this is becoming an accepted software development best-practice

What are equivalent indicators of Model Quality Assurance practices? Is validation analogous to acceptance testing or integration testing? Can we describe a pattern for continuous model validation? Can we make the analogy that "tests are to software as use cases are to validation"? Can we define a model development process that captures the spirit and intent of test-driven software development?

As the quotations from the Sargent paper indicate, models ought to be created to answer specific questions. Before considering this further, we identify two general purposes for models:

- Prediction
- Exploration

Models can be built to predict the behavior of the system being modeled or to explore the behavior of the system being modeled. The general purpose of a model will likely affect its construction. However, keep in mind that exploration and prediction are not exclusive. Exploratory models can evolve into predictive models. Predictive models can be used to explore the systems being modeled. In any case, it is helpful to know what is the main purpose of a model before its construction begins.

Validation criteria for each kind of model are likely to be different. How well can the model user explore a system? How well, and how accurately, can the model user predict the behavior of a system? Questions leading to the construction of each kind of model will be different.

The questions that a model is meant to answer should be identified before building the model. Questions should be clearly understood by model builders, model validators and model customers (stakeholders). Of course, questions can be refined and the set of questions expanded, over time. New and refined questions should be the primary drivers for model evolution. Questions typically reflect whether the primary model purpose is prediction or exploration.

Validation should be understood in relation to the identified questions. Moreover, the questions can be used to derive model acceptance criteria. If acceptance criteria are well defined and “testable”, validation can be as simple as checking whether a model satisfies stated acceptance criteria. Models will either “pass” or “fail” with respect to such criteria.

Fundamentally, what are model acceptance criteria? Acceptance criteria are related to, but not the same as, requirements. Acceptance criteria should be related to the questions that the model is supposed to answer. What kinds of answers are expected? What are the expected qualities of the answers? Acceptance criteria can be quantitative or qualitative. Quantitative acceptance criteria include performance measures and benchmark results. They allow the model to be compared to reality in a numeric way. Qualitative acceptance criteria focus on observing and documenting behaviors and specific results.

One possibility is for the acceptance criteria to be expressed as use cases that express intents and effects of one or more requirements. Model requirements in turn are developed based on the set of questions. And a question can be treated as a mega-requirement (overarching requirement) that inspires the creation of specific requirements that collectively allow a determination that the model is able to answer the question.

Use cases are based on questions and requirements and are meant to determine how well the model matches the system being modeled. One point of view on validation is to say that a model is valid if it supports the representation of a set of use cases that come from the real world – what the model is meant to capture. This support includes an accurate reflection of the use cases through model components that represent identifiable and recognizable pieces of the real world. Thus, when one use case is a simple modification of another, and the expectation exists that the outcome of this use case is an expected variation of the outcome of the first use case, the model should reflect this modified outcome.

Thus, a valid model is one that takes "near things to near things" (this phrase is due to a favorite Math professor of the author) where nearness is understood in the real world context. If we modify a use case, and observe a modified outcome in the real world, it should be the case that the model's reflection of the modified use case has an expectedly modified outcome. The model is predictive in this rather weak sense. It conforms to real world observations.

One caveat would be that since the model is an abstract representation, rather than a parallel implementation, we always have the potential for use case outcomes in a model that diverge from the real world. And as we've seen in the past, sometimes such divergence is a bug in our model, sometimes it is a bug in the real world implementation, and sometimes it is neither (or both).

A related observation is that when use case modifications exhibit large shifts in observed real world behavior (discontinuities of behavior – singularities to steal another math term), the model's reflected use case modifications should ALSO exhibit these large shifts in modeled behavior. So if there is a use case on which the real world system exhibits a breakdown (or tipping point to use another current buzzword), we would expect the model's handling of the use case to also exhibit a breakdown. But once again, a singularity in the real implementation or in the model, might signal a bug in either the real system or the model of that system.

One kind of use case validation is to measure the length of time it takes to complete a task or process in the real world and then compare the corresponding time reported by the model. In particular for SOAs, Round Trip Time (RTT) is often used as a performance measure: how long it takes for a request initiated by a service consumer to result in a response received from a service provider. This point-of-view includes the expectation of measuring service request and response RTTs in the context of a range of experimental conditions such as varying numbers of simultaneous requests, large message sizes, or other increasingly stressful operational conditions.

So, if the reality is that there is a "gradual rise" of RTT as the test bed tries to generate 10, 20, … N simultaneous requests in a real world implementation, we expect that the model should also show a gradual rise. It is not necessary that the SAME rise be seen. Nor is it necessary to tinker with the model to get the same rise – getting an expected rising behavior might be enough. This might be enough to claim validity for this aspect of the models behavior.

As shown below, for some models and experimental conditions we have been able to quantify whether measurements (specifically of RTT) made using a model of a real world situation statistically match measurements taken under real world experimental conditions. As such, we can quantify the validity of the model with respect to these measurements.

Validity is NOT a condition but a spectrum of conditions where we as model builders, and others as model stakeholders, have to agree on where in that spectrum we expect the model to be as a reflection of reality. It's up to us to look at some section of this spectrum and agree to try to validate our models within this spectrum.

We believe it is useful to concentrate on use cases to drive our work on model validation. We expect to see correct use case behaviors for the most part. And, if in the real world, varying use cases in a controlled way leads to observed outcomes, then in the model world, varying the model's reflection of those use cases should lead to modeled outcomes that conform to the real world outcomes.

4 Important C2 Domain Constraints and Concerns

While the discussion of models and model validation that is presented in this paper remains at an abstract, general level of applicability, the ARCES mission is to look for problems and situations that are of direct and immediate relevance to C2, and in particular C2 on the edge. In the area of SOA and network based systems there are two main concerns that ARCES wishes to address:

- Dynamic presence of consumer, provider and ESB nodes;
- Fragile and constrained network connectivity within the nodal fabric of nodes.

Computer systems and the services they provide are expected to come and go. Their availability and reliability cannot be relied upon. It will be best if there are multiple service providers deployed whose ability to provide information in response to consumer requests can seamlessly be incorporated in the fabric so that the “best fit” provider can be contacted to respond to a consumer request. What is best will depend on operational circumstances such as the network bandwidth available between consumer and provider, the workload currently being handled by the provider, and perhaps other contextual pieces of information.

We believe that the services and architecture of ESBs can be of direct benefit in enabling an effective edge operational scenario. The ESB can help support a dynamic population of consumers and providers that can discover and interact with one another. As each provider plugs into the bus, other current members of the bus are informed of the presence and service offerings available within the provider. When a provider un-plugs from the bus, the removal of its available services is communicated.

We imagine a dynamic network with multiple ESB-based enclaves that share information about each other in a federated manner. To support this federation, a Google-like indexing mechanism is available to allow consumers within one enclave to discover available service providers in another enclave. Each ESB is aware of its own network context and the network conditions that apply between different enclaves and routes consumer requests to the provider best able to provide the needed information in a timely manner. As necessary, messages to be exchanged are compressed and decompressed automatically on either side of a constrained network between ESBs.

ARCES plans to address many of the problems and opportunities that this imagined view suggests. So far, we have concentrated on interconnecting a set of ESBs, each of which supports multiple consumers and providers. Our detailed modeling to-date has focused on the effective use of compression technologies when constrained network conditions are known to apply. Recently ARCES has begun to model the general discovery problem and how best to support discovery under edge conditions. A preliminary conceptual model under development is described in section 7.

5 ARCES Compression ESB Fabric Model

Figure 5 (a copy of Figure 1) illustrates the model topology for the Compression ESB Fabric model we built to explore some SOA behaviors and the ability to improve component and message flow performance. The model shows families of consumer, provider and ESB nodes clustered in a fabric. A consumer node is the source of information (service) requests that are ultimately handled by a provider node. An ESB node will act as a broker/proxy for a consumer to help it locate a provider of a desired service and route the consumer’s request to an available provider. In addition, the model includes a number of WAN (Wide Area Network) nodes that represent the multiple network hops that are part of the fabric. The model includes a MESA link object that separates the “A” and “B” families of nodes from the “C” family (the C family does not include a consumer in the current version).

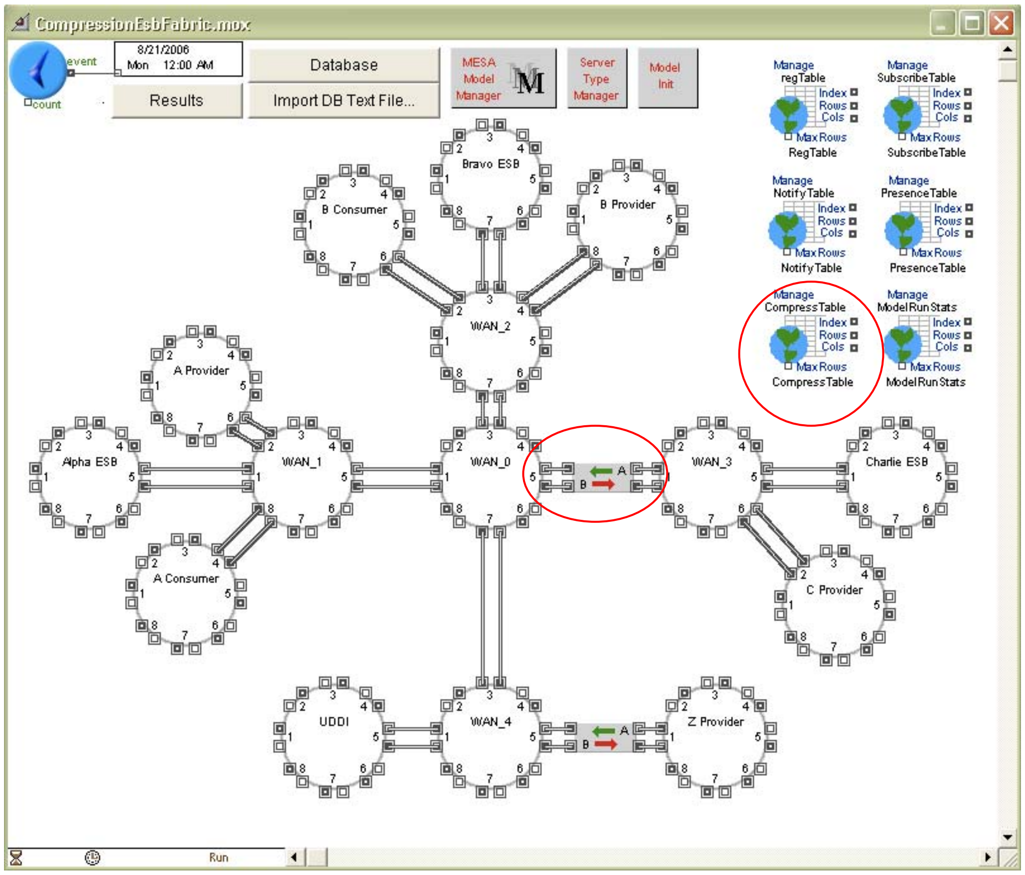
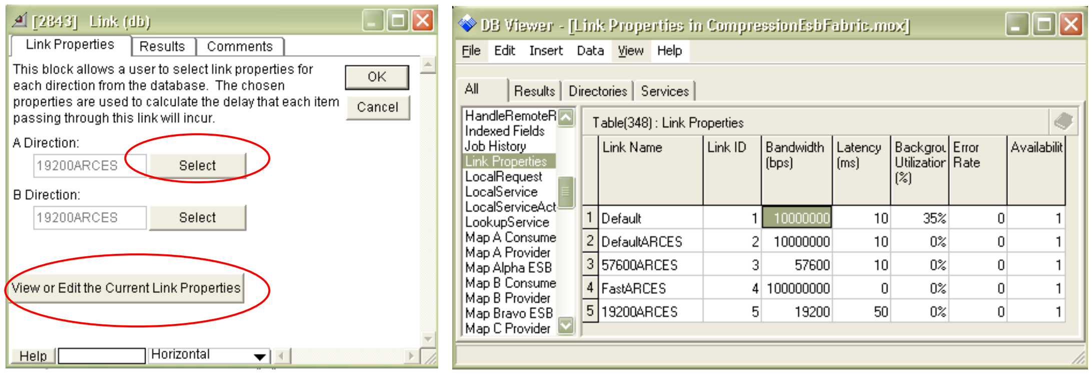


Figure 5. Compression ESB Fabric Model

Double clicking the link object between WAN_0 and WAN_3 produces the dialog box shown in the left side of Figure 6. Clicking the Select Button can set link characteristics for each direction. As shown, the properties of each link are set to be 19200ARCES. Clicking the View or Edit Current Link properties opens the model database view of the Link Properties table as shown in the right side of the figure.



This block allows a user to select link properties for each direction from the database. The chosen properties are used to calculate the delay that each item passing through this link will incur.

A Direction: 19200ARCES

B Direction: 19200ARCES

Table(348) : Link Properties

	Link Name	Link ID	Bandwidth (bps)	Latency (ms)	Backgrou Utilizatior (%)	Error Rate	Availabilit
1	Default	1	10000000	10	35%	0	1
2	DefaultARCES	2	10000000	10	0%	0	1
3	57600ARCES	3	57600	10	0%	0	1
4	FastARCES	4	100000000	0	0%	0	1
5	19200ARCES	5	19200	50	0%	0	1

Figure 6. Link Adjustments and Properties

You can make changes to the link properties of each of the named links by editing them in this table view. Of particular interest are the settings for the Bandwidth of the link and the latency of the link. This link object allows the model to support experimentation with constrained network conditions in the fabric. One approach for handling messages, including service requests and responses, over a constrained link is to apply compression to the message before it is sent. One of the main goals of the model is to investigate under what conditions compression is meaningful and efficient with respect to the goal of improving message transfer over the link.

MESA uses databases to store essential SOA properties but MESA service action patterns are not able to read and write to database tables during model execution. Instead, the global array feature of the Extend product is used for model run-time information that needs to be shared across the blocks in the model. These arrays are indicated in Figure 5 by the globe icons in the upper right of the display. Compression characteristics are stored in the Compress Table global array. This table is actually initialized from an Excel worksheet that can be cut-and-pasted into MESA. Sample contents for such a worksheet are illustrated in Figure 7.

midpoint	compressed %	time to compress	time to decompress	Provider Mean Delay	Proxy Mean Delay	Pass Through Mean Delay	Compress Proxy Mean Delay	Compress Pass Through Mean Delay
500	0.5	62	81	4	160	160	15	15
5000	0.27	40	40	4	150	150	15	15
50000	0.15	60	60	100	2500	2500	560	560
175000	0.11	150	150	250	8000	8000	1000	1000
1000000	0.24	2000	2000	2000	190000	190000	55000	55000

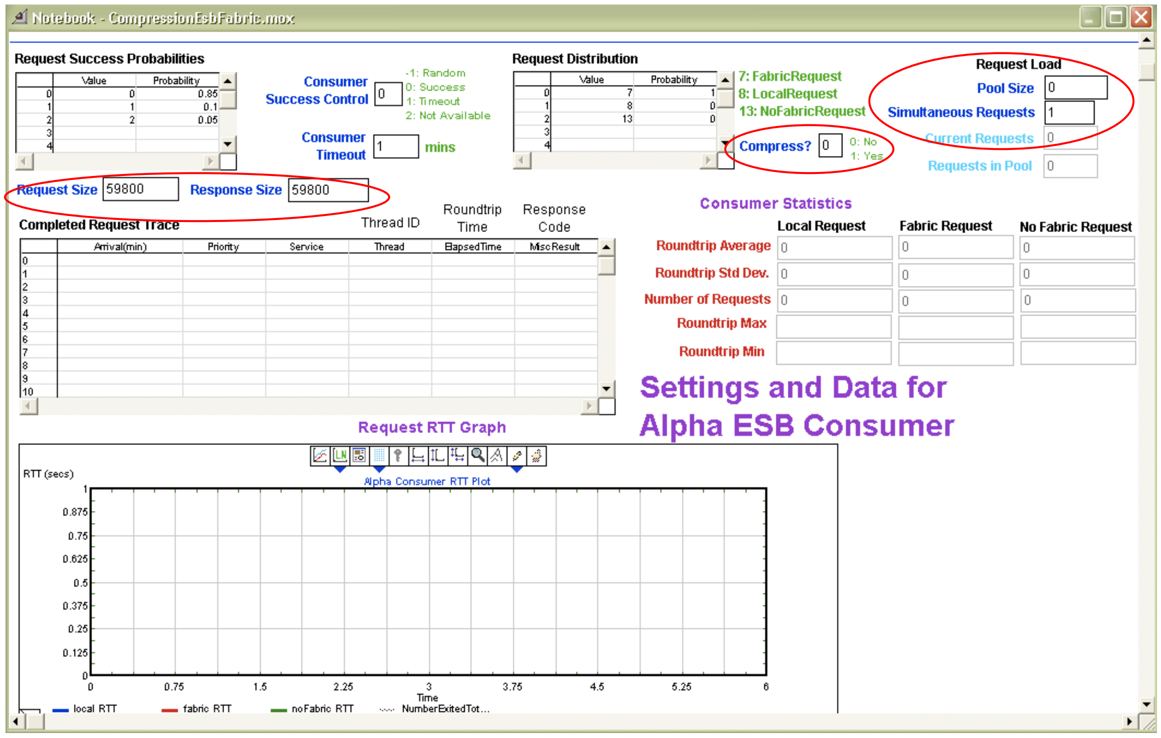
Figure 7. Compression Table Statistics

There are 5 rows and 9 columns in this table. Column 1 is a midpoint value (in bytes) for a specific message size range. For example, 500 is the midpoint between 0 and 1000 bytes, 5000 is the midpoint between 0 and 10000 bytes, etc. Each midpoint is used in a compression calculation within the model. The second column is a percentage reduction achieved by compression for messages with a size in the range designated by the midpoint. For example, messages with sizes above 10000 bytes but below 100000 bytes will have a compressed size of 0.15 x the message size. The next two columns represent the processing effort to perform compression and decompression for messages with sizes in the range specified by the midpoint. Each number is understood to be MESA Benchmark Units (BU's) value. For a MESA node identified as a Default Server, which has a benchmark rating of 1000 Benchmark Units Per Second (BUPS), these numbers translate to milliseconds of processor time required to complete the corresponding effect. For example, for a message of size 50000 bytes being compressed on a standard MESA processor, it takes 60 milliseconds to compress the message and 60 milliseconds to decompress it.

The other five columns represent processing delays that are implemented in the model. Each is an average (mean) delay and there are some probabilistic effects that vary the actual values applied on a per message basis. There is a delay for the provider, and a pair of proxy and pass-through delays corresponding to the effort spent by the ESB node on each side of the constrained link to process the message as it is forwarded from consumer to provider and back again. When compression is applied, there is one pair of delays, identified as the Compress Delays in Figure 7 in the last two columns, which apply when the message is being compressed and decompressed on each side of the link. The other delay values apply when the message is not being compressed. Since transmission over the link is expected to be significantly higher when there is no compression, the compress delays are much smaller than the non-compression delays.

The compression statistics represented in this table are taken from actual compression measurements compiled by a member of the ARCES team. The delays values are based on measurements made in the lab.

The next two figures: Figure 8 and Figure 9 show a user interface, implemented using Extend, to allow the user to select model configuration items and observe model run-time results The user interface elements for the benchmarking consumer associated with the Alpha ESB and the distribution-based consumer associated with the Bravo ESB are grouped together – there is a thin blue line marking where each portion begins. The same statistics-reporting features support each consumer: tables, statistical summaries and plots. More importantly, each consumer’s processing of requests and responses is now affected by request and response size settings as well as whether or not compression is applied to messages as they are processed within the fabric. We will see examples of how changes to these settings affect message flow below. The main difference between each section of the model is the means by which requests are generated for nodes in the model to process. The Alpha consumer is aimed at exploring request loads – numbers of simultaneous requests – whereas the Bravo consumer injects requests based on inter-arrival time distributions.



Notebook - CompressionEsbFabric.mox

Request Success Probabilities

	Value	Probability
0	0	0.85
1	1	0.1
2	2	0.05
3		
4		

Consumer Success Control: 0 (-1: Random; 0: Success; 1: Timeout; 2: Not Available)

Consumer Timeout: 1 mins

Request Distribution

	Value	Probability
0	7	1
1	8	0
2	13	0
3		
4		

7: FabricRequest
8: LocalRequest
13: NoFabricRequest

Compress? 0 (0: No; 1: Yes)

Request Load
Pool Size: 0
Simultaneous Requests: 1
Current Requests: 0
Requests in Pool: 0

Request Size: 59800 Response Size: 59800

Completed Request Trace (Thread ID, Roundtrip Time, Response Code)

	Arrival(min)	Priority	Service	Thread	ElapsedTime	MiscResult
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

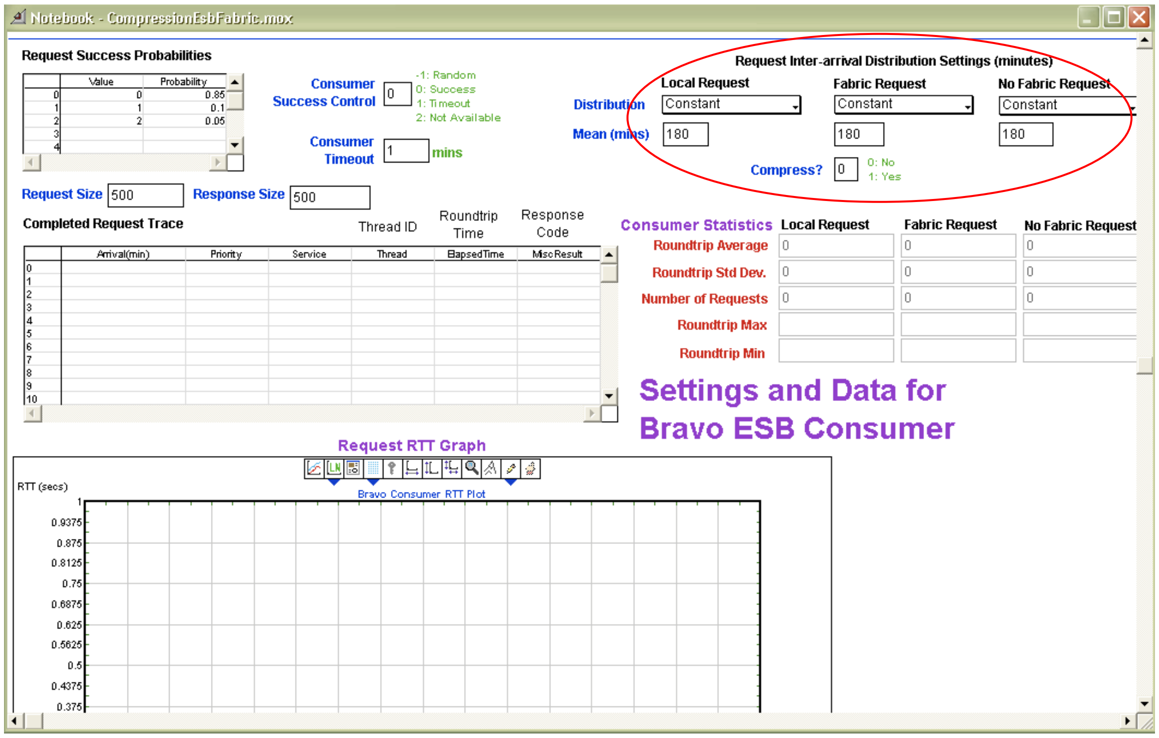
Consumer Statistics

	Local Request	Fabric Request	No Fabric Request
Roundtrip Average	0	0	0
Roundtrip Std Dev.	0	0	0
Number of Requests	0	0	0
Roundtrip Max			
Roundtrip Min			

Settings and Data for Alpha ESB Consumer

Figure 8. Model Notebook for Benchmark Consumer

The ovals on each figure show where the model user can set up important aspects of the model prior to execution. You can define request and response sizes, whether a message is compressed as part of its processing, and either how many simultaneous requests should be generated (Figure 8) or how requests arrive for handing by the model (Figure 9).



Request Success Probabilities

	Value	Probability
0	0	0.85
1	1	0.1
2	2	0.05
3		
4		

Consumer Success Control: 0 (-1: Random, 0: Success, 1: Timeout, 2: Not Available)

Consumer Timeout: 1 mins

Request Size: 500 Response Size: 500

Request Inter-arrival Distribution Settings (minutes)

	Local Request	Fabric Request	No Fabric Request
Distribution	Constant	Constant	Constant
Mean (mins)	180	180	180

Compress? 0 (0: No, 1: Yes)

Completed Request Trace

	Arrival(min)	Priority	Service	Thread	ElapsedTime	MiscResult
				Thread ID	Roundtrip Time	Response Code
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

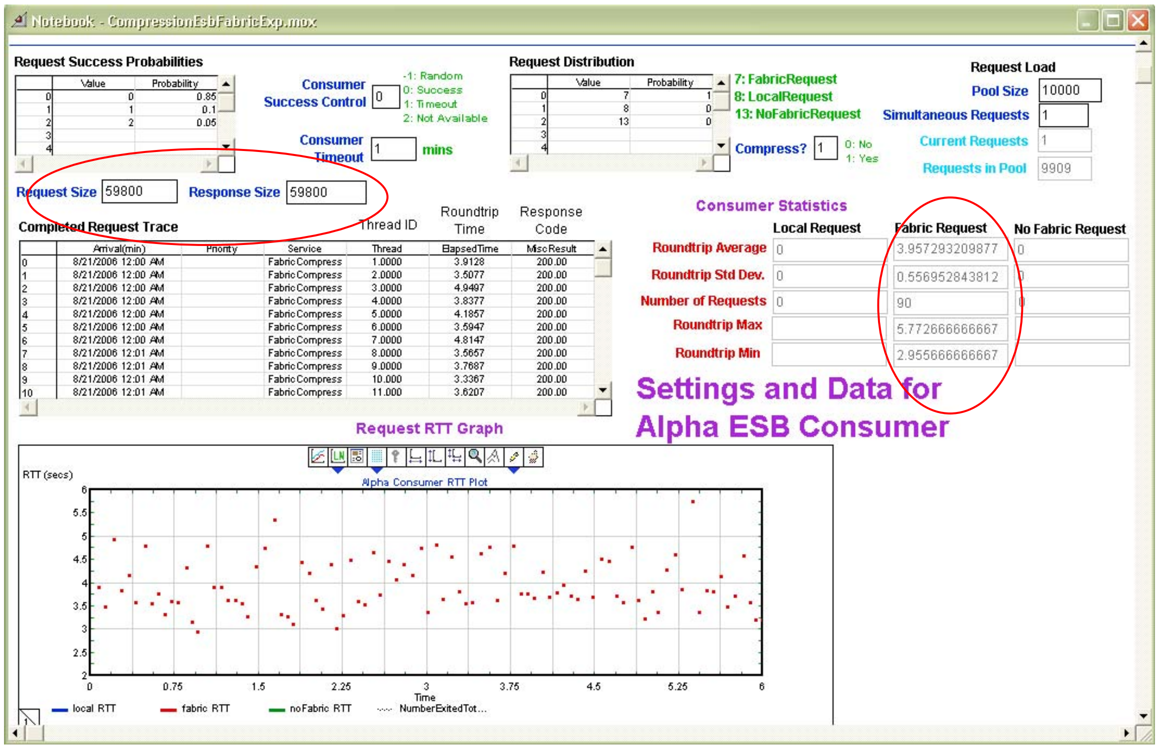
Consumer Statistics	Local Request	Fabric Request	No Fabric Request
Roundtrip Average	0	0	0
Roundtrip Std Dev.	0	0	0
Number of Requests	0	0	0
Roundtrip Max			
Roundtrip Min			

Settings and Data for Bravo ESB Consumer

Figure 9. Model Notebook for Distribution Consumer

The next several figures will document 4 separate model runs for the AquaLogic calibrated model for a sample request and response message size (59800 bytes – this corresponds to an actual file used in the lab environment for testing AquaLogic and the Apache ServiceMix ESB product), with and without compression applied to the messages, and with different Link object settings, one representing a low bandwidth connection (modem speed: 56.2 kilobits per second) and one representing a normal wired network connection. The consumer is modeled to generate a new request as soon as a response is received to the previous request – the number of simultaneous requests is set to 1.

Each of the model runs represent 6 minutes of actual AquaLogic model processing. The figure captions identify which set of model runtime conditions were used to generate the data shown in the figure. For example, Figure 10 documents a model run for the low bandwidth connection, where messages are compressed. Over the 6-minute run, 90 request/response roundtrips were processed with an average roundtrip time (RTT) of approximately 4 seconds. This figure can be compared to Figure 11 that documents a similar simulation except that compression is off. Under this constraint, only 15 roundtrips complete in the 6 minute run with an average RTT of approximately 23 seconds. Thus, end-to-end, compression results in an overall five-fold increase in performance compared to the no compression case over the runtime that the model simulates. Comparing how well compression aids message throughput in the lab installation of AquaLogic can then be used to validate such an increase in performance. This comparison is documented in the next section of the paper.



Request Success Probabilities

	Value	Probability
0	0	0.85
1	1	0.1
2	2	0.05
3		
4		

Consumer Success Control: 0 (-1: Random; 0: Success; 1: Timeout; 2: Not Available)

Consumer Timeout: 1 mins

Request Size: 59800 Response Size: 59800

Request Distribution

	Value	Probability
0	7	1
1	8	0
2	13	0
3		
4		

7: FabricRequest
8: LocalRequest
13: NoFabricRequest

Compress? 1 (0: No; 1: Yes)

Request Load
Pool Size: 10000
Simultaneous Requests: 1
Current Requests: 1
Requests in Pool: 9909

Completed Request Trace (Thread ID, Roundtrip Time, Response Code)

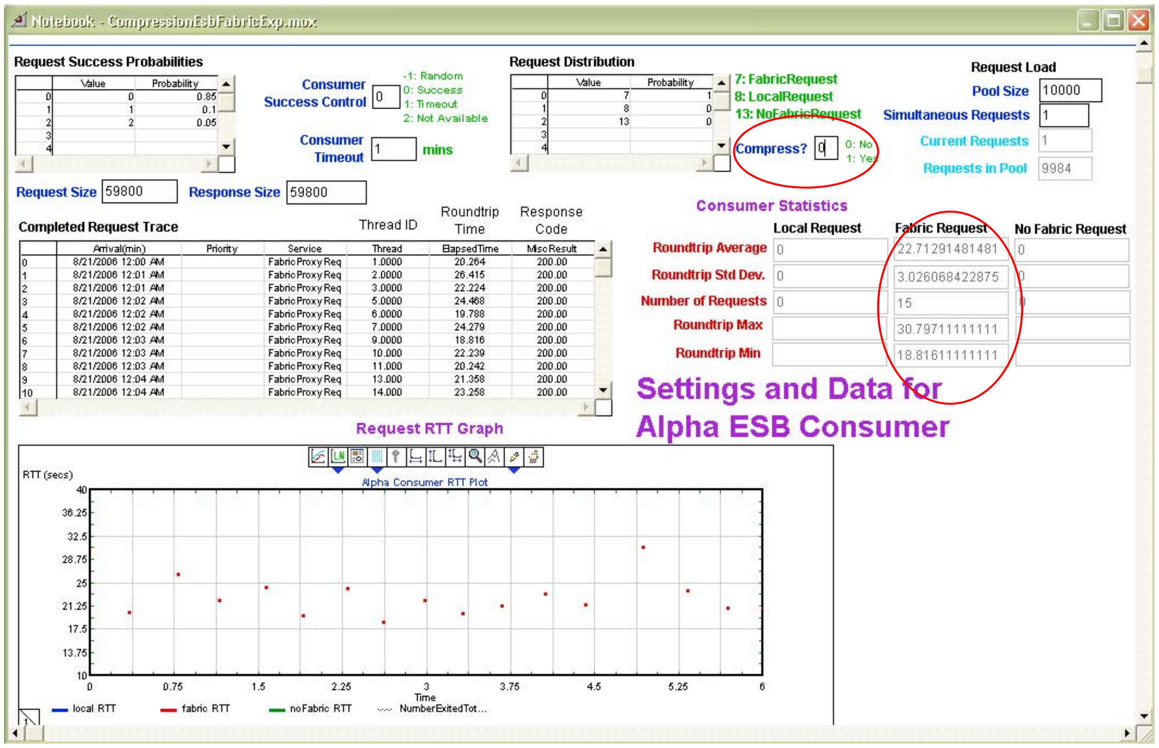
	Arrival(min)	Priority	Service	Thread	ElapsedTime	MiscResult
0	8/21/2006 12:00 AM		FabricCompress	1.0000	3.9128	200.00
1	8/21/2006 12:00 AM		FabricCompress	2.0000	3.5077	200.00
2	8/21/2006 12:00 AM		FabricCompress	3.0000	4.9497	200.00
3	8/21/2006 12:00 AM		FabricCompress	4.0000	3.8377	200.00
4	8/21/2006 12:00 AM		FabricCompress	5.0000	4.1857	200.00
5	8/21/2006 12:00 AM		FabricCompress	6.0000	3.5947	200.00
6	8/21/2006 12:00 AM		FabricCompress	7.0000	4.8147	200.00
7	8/21/2006 12:01 AM		FabricCompress	8.0000	3.5657	200.00
8	8/21/2006 12:01 AM		FabricCompress	9.0000	3.7687	200.00
9	8/21/2006 12:01 AM		FabricCompress	10.000	3.3367	200.00
10	8/21/2006 12:01 AM		FabricCompress	11.000	3.6207	200.00

Consumer Statistics

	Local Request	Fabric Request	No Fabric Request
Roundtrip Average	0	3.957293209877	0
Roundtrip Std Dev.	0	0.556952843812	0
Number of Requests	0	90	0
Roundtrip Max		5.772666666667	
Roundtrip Min		2.955666666667	

Settings and Data for Alpha ESB Consumer

Figure 10. AquaLogic Low Bandwidth With Compression



Request Success Probabilities

	Value	Probability
0	0	0.85
1	1	0.1
2	2	0.05
3		
4		

Consumer Success Control: 0 (-1: Random; 0: Success; 1: Timeout; 2: Not Available)

Consumer Timeout: 1 mins

Request Size: 59800 Response Size: 59800

Request Distribution

	Value	Probability
0	7	1
1	8	0
2	13	0
3		
4		

7: FabricRequest
8: LocalRequest
13: NoFabricRequest

Compress? 0 (0: No; 1: Yes)

Request Load
Pool Size: 10000
Simultaneous Requests: 1
Current Requests: 1
Requests in Pool: 9984

Completed Request Trace (Thread ID, Roundtrip Time, Response Code)

	Arrival(min)	Priority	Service	Thread	ElapsedTime	MiscResult
0	8/21/2006 12:00 AM		FabricProxyReq	1.0000	20.264	200.00
1	8/21/2006 12:01 AM		FabricProxyReq	2.0000	26.415	200.00
2	8/21/2006 12:01 AM		FabricProxyReq	3.0000	22.224	200.00
3	8/21/2006 12:02 AM		FabricProxyReq	5.0000	24.468	200.00
4	8/21/2006 12:02 AM		FabricProxyReq	6.0000	19.788	200.00
5	8/21/2006 12:02 AM		FabricProxyReq	7.0000	24.279	200.00
6	8/21/2006 12:03 AM		FabricProxyReq	9.0000	18.816	200.00
7	8/21/2006 12:03 AM		FabricProxyReq	10.000	22.239	200.00
8	8/21/2006 12:03 AM		FabricProxyReq	11.000	20.242	200.00
9	8/21/2006 12:04 AM		FabricProxyReq	13.000	21.358	200.00
10	8/21/2006 12:04 AM		FabricProxyReq	14.000	23.258	200.00

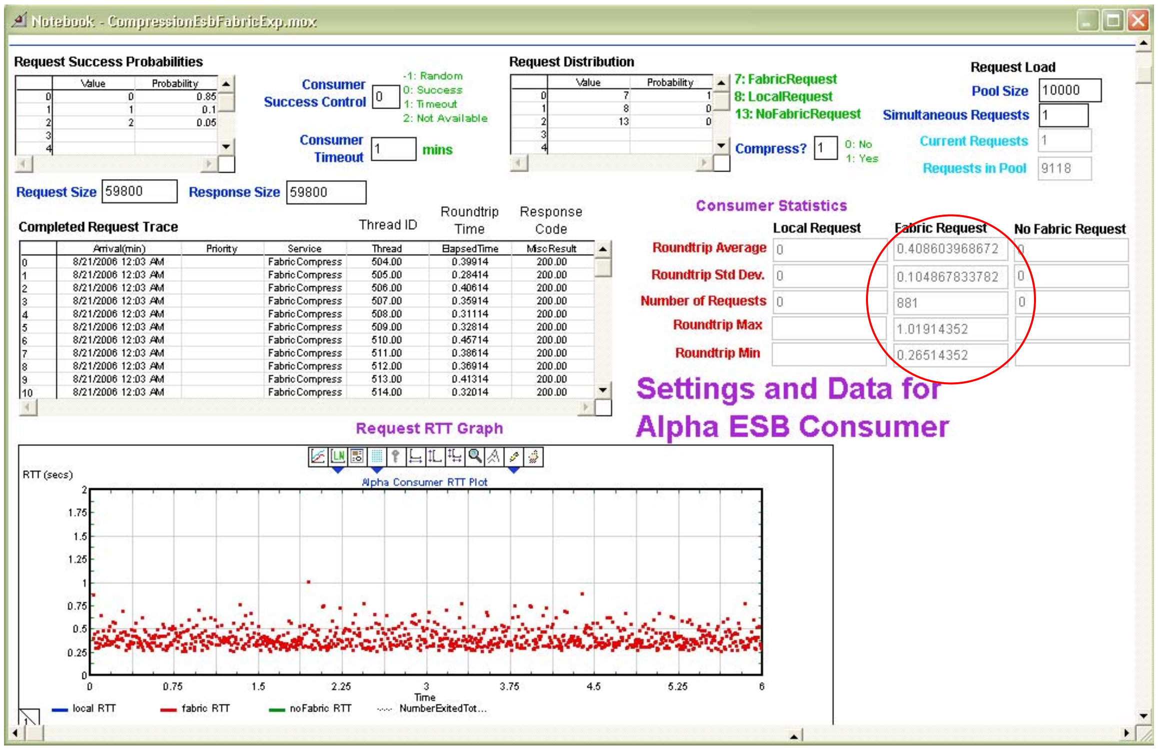
Consumer Statistics

	Local Request	Fabric Request	No Fabric Request
Roundtrip Average	0	22.71291481481	0
Roundtrip Std Dev.	0	3.026068422875	0
Number of Requests	0	15	0
Roundtrip Max		30.79711111111	
Roundtrip Min		18.81611111111	

Settings and Data for Alpha ESB Consumer

Figure 11. AquaLogic Low Bandwidth No Compression

Figure 12 and Figure 13 document the analogous AquaLogic model runs – same message sizes and simulated runtime – where the link properties have been changed to simulate a normal Ethernet network connection. Notice that the model runs show that adding compression to the message exchange path actually reduces the number of roundtrips processed and increases roundtrip time. This is because the model accounts for the processing time to compress and decompress the message and, for fast connections, this extra processing overhead negates the benefits that compression provides in reducing the size of the payload.



Request Success Probabilities

	Value	Probability
0	0	0.85
1	1	0.1
2	2	0.05
3		
4		

Consumer Success Control: 0 (-1: Random, 0: Success, 1: Timeout, 2: Not Available)

Consumer Timeout: 1 mins

Request Distribution

	Value	Probability
0	7	1
1	8	0
2	13	0
3		
4		

7: FabricRequest
8: LocalRequest
13: NoFabricRequest

Compress? 1 (0: No, 1: Yes)

Request Load

Pool Size: 10000
Simultaneous Requests: 1
Current Requests: 1
Requests in Pool: 9118

Request Size: 59800 Response Size: 59800

Completed Request Trace (Thread ID, Roundtrip Time, Response Code)

	Arrival(min)	Priority	Service	Thread	ElapsedTime	MiscResult
0	8/21/2006 12:03 AM		FabricCompress	504.00	0.39914	200.00
1	8/21/2006 12:03 AM		FabricCompress	505.00	0.28414	200.00
2	8/21/2006 12:03 AM		FabricCompress	506.00	0.40614	200.00
3	8/21/2006 12:03 AM		FabricCompress	507.00	0.35914	200.00
4	8/21/2006 12:03 AM		FabricCompress	508.00	0.31114	200.00
5	8/21/2006 12:03 AM		FabricCompress	509.00	0.32814	200.00
6	8/21/2006 12:03 AM		FabricCompress	510.00	0.45714	200.00
7	8/21/2006 12:03 AM		FabricCompress	511.00	0.38614	200.00
8	8/21/2006 12:03 AM		FabricCompress	512.00	0.36914	200.00
9	8/21/2006 12:03 AM		FabricCompress	513.00	0.41314	200.00
10	8/21/2006 12:03 AM		FabricCompress	514.00	0.32014	200.00

Consumer Statistics

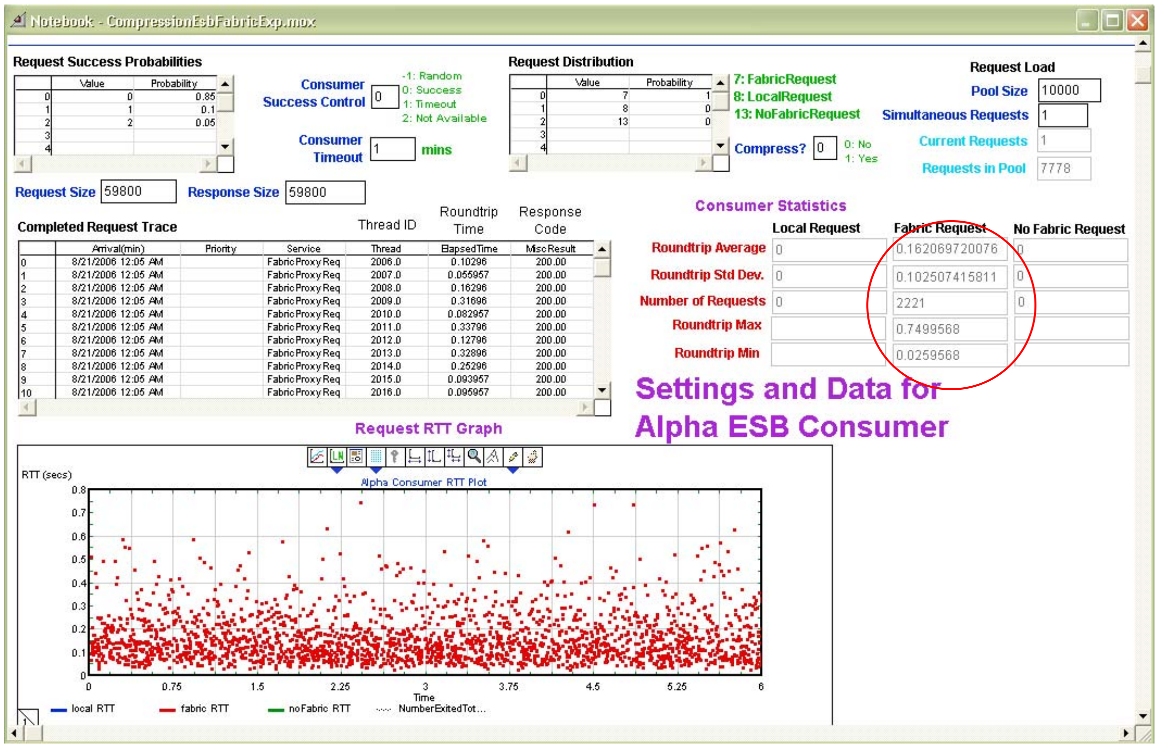
	Local Request	Fabric Request	No Fabric Request
Roundtrip Average	0	0.408603968672	0
Roundtrip Std Dev.	0	0.104867833782	0
Number of Requests	0	881	0
Roundtrip Max		1.01914352	
Roundtrip Min		0.26514352	

Settings and Data for Alpha ESB Consumer

Figure 12. AquaLogic Normal Bandwidth With Compression

The model runs report that when compression is applied, the average RTT is around 410 milliseconds with 881 complete roundtrips reported. Without compression, the average RTT is reduced to around 160 milliseconds with the total number of message roundtrips reported as 2221. For the high bandwidth case, delays associated with compression are significant – the roundtrip time is more than doubled.

The ARCES Test Report CDRL, submitted to ESC, reports in significant detail many more test runs for various message sizes, link conditions, and compression settings. By running companion experiments in the lab and comparing the results with related model simulation runs, we are working to determine how accurate the model is: how well it matches observed behavior and if it eventually can be used to predict real world behaviors. In addition, the lab experiments can help calibrate the model by providing target runtime statistics that are used to adjust model parameters.



Request Success Probabilities

	Value	Probability
0	0	0.85
1	1	0.1
2	2	0.05
3		
4		

Consumer Success Control: 0 (-1: Random, 0: Success, 1: Timeout, 2: Not Available)

Consumer Timeout: 1 mins

Request Distribution

	Value	Probability
0	7	1
1	8	0
2	13	0
3		
4		

7: FabricRequest
8: LocalRequest
13: NoFabricRequest

Compress? 0 (0: No, 1: Yes)

Request Load

Pool Size: 10000
Simultaneous Requests: 1
Current Requests: 1
Requests in Pool: 7778

Request Size: 59800
Response Size: 59800

Completed Request Trace

	Arrival(min)	Priority	Service	Thread	ElapsedTime	MiscResult
				Thread ID	Roundtrip Time	Response Code
0	8/21/2006 12:05 AM		FabricProxyReq	2006.0	0.10296	200.00
1	8/21/2006 12:05 AM		FabricProxyReq	2007.0	0.055957	200.00
2	8/21/2006 12:05 AM		FabricProxyReq	2008.0	0.16296	200.00
3	8/21/2006 12:05 AM		FabricProxyReq	2009.0	0.31696	200.00
4	8/21/2006 12:05 AM		FabricProxyReq	2010.0	0.082957	200.00
5	8/21/2006 12:05 AM		FabricProxyReq	2011.0	0.33796	200.00
6	8/21/2006 12:05 AM		FabricProxyReq	2012.0	0.12796	200.00
7	8/21/2006 12:05 AM		FabricProxyReq	2013.0	0.32896	200.00
8	8/21/2006 12:05 AM		FabricProxyReq	2014.0	0.25296	200.00
9	8/21/2006 12:05 AM		FabricProxyReq	2015.0	0.093957	200.00
10	8/21/2006 12:05 AM		FabricProxyReq	2016.0	0.095957	200.00

Consumer Statistics

	Local Request	Fabric Request	No Fabric Request
Roundtrip Average	0	0.162069720076	0
Roundtrip Std Dev.	0	0.102507415811	0
Number of Requests	0	2221	0
Roundtrip Max		0.7499568	
Roundtrip Min		0.0259568	

Settings and Data for Alpha ESB Consumer

Figure 13. AquaLogic Normal Bandwidth No Compression

6 Validating the ESB Fabric and Compression Model

The next four figures document partial tabulated results comparing our AquaLogic model runs under low bandwidth conditions to AquaLogic execution results as reported in the lab. Both tables and graphs help us document the comparison. We collect raw data in the lab and analogous raw data from the model runs. We then use Excel to generate various. Each experiment and model run was performed a number of times (usually 4 repetitions) and the separate data averaged into a single RTT estimate. We also compute the average throughput (number of completed message roundtrips per minute) since the model and lab runtimes were different for the various message sizes. Figure 14 and Figure 16 show data comparison tables for the low bandwidth cases, with and without compression. These tables show 5 different message size samples and computed RTT and average throughput measured in requests per minute. The top table shows data generated in the lab and the bottom table shows data collected in the model.

Message Size	File Name	Uncompressed Size (KB)	Avg. RTT	AVG Throughput (requests/minute)
0KB - 1KB	class1.xml	0.7	442	134.38
1KB - 10KB	class2.xml (io.xml)	4.3	543	109.50
10KB-100KB	class3.xml (foxpro.xml)	59.8	3951	15.21
100KB-214KB	class4.xml (progress.xml)	173.2	8173	7.33
1MB - 8MB	class5.xml (factbook.xml)	4,124	392467	0.16

Message Size	File Name	Uncompressed Size (KB)	Avg. RTT	AVG Throughput (requests/minute)
0KB - 1KB	class1.xml	0.7	440	136.13
1KB - 10KB	class2.xml (io.xml)	4.3	540	110.88
10KB-100KB	class3.xml (foxpro.xml)	59.8	3940	15.13
100KB-214KB	class4.xml (progress.xml)	173.2	8221	7.28
1MB - 8MB	class5.xml (factbook.xml)	4,124	397229	0.15

Figure 14. Model Vs. Lab Compressed Data Comparison

Figure 15 and Figure 17 show the average RTT's graphed next to each other with the model data shown in red and the lab data shown in blue. The RTT values are in milliseconds and the scale on the vertical axis is logarithmic to allow the widely differing roundtrip average times for each message size category to be displayed on the same graph. Note however that the logarithmic scale can make the bar heights appear to be identical for each message size.

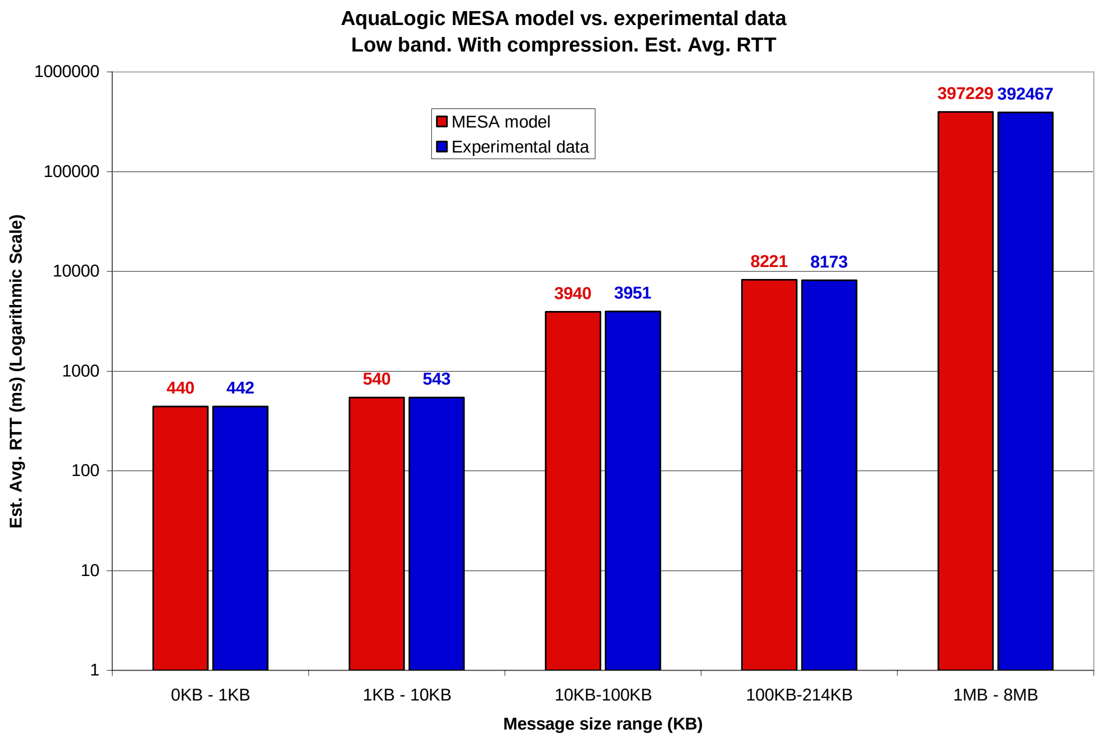
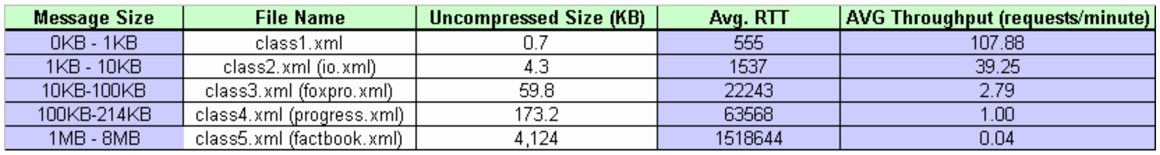
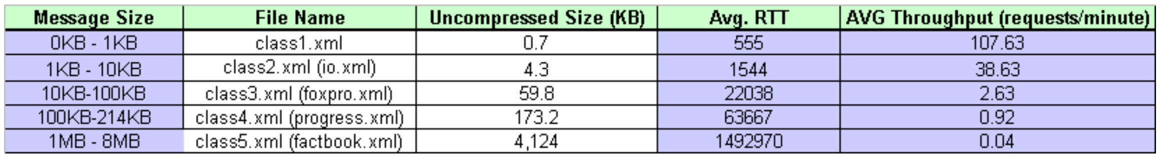


Figure 15. Model Vs. Lab Compressed Data RTT



Message Size	File Name	Uncompressed Size (KB)	Avg. RTT	AVG Throughput (requests/minute)
0KB - 1KB	class1.xml	0.7	555	107.88
1KB - 10KB	class2.xml (io.xml)	4.3	1537	39.25
10KB-100KB	class3.xml (foxpro.xml)	59.8	22243	2.79
100KB-214KB	class4.xml (progress.xml)	173.2	63568	1.00
1MB - 8MB	class5.xml (factbook.xml)	4,124	1518644	0.04



Message Size	File Name	Uncompressed Size (KB)	Avg. RTT	AVG Throughput (requests/minute)
0KB - 1KB	class1.xml	0.7	555	107.63
1KB - 10KB	class2.xml (io.xml)	4.3	1544	38.63
10KB-100KB	class3.xml (foxpro.xml)	59.8	22038	2.63
100KB-214KB	class4.xml (progress.xml)	173.2	63667	0.92
1MB - 8MB	class5.xml (factbook.xml)	4,124	1492970	0.04

Figure 16. Model Vs. Lab Uncompressed Data Comparison

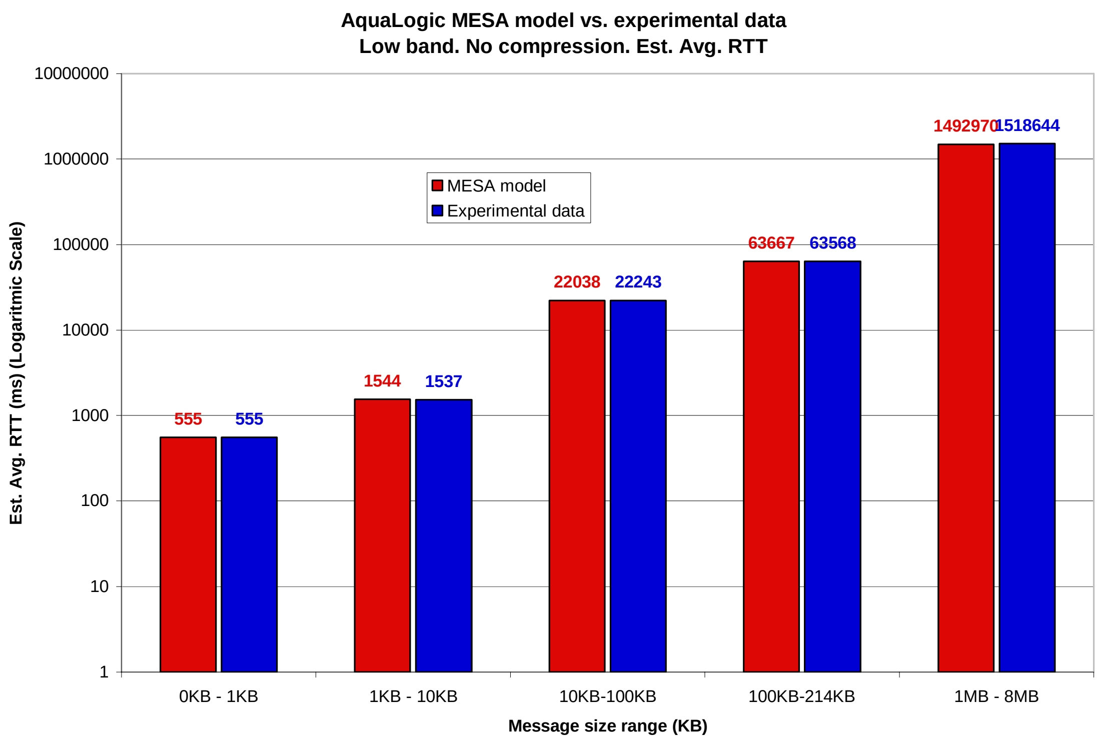
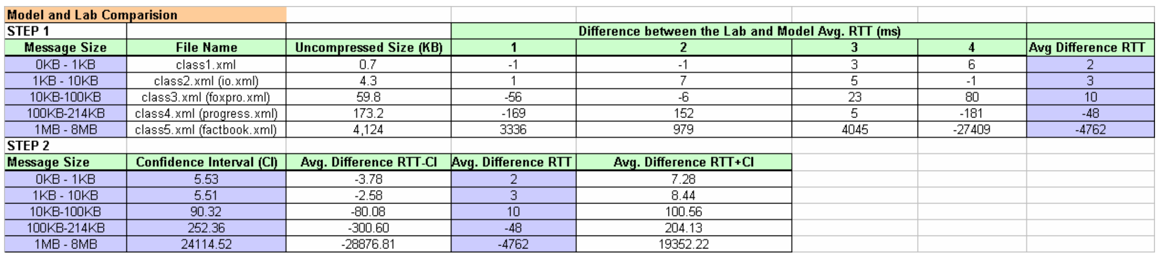


Figure 17. Model Vs. Lab Uncompressed Data RTT

In most cases, the data produced in the lab and the data produced by running the model show significant agreement. Recently, ARCES invited Dr. Averill Law [3], well-known author and consultant on simulation modeling and analysis, to review our experimental methods and reported results. He suggested that we take our replicated experiments, compute the difference in means for each replication, and generate a confidence interval for this difference as an estimate of the actual difference between reported lab and model RTT results. By doing this we can statistically report how close the two methods of benchmark generation are to one another. In particular, if the number 0 is in the confidence interval, we can assert that the model is a fair and accurate measure of the data generated in the lab.

The next four figures summarize our statistically based comparison results. We provide tabular reports of the difference in means for each lab and model run, the computed average difference (the last column) and the generated 95% confidence interval. Then the confidence intervals for the four smaller message sizes are shown graphically. The difference in scale for the largest message RTT makes showing confidence intervals for all five message size categories on the same chart problematic. Figure 18 and Figure 19 report our results for the low bandwidth, with compression comparison.



Model and Lab Comparision							
STEP 1			Difference between the Lab and Model Avg. RTT (ms)				
Message Size	File Name	Uncompressed Size (KB)	1	2	3	4	Avg Difference RTT
0KB - 1KB	class1.xml	0.7	-1	-1	3	6	2
1KB - 10KB	class2.xml (io.xml)	4.3	1	7	5	-1	3
10KB-100KB	class3.xml (foxpro.xml)	59.8	-56	-6	23	80	10
100KB-214KB	class4.xml (progress.xml)	173.2	-169	152	5	-181	-48
1MB - 8MB	class5.xml (factbook.xml)	4,124	3336	979	4045	-27409	-4762
STEP 2							
Message Size	Confidence Interval (CI)	Avg. Difference RTT-CI	Avg. Difference RTT	Avg. Difference RTT+CI			
0KB - 1KB	5.53	-3.78	2	7.28			
1KB - 10KB	5.51	-2.58	3	8.44			
10KB-100KB	90.32	-80.08	10	100.56			
100KB-214KB	252.36	-300.60	-48	204.13			
1MB - 8MB	24114.52	-28876.81	-4762	19352.22			

Figure 18. Difference in Means, Low Bandwidth, Compressed Messages

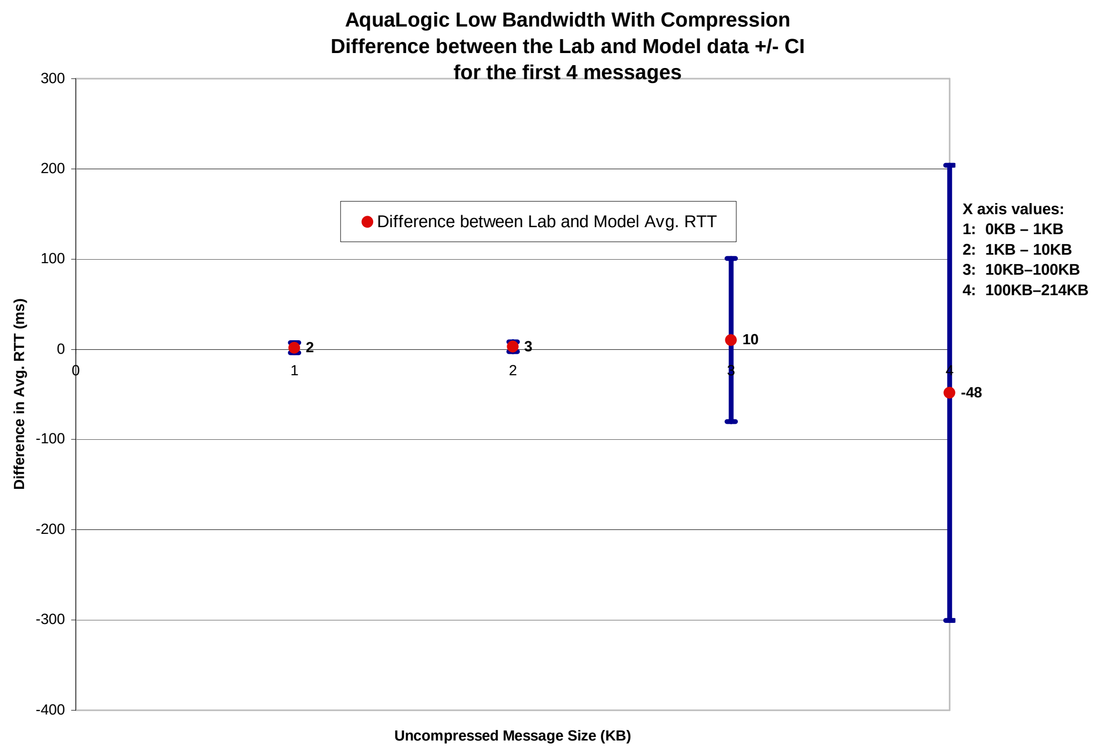
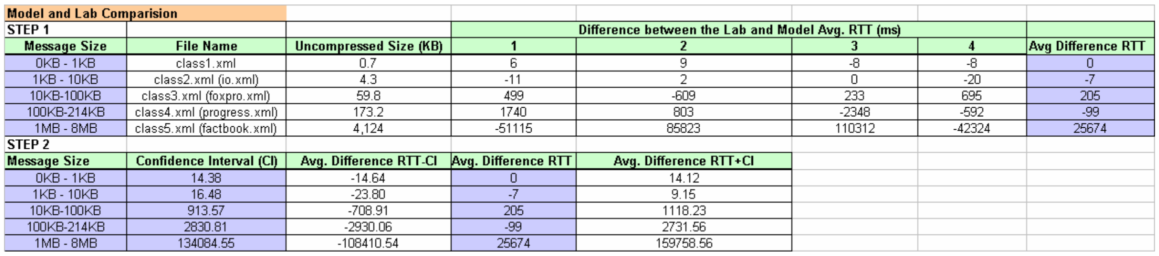


Figure 19. 95% Confidence Intervals, Low Bandwidth, Compressed Data

For the two smallest message size categories in the compressed data analysis, the width of the confidence interval for the difference in means is very small, yet 0 is contained in the interval. For all of the other message sizes, the width of the confidence interval is larger but still comfortably contains 0.

The next two figures summarize the same statistical analysis for the low bandwidth, no compression comparison. Once again, the confidence intervals are larger for the three largest message size categories but all of them include 0. Only the four smaller message size intervals are shown in Figure 21, once again for reasons of scale.



Model and Lab Comparision							
STEP 1			Difference between the Lab and Model Avg. RTT (ms)				
Message Size	File Name	Uncompressed Size (KB)	1	2	3	4	Avg Difference RTT
0KB - 1KB	class1.xml	0.7	6	9	-8	-8	0
1KB - 10KB	class2.xml (io.xml)	4.3	-11	2	0	-20	-7
10KB-100KB	class3.xml (foxpro.xml)	59.8	499	-609	233	695	205
100KB-214KB	class4.xml (progress.xml)	173.2	1740	803	-2348	-592	-99
1MB - 8MB	class5.xml (factbook.xml)	4,124	-51115	85823	110312	-42324	25674
STEP 2							
Message Size	Confidence Interval (CI)	Avg. Difference RTT-CI	Avg. Difference RTT	Avg. Difference RTT+CI			
0KB - 1KB	14.38	-14.64	0	14.12			
1KB - 10KB	16.48	-23.80	-7	9.15			
10KB-100KB	913.57	-708.91	205	1118.23			
100KB-214KB	2830.81	-2930.06	-99	2731.56			
1MB - 8MB	134084.55	-108410.54	25674	159758.56			

Figure 20. Difference in Means, Low Bandwidth, Uncompressed Messages

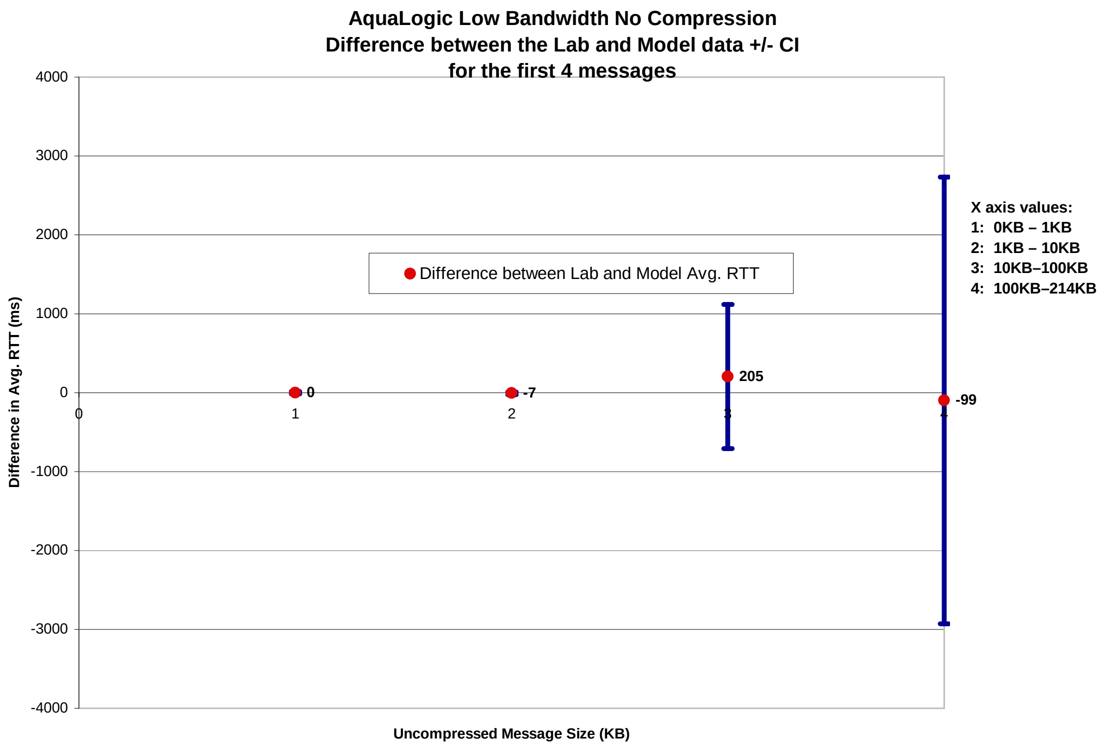


Figure 21. 95% Confidence Intervals, Low Bandwidth, Uncompressed Data

7 ARCES C2 Service Discovery Model

This is a placeholder for final version – actual text to be included as part of final paper submission by April ICCRTS deadline. This is still a work-in-progress.

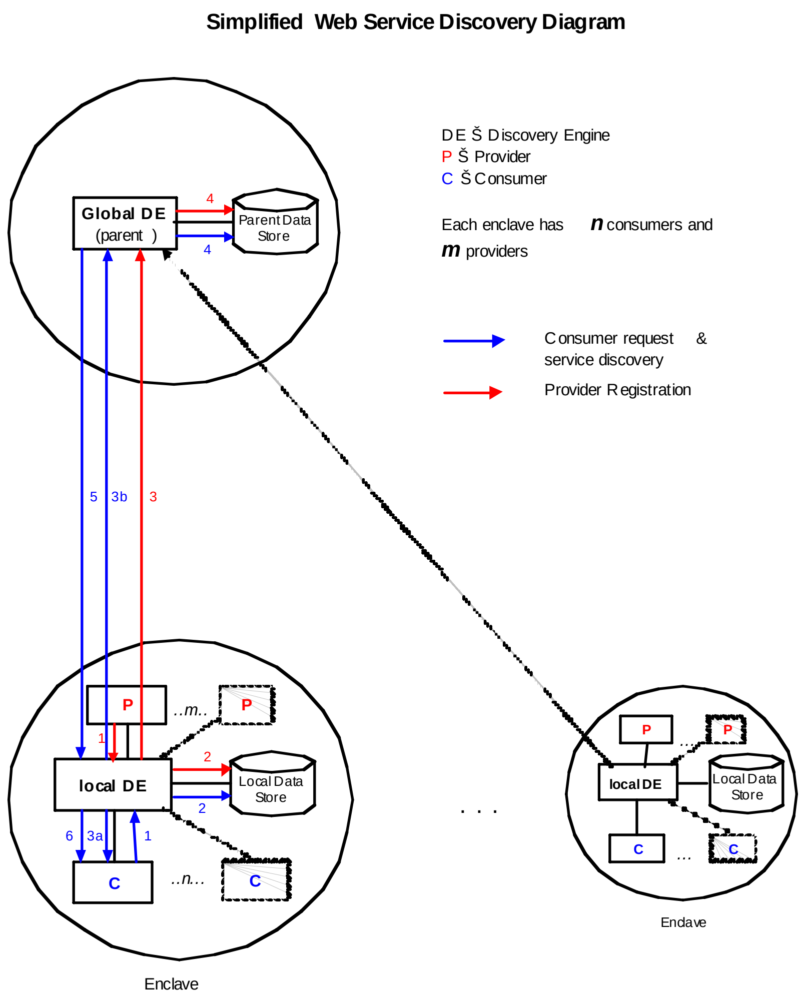


Figure 22. Conceptual Service Discovery Model

8 Conclusions

Based on the results reported on in sections 2 through 7 of this paper, we can make the following conclusions:

1. There are significant advantages to having the means to define SOA architectural models supporting C2 contexts and scenarios.
2. MESA/Extend provides a powerful and appropriate modeling technology to create, execute and evaluate such SOA architectural models.
3. MESA's ability to separate a model topology from model service semantics provides useful opportunities to independently alter service definitions and node arrangements when experimenting with a SOA fabric.
4. SOA models should be built based on specific questions whose answers will help model users to understand, explore and measure SOAs.
5. Use cases can serve as a rich elaboration of specific questions and be the basis for model validation.

6. Real world data compression statistics and measurements can help provide additional realism in model execution.
7. The same model topology and general service characteristics can serve as the basis for specialized models reflecting particular ESB behaviors and performance characteristics when combined with model support for message sizes and communication channel characteristics.
8. It is possible to calibrate the models for different types of ESBs (AquaLogic and ServiceMix) to get statistically validated performance data when simulated requests and responses are generated in the model that approximate the actual processing of requests and responses in the lab between instances of real ESB installations.
9. In order to allow the model to be tuned to have acceptably close correspondence to performance measurements made in the lab, it was necessary to insert parametric effects into the model so that model processing delays are adjusted based on message size, communication link characteristics, and ESB type.

9 References

1. "VERIFICATION AND VALIDATION OF SIMULATION MODELS", *Proceedings of the 2003 Winter Simulation Conference*, Robert G. Sargent, Department of Electrical Engineering and Computer Science, L.C. Smith College of Engineering and Computer Science, Syracuse University, Syracuse, NY 13244, U.S.A.

2. "Terminology for model credibility" *Simulation*, 32(3):103-104, Schlesinger, et al. 1979.

3. *Simulation Modeling and Analysis*, Third Edition, Averill Law and W. David Kelton, McGraw-Hill, 2000.