Addis Ababa University

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

By

Solomon Tiku

Performance Predictive Model of ESB as Service Integration Grows: The Case of ethio telecom

A Thesis Submitted as a Partial Fulfillment of the Requirements for the Degree of Master of Science in Telecommunication Engineering

Advisor: Mesfin Kifle (PhD)

Date: Jan 2020

ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Performance Predictive Model of ESB as Service Integration Grows: The Case of ethio telecom

By

Solomon Tiku

Approval by Board of Examiners

Chairman, School Graduate Committee	Signature
_____	_____
Advisor Dr. Mesfin Kifle	Signature _____
Examiner Dr. Yalemzewd Negash	Signature _____
Examiner Dr. Surafel Lemma	Signature _____

Abstract

Service Oriented Architecture (SOA) is an architectural pattern that is based on the concept of a service and addresses, the requirements of loosely coupled, and protocol independent distributed computing. SOA used enterprise Service Bus (ESB) to realize its principles. ESB is an architecture to overcome the limitation of traditional architectures. One of its main functions is message transformation, which is the conversion process of data structure among heterogeneous systems during integration. The second function is routing functionality between runtime services which may introduce performance overheads. Ethiotelecom has implemented ESB to integrate business-critical systems , due to additional service integrations, ESB's performance problem becomes a challenge to meet service level agreements. The main objective of this thesis is to develop and implement a performance predictive model, which can be used to evaluate the performance of the ESB system.

Queueing Petri nets are powerful formalism that can be exploited for modeling distributed systems and analyzing their performance and scalability. By combining the modeling power and expressiveness of queueing networks and stochastic Petri nets, queueing Petri nets provide a number of advantages. The model can be useful for service quality and performance management process during design and service operation times. The Jmeter measured performance data is used for comparison and validity of the Queuing petri net network model of the case study system.

Both in the experiment and the model similar performance impacts were observed for variety of workload inputs. Clearly one can see the level of the impact for each performance metrics as integration grows. The validation result indicates, the practicality of the model to do performance prediction and capacity estimation during system design and service operation of integrated applications in a heterogeneous environment. Moreover, unlike the existing system in which the case study is based upon, the model solution is further applicable for new capacity demands in similar system planning works.

Keywords--*Enterprise service bus; ESB performance; performance model; queue petri net, performance Metrix.*

Acknowledgement

First, I would like to express my sincere and special thanks of gratitude to my advisor Dr. Mesfin Kifle, and my evaluators who have given me great help, excellent advice and has treated me with patience throughout this thesis work.

I also want to thank my wife for encouraged me along the way and advice. Beside she, I want to thank my Kids especially my little daughter Remla , for their patience during the thesis work. Finally, I want to give my gratitude to ethio telecom and Addis Ababa University for giving me the chance to study in this program.

Table of Contents

Abstract iii
Acknowledgement..... v
List of Figures ..... ix -x
List of Tables xi
Chapter 1 Introduction 1
1.1 Background
1.1.1 Motivation..... 2
1.2 Statement of the Problem 3
1.3 Objectives 3
1.3.1 General Objective 4
1.3.2 Specific Objectives 4
1.4 Method..... 4
1.5 Scope and Limitation 4
1.6 Significance of the Study 4
1.7 Thesis Structure 5
Chapter 2 Literature Review 6
2.1 Introduction 6
2.2 Enterprise Application Integration 5
2.2.1 EAI as a Middleware.....6
2.2.2 Parameterization of EAI Patterns……..…..6
2.3 Enterprise Service Bus 7
2.4 ESB messaging & Routing Functionality.....8
2.4.1 Mediation……..……8
2.4.2 Message-oriented middleware (MOM)……..…….9
2.4.3 ESB as Hub node…….....10
2.4.4 The Broker Interaction Pattern……...11
2.5 Service Oriented Architecture 12
2.6 Web Service 13
2.7 Software Performance.....14
2.7.1 Software Performance Engineering (SPE)……....15
2.7.2 Capacity Management: Continuous Capacity Planning……....15

2.8 What Is a Queueing Network Model?.....16
2.8.1 Multiple Service Centers……..……..16
2.8.2 Single Station Queuing Systems……..17
2.8.3 Closed Queue Network .....……18
2.8.4 Poisson Process ……20
2.8.5 A two-state/birth-death Poisson process ……..……22
Summary 22
Chapter 3 Related works 23
3.1 Introduction 23
3.2. ESB Performance Measurement…….…23
3.3 Queueing Petri nets……26
Summary 27
Chapter 4 The Proposed Solution Approach.....30
4.1 Introduction 30
4.2 ESB Implementation in ethio Telecom 30
4.2.1 ESB Interface and Integration..... 30
4.2.2 Product overview..... 32
4.2.3 Routing with message flows..... 32
4.3 The Business Case..... 36
4.3.1 SIM Card Replacement Process sequence diagram..... 37
4.3.2 Publish /Subscribe Scenario in SIM Card Replacement Process..... 37
4.4 The Propose Technique 38
4.4.1 Communication Patterns Selected..... 39
4.4.2 Performance Models……..……..40
4.4.3 The Modeling Process……41
4.4.4 Performance Measurements……43
4.4.5 Tools Used……46
Summary 47
Chapter 5 The Model 48
5.1 Introduction 48
5.2 System Set up Description 52
5.3 The Model..... 52
5.3.1 Queueing Petri Net Modeling Environment.....56
Chapter 6 Proposed model Solution Evaluation……..59
6.1 Using Steady State Analysis and Confidence Intervals for Data analysis..... 60

6.2 Experiment Results ……..62
6.3 Discussion ……...67
Summary ……67
Chapter 7 Conclusion, Recommendation and Future Work ……68
7.1 Conclusion ……68
7.2 Recommendation …… 69
7.3 Limitations of the work……..……..69
7.4 Future Work ……70
REFERENCES…….71
Appendix ……..75
Declaration …… 79

List of Abbreviations

AMF	Action Message Format
API	Application Program Interface
B2B	Business-to-Business
BSS	Business Support System
BPEL	Business Process Execution Language
CBE	Commercial Bank of Ethiopia
CBS	Convergent Billing System
CGSPN	Colored, Generalized, Stochastic Petri Nets
CPN	Colored Petri Net
CRM	Customer Resource Management
EAI	Enterprise Application Integration
EBS	Event-based systems
EJB	Enterprise Java Bean
ERP	Enterprise resource planning
ESB	Enterprise Service Bus
FCFS	First Come-First-Served scheduling strategy
GSPN	Generalized Stochastic Petri Net
HTTP	Hyper Text Transfer Protocol
IBM	International Business Machines
IPCC	Internet Protocol Call Center
IS	Infinite Server scheduling strategy
IT	Information Technology
J2EE	Java Platform, Version 2 Enterprise Edition
JNDI	Java Naming and Directory Interface
JMS	Java Message Service
JS	JavaScript
JSON	JavaScript Object Notation
MOM	Message-oriented Middleware
MQ	Message queue
OSS	Operational system support
P2P	Point to point
PHP	Hypertext Preprocessor
PHP	Hypertext Preprocessor
PN	Petri Net
PRIO	Priority Scheduling strategy
PS	Processor Sharing scheduling strategy
QN	Queuing Network
QPN	Queueing Petri Net
RAM	Random-Access Memory
RANDOM	Random Scheduling strategy
RPC	Remote Procedure Calls
SAX	Simple API for XML
SCM	Supply-Chain Management
SDP	Session Description Protocol
SIM	Subscriber Identity Module

SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SRR	Service Registry and Repository
SPE	Software Performance Engineering
TEP	Telecom Expansion Project
UDDI	Universal Description, Discovery, and Integration
W3C	World Wide Web Consortium
WS	Web Service
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

List of Figures

Figure 2:1 Message Routing …… 6

Figure 2:2 An example of an enterprise service bus …….7

Figure 2:3 High level ESB architecture ……7

Figure 2:4 Basic ESB Scenario ……..…..10

Figure 2:5 SOA Reference Architecture with Product Mapping……..13

Figure 2:6 Basic Web Service Integration Model…….13

Figure 2:7 Stack view of Web services technology……. 14

Figure 2:8 A Network of Queues …… 17

Figure 2:9 Schematic diagram of a queening sys ……..17

Figure 2:10 Single service station with a single queue……..18

Figure 2:11 A closed network ……...20

Figure 2:12 Describes birth-death process ……23

Figure 2:13 Describes birth-death chain process ……..……23

Figure. 3:1 A queueing place and its shorthand notation……. 28

Figure 4:1 ESB Interface ……..32

Figure 4:2 A typical routing mechanism……..…..34

Figure 4:3 Database lookup of routing information…… 35

Figure 4:4 Routing with lookup using shared variables and refreshing of cache…… 35

Figure 4:5 SIM Card Replacement Sequence diagram ……37

Figure 4:6 Application's interactions on SIM Card Replacement Sequence diagram.…...38

Figure 4:7 Database lookup of routing information……... .35

Figure 4:8 Routing with lookup using shared variables and refreshing of cache…… 35

Figure 4:9 SIM Card replacement Sequence diagram ……....37

Figure 4:10 Publisher/subscriber messaging using topics ……..…39

Figure 4:11 Describe how JMeter works ……..… 45

Figure 4:12 SimQPN's main simulation routine ……..…..46

Figure 5:1 Sequence diagram of the publish/ subscribe system ……...….50

Figure 5:2 Graphical representation of empirical setup…….…..…. .51

Figure 5:3 Graphical representations of JMeter with sample output data…….…….51

Figure 5:4 QPE Main Window……52

Figure 5:5 Graphical user interface of color configuration …….……..…52

Figure 5:6 Graphical user interface & simulation run configurations of QPME …….,,54

Figure 6:1 Performance modeling and prediction …….……..... .56

Figure 6:2 Impact of no. of subscribers Vs throughputs…….…….....63

Figure 6:3 Impact of no. of event arrivals Vs throughputs ……...…64

Figure 6:4 Impact of no. of event arrivals in two topics Vs throughputs……...…65

Figure 6:5 Impact of no. of event arrivals in one & two topics 66

List of Tables

Table 5:1 Jmeter empirical test configuration for each test scenario type ……52

Table 5:2 Description of the model assumptions ……57

Table 5:3 Description of the model elements ……..58

Table 6:1 Scenario one Impact of the Number of Subscribers/ message replication ……..62

Table 6:2 Scenario two (Pub/Sub with one topic) ……...63

Table 6:3 Scenario three (Pub/Sub with multiple topic) ……..64

Table 6:4 Scenario four (CPU usage, response time and throughput with topics) ……..65

Table 6:5 Impact of topic configurations (scenario four) with stress load test……...67

Table Annex-A Validation Procedure ……...77

Table Annex-B Node model design description ……...78

Table Annex-C Model Transitions Design ……..79

1. Chapter One: Introduction

1.1 Background

Information processing system designers need methods for the quantification of system design factors such as performance and reliability. Modern computer and communication systems process complex workloads with random service demands. Probabilistic and statistical methods are commonly employed for the purpose of performance and reliability evaluation.

Event-based systems (EBS) have been gaining attention in many domains of industry. With the advent of ambient intelligent and ubiquitous computing, many new applications of EBS have been proposed, for example, in the areas of transport information monitoring, event-driven supply chain management, ubiquitous (wireless) sensor environments, environmental monitoring, ambient assisted living, and location-based services. Many of these novel event-based applications are highly distributed and data intensive and hence pose some serious performance and scalability challenges. With the increasing popularity of EBS and their gradual adoption in mission critical areas, performance issues are becoming a major concern [14].

The dynamics of most EBS applications and their underlying middleware makes it difficult to monitor and analyze system performance during operation. Closely related to EBS is the publish-subscribe paradigm that is nowadays used as a building block in major new software architectures and technology domains such as enterprise service bus (ESB), enterprise application integration (EAI), service-oriented architecture (SOA) and event-driven architecture (EDA). Modern EBSs are implemented using several technology platforms such as centralized systems based on message-oriented middleware (MOM), e.g. IBM WebSphere and TIBCO Rendezvous, large-scale distributed event-based systems (DEBS), e.g. SIENA, Hermes or REBECA. Several standards such as Java Message Service (JMS), Advanced Message Queuing Protocol (AMQP), and Data Distribution Service (DDS) have been established and are supported by middleware vendors [14]. In ethio telecom IBM Web Sphere ESB was implemented as middleware technology to realize SOA principle. Web Sphere ESB delivers its infrastructure to connect applications that have standard-based interfaces. It can be a stand-alone product, or it can be included in Web Sphere

process server. ESB provides different functions such as Routing, Data Transformation, Protocol Transformation and others [1]. The publish/subscribe paradigm provides communication services where message addressing is implicitly handled by an Event Notification Service (ENS) that matches the content of events produced by publishers against interests expressed by subscribers in the form of subscriptions [53].

Distributed event-based systems (DEBS) are gaining increasing attention in new application areas such as transport information monitoring, event-driven supply chain management and ubiquitous sensor-rich environments. However, as DEBS increasingly enter the enterprise and commercial domains, performance and quality of service issues are becoming a major concern. To avoid the pitfalls of inadequate Quality of Service (QoS), it is essential that event-based systems are subjected to a rigorous performance and scalability analysis before they are put into production. The analysis should include a detailed characterization of the expected system workload as well as its anticipated development over time. Furthermore, methods are needed to estimate the system performance as a function of its configuration and the workload it is exposed to. Common performance metrics of interest are the expected event notification latency as well as the utilization and message throughput of the various system components (event brokers, network links, etc) [54].

Publish/subscribe-based messaging systems are used increasingly often as a communication mechanism in data-oriented web applications such as Web 2.0 applications, social networks, online auctions and information dissemination applications to name just a few. Moreover, the publish/subscribe paradigm is part of major technology domains including Enterprise Service Bus, Enterprise Application Integration, Service-Oriented Architecture and Event-Driven Architecture. With the growing adoption of these technologies and applications, the need for benchmarks and performance evaluation tools in the area of publish/subscribe systems increases [13].

The goal of this thesis is to develop performance predictive model using common modeling approaches to predict the behavior of ESB and its performance from growing up service integrations.

1.1.1 Motivation

For the last 15 years the ethiotelecom has been delivering services which are many in type and with wide subscriber base, the infrastructures are heterogeneous and vendor specific. This environment has introduced IT services integration and quality assurance challenges. The traditional monitoring and service management tools for business-critical systems (network elements, data centers, and IT infrastructures) and together with the existing IT service architecture create the challenge of finding a balance between scalability and flexibility of computing resources.

The health of IT systems is significantly influencing the company service delivery quality, thus system performance constraints, particularly the lack of enough, experienced performance management is a big factor on delivery of a quality service. For seamless service interaction and integration, the company has implemented an enterprise service bus (ESB), and as services integration grows, service performance challenges become an issue, the performance of an ESB based applications like ERP, are critically impacting the company business processes. Many legacy and new systems' integrations have been done and it has been planned to add more. Due to this, there is a risk of ESB's performance to be bottleneck on the business. Also due to lack of proper performance analysis and capacity prediction, there is a chance to introduce unnecessary investments on more servers and system licenses.

1.2 Statement of the Problem

The Enterprise Service Bus (ESB) is a natural evolution and convergence of Enterprise Application Integration (EAI) and Message Oriented Middleware (MOM) implementing the Java Message Service (JMS). It is also inspired by the Web Service technology, leading to the following traditional definition [5]. ESB is an intermediary that makes a set of reusable business services widely available.

The performance of an ESB-based application is often critically important to a business. If an ESB doesn't perform well, will causes the integrated services to give limited benefit. Before some years the company has implemented a single ESB solution, as the integration grows, the new load starts creating performance problem and future related business risk. To minimize the possible load and

service interruptions the company technical management team has taken some work around solutions to reduce and monitor the load on the ESB. But as many ESB related researches shows [29], traditional monitoring tools are very inefficient to look ESB's full health. Researches show [26], ESB capacity should be planned and predicted early before impacting the service delivery.

In response of the situation, this study proposes a plan to carry out performance analysis and investigations to develop a prediction model using queue petri net modeling technique that can help to manage or control the ESB performance challenges that are coming from the growth of new service integrations.

Research questions

- What performance management practice & modeling technique can address the problem?
- What is the effect of adding (or removing) additional service on response time, throughput and CPU utilization?

1.3 Thesis Objectives

1.3.1 General Objective

The purpose of this study is to develop a performance predictive model of ESB as service integration grows, and the model can be used as a contribution work to improve ethio telecom's ESB and other related systems performance and capacity management works.

1.3.2 Specific Objective

- Understand the state of art on ESB performance and technical analysis
- Investigate the ESB service performance and monitoring practice at ethiotelecom
- Identify the determinant factors on the existing system
- Develop a predictive model
- Test and evaluate the model

1.4 Method

This research uses different methods to collect and analyze the data and finally develop a performance predictive model of ESB. The methods are:

- Literature review: different published papers, white papers, books and company design document will be used.
- Interview and focus group discussion (informal): to collect information about the performance of the implemented ESB.
- Data collection: By using and comparing different company documents and the- state-of-the-art identify the gap and proposed the solution.
- Evaluation: Finally perform evaluation by using the model and experiment results.

1.5 Scope and Limitation

There are different factors that affect the performance of ESB messaging: In this work, the focus was on the performance overhead of ESB's routing and message transformation factors. But this work does not include security overhead features.

1.6 Significance of the Study

The ESB infrastructure have been supporting the company sensitive business transactions and a backbone for handling critical business processes and operations. Performance management and practice of such systems is the key factor for the company service delivery quality and business success. This research will allow getting better insight into performance overheads in various stages of service based ESB implementations.

Therefore, the result of this work can bring an improvement on the company service delivery quality and maximizing company's revenue.

1.7 Thesis Structure

This research documentation is organized hierarchically in a way that discusses from the general concept of the research and its background information to a specific implementation and validation of scenarios. The rest of the Chapters in this research document is organized as follows.

Chapter Two discusses the theoretical background and detail concepts of the research based on reviewed literature. In this Chapter, the fundamental concepts of enterprise applications, Enterprise Applications Integration (EAI), Enterprise service bus (ESB), Service Oriented Architecture (SOA) and related concepts are discussed in detail. In Chapter Three other researchers related works is discussed, the methodology used, solution provided, research validation of the related works are discussed.

Chapter Four discusses the proposed solution approach and studying related areas of the company in which this research is based upon. Studying the company's problem area is very important and basic process to propose a solution based on the findings in the existing situation. Therefore, solution approach for the problem is discussed in Chapter four. Whereas in Chapter Five, the model and design considerations are explained together .

In Chapter Six, the Implementation and Validation of the proposed model is explained. Chapter Seven is the last Chapter of the research document. It contains the validation result, discussion, and interpretation of the model solution; it also explains the Limitation of the research and a concluding remark about the research. At last Recommendation and Future work are also discussed.

2. Chapter Two: Literature Review

2.1 Introduction

This chapter is focused on the basic concepts of the research by using different scientific research papers, articles and books. It consists of sections each of them focuses on certain area: Section 2.2 Enterprise Application Integration, Section 2.3 Enterprise Service Bus, Section 2.4 ESB Messaging & Routing Functionality, Section 2.5 Service Oriented Architecture, Section 2.6 Web Service, Section 2.7 Software Performance and Section 2.8 What is a Queueing Network Model?

2.2 Enterprise Application Integration

Due to the rapid growth of business world , organizations imposed to depend on technology and need of the integration of the disparate applications is in high demand. With the development of information technology, over the past several years, there have been numerous distributed computing models in the EAI domain, including Message-oriented Middleware (MOM), EAI, Web services, SOA and ESB. Before the appearance of ESB, many companies have adopted tradition architecture. ESB is a middle ware technology use to overcome the limitation that appears in traditional architecture [1]. Service orchestration works through the exchange of messages. This is usually accomplished through Enterprise Application Integration (EAI), which enables data integration, or the use of a central messaging engine such as an Enterprise Service Bus (ESB), which routes, transforms and enriches messages [19]. SOA uses open standards to communicate, and the utilization of web services are the most popular and most promising approach. Simple Object Access Protocol (SOAP) and Web Services Description Languages (WDSL) are commonly used together with Extensible Markup Language (XML). The use of open standards for communicating reduces the complexity of B2B communication [49].

Enterprise Application Integration (EAI) works as a connecting interface between the different applications running in an enterprise. EAI as a solution which is responsible to enhance, combined and communicate between the numbers of applications running in an enterprise. EAI has much functionality gathered together in a single and complete package which can provide better performance, unified and integrated data, and finally refine business processes of enterprise [23].

2.2.1 EAI as a Middleware

EAI works as middleware to work between different already installed programs. Middleware is a kind of computer software which facilitates and provides services other than the operating system. Middleware may have different types of its characteristics such as message-oriented middleware, procedural middleware, object and component middleware, service-oriented architecture and real time middleware. EAI is a kind of middleware which work for real time processing [23].

2.2.2 Parameterization of EAI Patterns

According to [24], we call configuration properties of individual types of EAI patterns as parameters. Here below some of the parameters of some patterns are discussed.

Message Construction: Message Construction patterns suggest ways how applications (i.e. filters) that use Message Channels can exchange a piece of information. The Message pattern, the most prominent pattern, describes how to structure information in form of a message consisting of a header and a body part.

Messaging Endpoints: Messaging Endpoints describe how applications are connected to messaging systems and thus, how applications send and consume messages. E.g. Polling Consumer, Event-driven Consumer or Selective Consumer.

Messaging Channels: are used to establish connections between applications and therefore transfer aspects of a messaging-based application. E.g. (i) reliable (yes/no), (ii) secure (kind of security, security token, security algorithm used), and (iii) persistent delivery (yes/no).

Message Routing: In order to decouple a message source from its ultimate destination, message-oriented middleware employs Routers to guide messages through a network of channels. Furthermore, routers not only direct messages through the system but they split, aggregate, or rearrange messages while traversing the middleware.

Fig 2:1 Message Routing [24]

2.3 Enterprise Service Bus

An ESB provides an infrastructure that removes any direct connection between service consumers and providers. Consumers connect to the bus and not the provider that implements the service. This type of connection further decouples the consumer from the provider. A bus also implements further value add capabilities. For example, security and delivery assurance can be implemented centrally within the bus instead of having this buried within the applications. Integrating and managing services in the enterprise outside of the actual implementation of the services in this way helps to increase the flexibility and manageability of SOA. ESB simplifies the task of both service consumers and service providers by adding a layer of abstraction that shields the consumers and the providers from having to worry about the specific of message format or message transport. It is provided different functions those used to create smooth integration among heterogeneous systems. Some of them are message Transformation, Content Based Routing and Virtualization [4].

Many ESBs provide the following features:

- Provides Web services connectivity, JMS messaging, and service-oriented integration by including support for:
  - SOAP/HTTP
  - SOAP/JMS
  - WSDL V1.1
  - UDDI V3.0

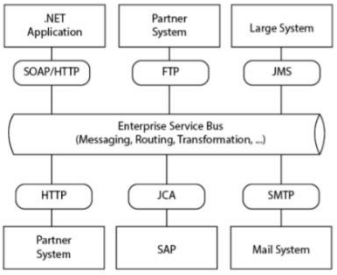
- Provides support for building an ESB with integration logic such as:

➢ Protocol conversion for messages received over HTTP and JMS
➢ Format transformation between XML, SOAP, and JMS message
➢ standards, and many more when used with adapters

- Mediation capabilities, including pre-built mediations for the following functions:

➢ Message logging
➢ Flow of business events
➢ Use of WebSphere Adapters to capture and disseminate business events

Here below figure 2:2 and figure 2:3 shows the above functionalities.



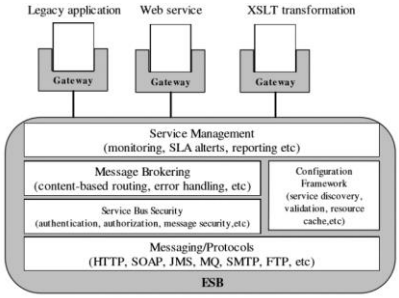


Fig 2:2 An example of an enterprise service bus[43] *Fig 2:3 High level ESB architecture[25]*

2.4 ESB Messaging & Routing Functionality

Messaging is a technology that enables high-speed, asynchronous, program-to-program communication with reliable delivery, the message itself is simply some sort of data structure such as a string, a byte array, a record, or an object.
A message is transmitted in five steps:

1. Create : The sender creates the message and populates it with data.
2. Send : The sender adds the message to a channel.
3. Deliver: The messaging system moves the message from the sender's computer to the receiver's computer, making it available to the receiver.

4. Receive: The receiver reads the message from the channel.

5. Process : The receiver extracts the data from the message

2.4.1 Mediation

The messaging system acts as a mediator between all the programs that can send and receive messages. An application can use it as a directory of other applications or services available to integrate with. If an application becomes disconnected from the others, it need only reconnect to the messaging system, not to all the other messaging applications. The messaging system can be used to provide a high number of distributed connections to a shared resource, such as a database. The messaging system can employ redundant resources to provide high-availability, balance load, reroute around failed network connections, and tune performance and quality of service [5].

2.4.2 Message-oriented middleware (MOM)

It is a specific class of middleware that supports loosely coupled communication between distributed software components by means of asynchronous message passing as opposed to a request/response metaphor. Message-oriented Middleware contains notification services supporting two message models:

Point to point messaging is built around the concept of a message queue which forms a virtual communication channel. Each message is sent to a specific queue and is retrieved and processed by a single consumer.

Publish/subscribe messaging is a one to many communication models. Each message is sent (published) by a message producer (publisher) to the notification service and it may be delivered to multiple consumers (subscriber) interested in the notification. Consumers are required to register (subscribe) for notification they are interested in before being able to receive messages.

ESB provides foundational services for more complex Service-Oriented Architectures (SOA) via an XML-based messaging engine (the bus), and thus provides an abstraction layer on top of an enterprise messaging system that allows integration architects to exploit the messaging without writing code. Also, it provides increased flexibility in changing systems as requirements change.

Since an ESB is a bus, it is topologically located at an intermediary position between two types of SOA participants: service requestors and service providers [6]. In message-oriented middleware platforms such as JMS, ActiveMQ, RabbitMQ, etc., the message queue is deployed as a server, and producers and consumers connect to the message queue through a protocol (e.g., AMQP, TCP, NIO, etc.) and exchange messages.

MOM provides asynchronous and loosely coupled communications. It supports both queue and topic (publish/subscribe) model of messaging. A message queue is a one-to-one communication between sender and receiver. This topology has limitation for active communication among multiple applications in heterogeneous systems. A better model publish/subscribe (or pub/sub) model, can easily solve this problem by extending MOM functionalities to support one-to-many, many-to-one and many-to-many communications. A pub/sub model normally consists of three basic elements publisher, subscriber and broker. An application can be either a publisher or a subscriber or can be both a publisher and a subscriber at the same time. Additionally, the number of publishers and subscribers can grow and shrink over time [48].

2.4.3 ESB as Hub node

This node supports the key ESB functions and, therefore, fulfills a large part of the ESB capabilities. The hub has a fundamental service integration role and should be able to support various styles of interaction. There are two interaction styles that the hub supports. Those styles are the Router and Broker interaction patterns. The Router interaction pattern is where a request is routed to a single provider. The Broker interaction pattern supports requests that are routed to multiple providers, including aggregation and disaggregation of messages. Figure 2:4 shows these interactions.

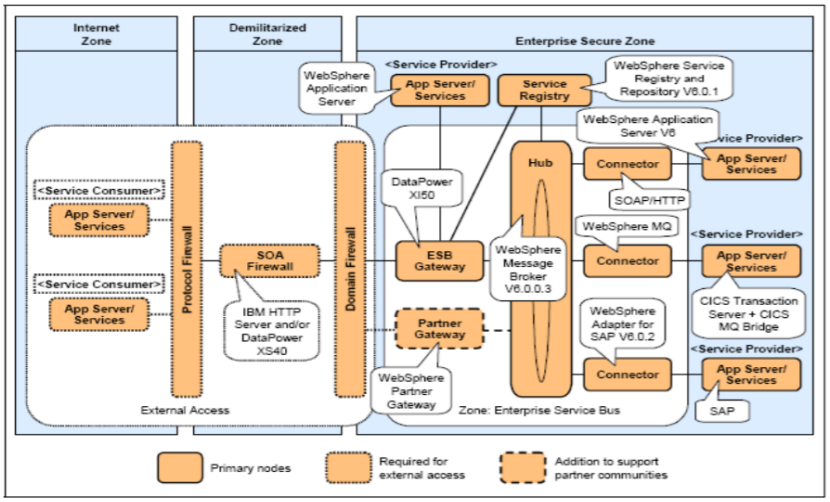


Fig 2:4 Basic ESB Scenario – Single ESB with WMB as the ESB with External Exposure[36]

The hub must contain rules for routing messages, and in the case of hubs that support the Broker interaction pattern, this node supports the key ESB functions and, therefore, fulfills a large part of the ESB capabilities. The hub has a fundamental service integration role and should be able to support various styles of interaction.

2.5 Service Oriented Architecture

Service-Oriented Architecture (SOA) is an architectural pattern to build application landscapes from single business components. These business components are loosely coupled by providing their functionality in form of services. A service represents an abstract business view of the functionality and hides all implementation details of the component providing the service. The definition of a service acts as a contract between the service provider and the service consumer. Services are called using a unified mechanism which provides a platform independent connection of the business components while hiding all the technical details of the communication. The calling mechanism also includes the discovery of the appropriate service [7]. Service-Oriented Architectures (SOA) is a distributed computing paradigm for developing large complex applications by composing software components running on heterogeneous platforms, which offer services to one another through well-defined interfaces. These services implement well-defined business functionalities offered by software components which can be reused for different applications. There are different supporting technologies (provided by service platforms) that help developers to implement systems, each providing different features for publishing, discovering

and invoking services [19]. The SOA paradigm describes how loosely coupled software components offer services in a distributed environment. SOA enables the integration of legacy applications and aims at increasing the flexibility of enterprises. However, one of the main concerns when implementing a SOA is the expected performance of the overall system, Figure 2:5 describes the SOA implementation.

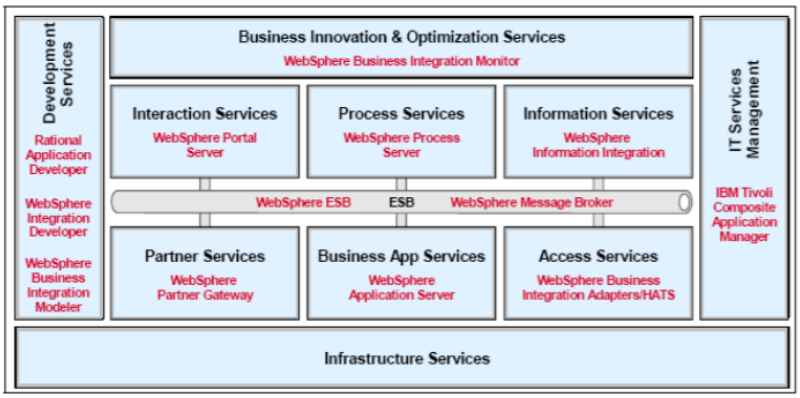


Fig 2:5 SOA Reference Architecture with Product Mapping[36]

Legacy systems found in enterprises are often not designed for this type of interaction. New access patterns and additional software layers therefore lead to different performance characteristics [20].

2.6 Web Service

Web services are connected programs to each other across distant points on the global map, transporting large amounts of data more efficiently and cheaply. It enables businesses to connect applications to other business applications, to deliver business functions to a broader set of customers and partners, to interact with marketplaces more efficiently, and to create new business models dynamically [1]. Web Services realize a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It is an approach to have software resources in an enterprise available and discoverable on network as well-defined services. Each service would achieve a predefined business objective and perform discrete units of work. Well-designed services are independent do not depend on the context or state of the other services. They work within distributed systems architecture, figure 2:6 basic web service Integration model [1].

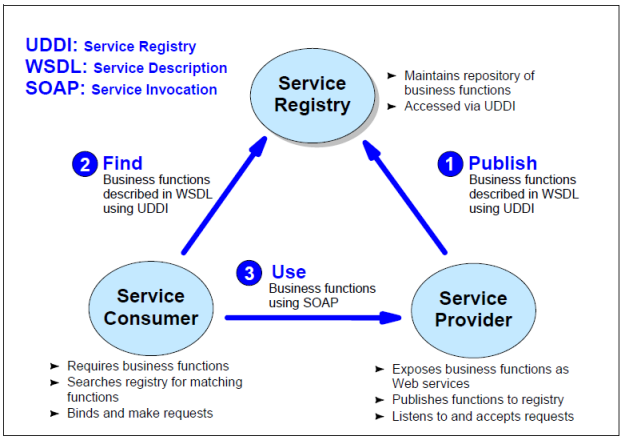


Fig 2:6 Basic Web Service Integration Model [1]

The interactions work as follows

1. The service provider publishes some WSDL defining its interface and location to a service registry.
2. The service consumer contacts the service registry in order to obtain a reference to a service provider.
3. The service consumer, having obtained the location of the service provider, makes calls on the service provider.

Web services and SOA reduce complexity of enterprise application eco-systems by encapsulation and minimizing the requirements for shared understanding by defining service interfaces in an unambiguous and transparent manner. Also, Web services enable just in time integration and interoperability of legacy applications. Web service Definition Language (WSDL): WSDL is an XML based language that describes the implementation of services. It is used both by the service provider and service requester. WSDL document includes information about the location of the services and their message format. UDDI is a web service used to register the description of services makes it available for other service consumers to discover. The core standards used in web-service implementation are SOAP, WSDL, UDDI, and application transfer protocols like HTTP, FTP, SMTP etc. But other standards used for web-service management and web-service security also exists, figure 2:7 shows the stack of web service technology [22].

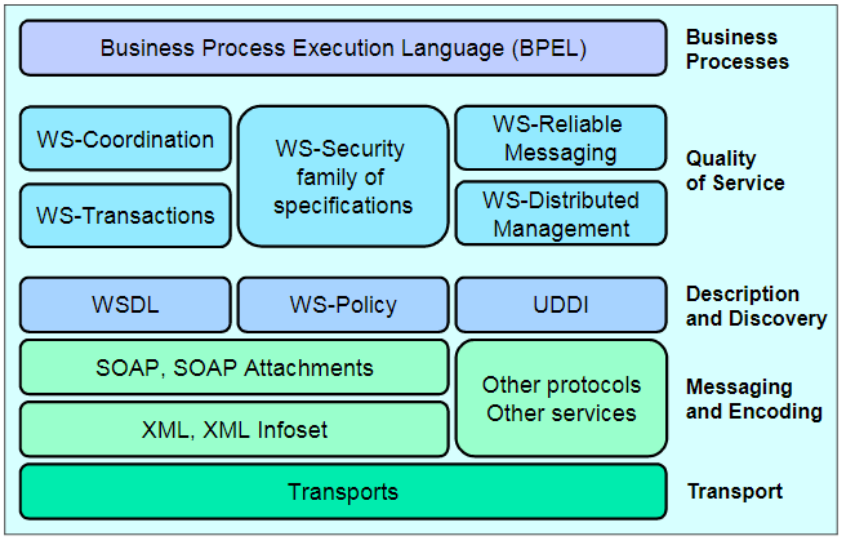


Fig 2:7 Stack view of Web services technology [37]

2.7 Software Performance

Software Performance is the degree to which a software system or component behavior fulfils its objectives for timeliness. The performance is described in term of requests made by the users and system responses, as well as usage of system resources (software and hardware) to serve the respective requests. Frequently used performance measures are response time and throughput. Response time is typically described from a user perspective and is defined as the time required by the system to serve a user's request. The throughput is the number of requests that can be processed per unit of time. The maximum throughput indicates the capacity of the system. The performance parameters are workload dependent. Workload intensities specify the level of usage of the system and can be open or closed. An open workload is usually specified by the arrival rate of the requests, and a closed workload by the number of concurrent users.

Performance is an essential attribute of every software system. The performance failure of a software system can cause various consequences including damaged customer relations, income loss, need for additional project resources, market loss, business failure, and project failure. To understand and define performance, we need to view a software system in three related ways [19].

- Scenarios: a scenario describes system behavior for a high-level operation by a sequence of sub-operations.
- Resource demands: we need to know how an operation loads the system resources.
- Architecture: the architecture describes the organization of objects and modules with interaction relationships.

2.7.1 Software Performance Engineering (SPE)

SPE is a systematic, quantitative approach to construct software systems to meet their performance objectives. It uses quantitative techniques to identify architectural, design, and implementation alternatives that will meet performance objectives. SPE is a model-based and software-oriented approach. Modeling is central to both SPE and object-oriented development. SPE is defined as a systematic, quantitative approach to constructing software systems that meet performance requirements [20].

According to the definition it is a set of six activities during system development,

1. Identify concerns
2. Define and analyze requirements
3. Predict performance from scenarios, architecture, and detailed design
4. Performance testing
5. Maintenance and evolution: predict the effect of potential changes and additions.
6. Total system analysis: consider the planned software in the complete and final deployed system.

2.7.2 Capacity Management: Continuous Capacity Planning

The term "capacity planning" is used to describe the activity of initially estimating the required capacity whereas capacity management describes the continuous application of "capacity planning" as soon as an EA is in production. In capacity management scenarios there are basically two ways of adding additional resources if additional capacity is required scaling horizontally (scaling out) and scaling vertically. Scaling horizontally means adding additional servers of the same type, whereas scaling vertically means adding additional resources to the existing servers or replacing existing resources with faster ones. To estimate the required capacity of an EA for a specific workload without the need to test it on a real system, performance models need to be used.

A prediction error under 30% is still acceptable for capacity planning of a system. Capacity is typically represented as the maximum number of messages or a byte count that can be transported between consumers and service endpoints. In this sense, capacity is analogous to **maximum throughput** capacity can also be influenced by or augmented with the other ESB performance criteria as well [51].

2.8 What Is a Queueing Network Model?

Queueing network modelling, the specific subject of this work, is an approach to computer system modelling in which the computer system is represented as a network of queues which is evaluated analytically. A network of queues is a collection of service centers, which represent system resources, and customers, which represent users or transactions. Analytic evaluation involves using software to solve efficiently a set of equations induced by the network of queues and its parameters [44]. A queueing system is a place where customers arrive according to an 'arrival process' to obtain service from a service facility. The service facility may contain more than one server (or more generally resources) and it is assumed that a server can serve one customer at a time. If an arriving customer finds all servers occupied, he joins a waiting queue. This customer will receive his service later, either when he reaches the head of the waiting queue or according to some service discipline. It leaves the system upon completion of his service.

2.8.1 Multiple Service Centers

Figure 2:8 shows a more realistic model in which each system resource (in this case a CPU and three disks) is represented by a separate service center. The service demand must be specified, but this time it should be provided a separate service demand for each service center. If customers are shown in the model as corresponding to transactions in the system, then the workload intensity corresponds to the rate at which users submit transactions to the system, and the service demand at each service center corresponds to the total service requirement per transaction at the corresponding resource in the system. Defining a queueing network model of a system is made relatively straightforward by the close correspondence between the attributes of queueing network models and the attributes of computer systems.

For example, service centers in queueing network models naturally correspond to hardware resources and their software queues in computer systems, and customers in queueing network models naturally correspond to users or transactions in computer systems.

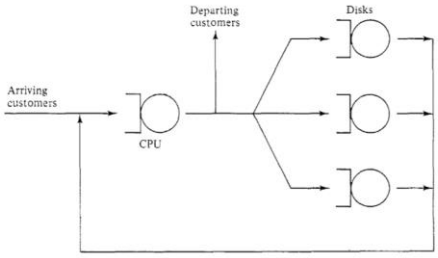


Fig 2:8 A Network of Queues[44]

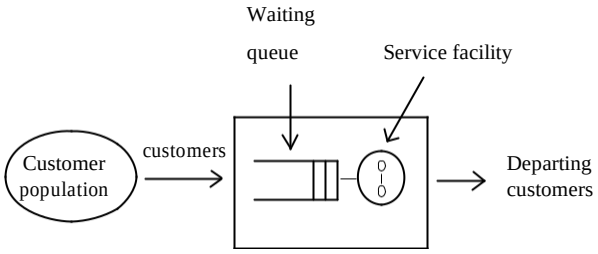


Fig 2:9 Schematic diagram of a queening sys[46]

2.8.2 Single Station Queuing Systems

Single station queuing models are useful for performance evaluation of many devices, sub-systems and communications nodes that make up communications networks. In a single station queuing model, the queuing system consists of a buffer for queuing customers and one or more identical servers. The queue may be of zero, finite or infinite size. A server may only serve one customer (packet, call, request, etc.) at a time and so, at any given time, it is either busy or idle. A customer represents some unit of work that keeps a server busy for some amount of time (e.g. transmission of a packet of some length, carrying of a call for some amount of time, processing of a request, etc.).

There are two sources of randomness in such a system:

1. Customers arrive at random times, that is, the inter-arrival-time of customers is described by a random variable.
2. The amount of time required to service a customer is random, that is, the service time is described by a random variable.

It is often assumed that arrivals are independent events and the service times of different customers are also independent. Kendall's Notation for Queues use the following notation to describe different types of queuing systems:

A/B/m/K/p - queuing discipline

A is the distribution of inter-arrival times and B is the distribution of service times.

The distributions A and B may be indicated as:

M - Exponentially distributed inter-arrival times or service times (where the M indicates Memoryless)

D - Deterministic distribution (that is, constant)

G - General distribution (of unknown distribution)

GI - General independent (unknown distribution but with independent arrivals and service times).

m - indicates the number of servers in the system.

K - indicates the capacity of the overall system (the number of queuing places plus the number of servers). If K is not specified in the Kendall notation, then the buffering capacity (queue length) is taken as infinite.

p - indicates the number in the population of customers that may arrive to the system, that is, the number of users of the system. If p is not specified in the notation, then the population is taken as infinite.

The queuing discipline determines which customer is selected from the queue for processing when a server becomes available. Examples of different queuing disciplines are:

FCFS - First Come-First-Served - If no queuing discipline is stated in the Kendall description, this one is assumed. All the systems we consider have a FCFS queuing discipline.

LCFS - Last-Come-First-Served. PS - Processor-Sharing- All customers are served simultaneously where processing of all customers is equally spread across all servers. For the distributions A and B above, the mean arrival rate is normally denoted as λ and the mean service time as 1/μ, that is, μ is the mean service rate [10]. Here below in figure 2:10, a description of single service stations. (w- queue waiting time, s- service time and t- total time)

Fig 2:10: Single service station with a single queue [20]

2.8.3 Closed Queuing Networks

A closed queuing network consists of several stations as illustrated in *Figure 2:11. f*or this structure the flow through each station is passed to other stations. It is a closed network because no flow enters from outside the network. The number of items present in the system is N, each component of the network is a single server queue with the service activity.

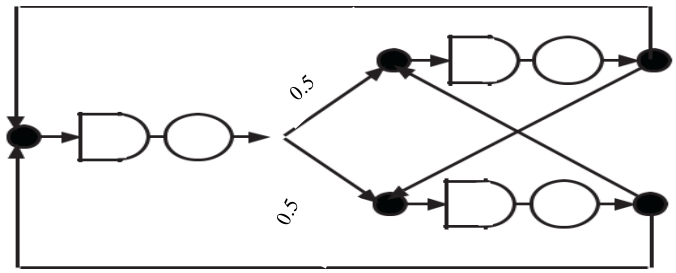


Fig 2:11 A closed network (the numbers, 0.5, example branching probability) [18].

For the closed network it is defined the following notation. *i* stands for the general station index and these quantities all depend on the number of items in the closed system.

$x_i(N)$ = the flow through station *i*. (arrival rate)

$n_i(N)$ = the expected number of items at station *i*.

$w_i(N)$ = the expected time at station *i*.

Little's Law

Number of requests in the system = arrival rate * mean response time
Number of requests in the queue = arrival rate * mean waiting time in the queue

- Relates the number of requests to the response time
- Valid for any type of queueing system
- Valid for systems in its entirety or for parts of the system

From Little's Law we can compute the expected numbers at the stations:

- $n_i(N) = x_i(N)\, w_i(N)$, for $i = 1 \ldots K$
- Mean time in system, $W = N/\lambda$
- Mean time in system queues, $W_q = Nq/\lambda$

2.8.4 Poisson Process

The process N(t) counts the number of events which occurred between time 0 and t. One assumes that N(0) = 0. Denoting T1, T2, . . . the times at which the event occurred, we have that the process N(t) increases with 1 at each time. N(t) = 0 if $t < t1$, N(t) = 1 if $t1 \leq t \leq t2$, . . . N(t) = k if $tk \leq t < tk+1$, etc. Arrivals of customers at a ticket office and Bus arrivals at a station are common examples for Poisson process. According to [46], queue systems can be characterized by its input and out processes as below,

Input Process,

1. **The size of arriving population**

The size of the arriving customer population may be infinite in the sense that the number of potential customers from external sources is very large compared to those in the system, so that the arrival rate is not affected by the size. The size of the arriving population has an impact on the queueing results. An infinite population tends to render the queueing analysis more tractable and often able to provide simple closed-form solutions, hence this model will be assumed for subsequent queueing systems. On the other hand, the analysis of a queueing system with finite customer population size is more involved because the arrival process is affected by the number of customers already in the system. Examples of the finite calling populations would be a group of stations in a local area network presenting data frame to the broadcast channel, or a group of video display units requesting response from a CPU.

2. **Arriving patterns**

The parameter that we commonly use to describe the arrival process is the inter-arrival time between two customers. We generally fit a probability distribution to it so that we can call upon the vast knowledge of probability theory. The most commonly assumed arriving pattern is the Poisson process whose inter-arrival times are exponentially distributed. The popularity of the Poisson process lies in the fact that it describes very well a completely random arrival pattern, and also leads to very simple and elegant queueing results.

Output Process,

1. **Queueing discipline or serving discipline**

Queueing discipline, sometimes known as serving discipline, refers to the way in which customers in the waiting queue are selected for service. In general, we have,

- First-come-first-served (FCFS)
- Last-come-first-served (LCFS)
- Priority
- Processor sharing
- Random

2. Service-time distribution

Different customers require different amounts of service times; hence we again use a probability distribution to describe the length of service times the server renders to those customers. The most commonly assumed service time distribution is the negative exponential distribution. Markovian (or Memoryless) imply exponential distributed service times , deterministic imply constant service times and Erlang of order K (E_k) service time distribution.

2.8.5 A Two-State/Birth-Death Poisson Process

The birth-death process is a special case of continuous time Markov process, where the states (for example) represent a current size of a population and the transitions are limited to birth and death. When a birth occurs, the process goes from state i to state i + 1. Similarly, when death occurs, the process goes from state i to state i − 1. It is assumed that the birth and death events are independent of each other. The birth-and-death process is characterized by the birth rate $\{\lambda_j\}i=0,...,\infty$ and death rate $\{\mu i\}i=0,...,\infty$, which vary according to state i of the system [38]. From paper [41], the description of the process is as follows,

The process sojourns (travels) in each state i for a random length of time following an exponential distribution with parameter $\lambda i + \mu i$. When leaving state i, the process enters state (i + 1) or state (i − 1). The motion is analogous to that of a random walk except that transitions occur at random

times rather than fixed time periods. Figure 2:12 and Figure 2:13 describes birth-death process.

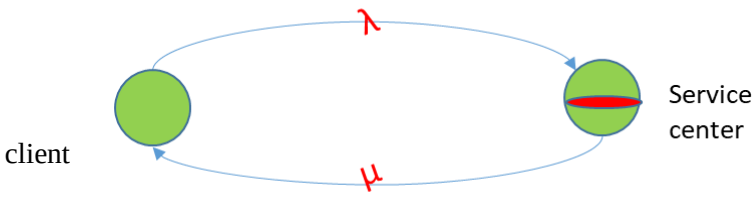


Fig 2:12 Describes birth-death process

Summary

In this Chapter, the core concepts that construct the performance predictive solution model are discussed in detail. EAI is the combination of processes, software, standards, and hardware technologies resulting in the integration of two or more enterprise systems allowing them to operate as seamless manner. ESB is the recent integration architecture to overcome the limitation of traditional architecture and realized the principle of SOA. Performance is an essential attribute of every software system. The performance failure of a software system like ESB can cause various consequences including damaged customer relations, income loss. SPE is a systematic, quantitative approach to construct software systems to meet their performance objectives. To estimate the required capacity of an EA for a specific workload without the need to test it on a real system, performance models need to be used. As per the papers [56] & [51] a prediction error under 30% is still acceptable for capacity planning of a system. Queueing network modelling, the specific subject of this work, is an approach to computer system modelling in which the computer system is represented as a network of queues which is evaluated analytically.

Chapter Three: Related Works

3.1 Introduction

This chapter review a few research papers that are related with the main concept of the research they are called ESB performance measuring & modeling, software Performance and Software performance Engineering (SPE). Mainly discussing implementation of ESB performance analysis and prediction in publish/subscribe communication. Finally, it will be discussed an idea that related with this research and point out the limitation of the researches.

3.2. ESB Performance Measurement

Performance is an important non-functional requirement of an IT system and is vital to the acceptance and the operability of the system. Since performance tests are usually not performed until the system is already in place, performance issues are often revealed at a late stage in the development process. In order to improve the performance of the system extensive changes to the architecture are needed which ultimately leads to significant project risks. As the introduction of SOA can deteriorate the performance it is crucial to be aware of the performance drawbacks and how to address them properly at the stage of the system design. Real time applications and bulk data processing demands a high-performance implementation. Current approaches to implement an SOA using web service technologies and infrastructures do not match very well with a batch-processing model since they are focused on a request-response communication scheme [24]. Dealing with the performance of services integrated by ESBs falls into the second category of SOA performance issues, and it is far more complex than atomic services. Services registered to an ESB can be uniquely identified by their endpoints. One challenging issue is that these performance characteristics of ESBs are hard to measure by instrumentation or monitoring due to the nature of highly distributed architecture in SOA. This necessitates a systematic approach that facilitates estimating performance characteristics of ESBs and analyzing the performance relationship between the ESB and the services it composes. This paper [25], recommends “a combined performance modeling and benchmark approach to address the above problems. A queueing network performance model is devised to represent the performance critical ESB

infrastructure components and the services running on the ESB. The performance characteristics of these ESB components are estimated by benchmarking results" [25].

3.3. Queueing Petri nets

A Petri net is an oriented graph with two kinds of nodes: places and transitions. These two are commonly represented as circles and rectangles each. The arches connect the places and the transitions or vice versa, the arcs will show the path that the marks will follow in the model. Each place can contain either a positive number of marks or zero marks. Each mark will represent information depending on the model that has been created,

A mark is called token and a token could represent a state in the process, a unit of time or a resource that can be used, among others. The marking specifies several tokens in each place. The tokens are used in modelling the transitions, they move from the places towards ingoing arcs to the transition node, and from there via outgoing arcs to the next place. Transition can have several input and output nodes or places which are interpreted as pre - input data, input signals, logic conditions, buffers, resources needed, and post conditions are output data, output signals, logic conclusions, resources released. The event corresponded to this transition can be computation step, processor's signal, logic clause, CPU cycle. The token held in the place reflects the evaluation of the corresponding condition to the logical true. Transitions are events or actions which cause the change of state. A transition is enabled if all input nodes have the number of tokens equal to or greater than those needed for firing the transition. This is, if a transition has 2 arcs that aim to it, then it should have at least two tokens, each one of them going through each arc [45].

Queueing Petri nets can be a combination of several different extensions to conventional Petri nets along several different dimensions. An ordinary Petri net (also called place-transition net) is a bipartite directed graph composed of places, drawn as circles, and transitions, drawn as bars. A formal definition is given below [9],

Definition 1 *(PN)*

An ordinary Petri net (PN) is a five-tuple PN = (P, T, $I-$, $I+$, M_0) where

1. P is a finite and non-empty set of places ($p1, p2, \ldots, p\,|P|$),

2. T is a finite and non-empty set of transitions ($t1, t2, \ldots, t\,|T|$),

3. $P \cap T = \emptyset$,

4. $I-, I+: P \times T \rightarrow N_0$ are called backward and forward incidence functions, respectively,

5. $M_0: P \rightarrow N_0$ is called initial marking.

The incidence functions $I-$ and $I+$ specify the interconnections between places and transitions. If $I-(p, t) > 0$, an arc leads from place p to transition t and place p is called an *input place* of the transition. If $I+(p, t) > 0$, an arc leads from transition t to place p and place p is called an *output place* of the transition. The incidence functions assign natural numbers to arcs, which we call *weights* of the arcs. When each input place of transition t contains at least as many tokens as the weight of the arc connecting it to t, the transition is said to be *enabled*. An enabled transition may *fire*, in which case it destroys tokens from its input places and creates tokens in its output places. The amounts of tokens destroyed and created are specified by the arc weights. The initial arrangement of tokens in the net (called *marking*) is given by the function $M0$, which specifies how many tokens are contained in each place.

Different extensions to ordinary Petri nets have been developed in order to increase the modeling convenience and/or the modeling power. *Colored Petri nets* (CPNs) introduced by Jensen [9] are one such extension. This allows a *type* (*color*) to be attached to a token. A color function C assigns a set of colors to each place, specifying the types of tokens that can reside in the place. In addition to introducing token colors, CPNs also allow transitions to fire in different *modes* (*transition colors*). The color function C assigns a set of modes to each transition and incidence functions are defined on a per mode basis.
A formal definition of a CPN follows [9]:

Definition 2 *(CPN)*

A colored Petri net (CPN) is a six-tuple CPN = ($P, T, C, I-, I+, M0$) where

1. P is a finite and non-empty set of places ($p1, p2, \ldots, p_{|P|}$),

2. T is a finite and non-empty set of transitions ($t1$, $t2$, . . ., $t_{|T|}$),

3. $P \cap T = \emptyset$,

4. C is a color function defined from $P \cup T$ into non-empty sets,

5. $I-$ and $I+$ are the backward and forward incidence functions defined on $P \times T$, such that $I-(p, t)$, $I+(p, t)$: $C(t) \rightarrow C(p)$MS, $\forall (p, t) \in P \times T$,2

6. $M0$ is a function defined on P describing the initial marking such that $M0(p) \in C(p)$MS, $\forall p \in P$.

QPNs can be used to combine qualitative and quantitative system analysis. Several efficient techniques from Petri net theory can be exploited to verify some important qualitative properties of QPNs. The latter not only help to gain insight into the behavior of the system but are also essential preconditions for a successful quantitative analysis. A number of extensions are combined in Colored, Generalized, Stochastic Petri Nets (CGSPNs), which add advanced timing control and typing of messages. None of the extensions are suitable to represent scheduling strategies, the specialty of queueing networks. To overcome this Queueing Petri Nets (QPN) include some ideas from queueing networks. In QPNs the places of CGSPNs are extended by adding a queueing place, which consists of multiple service stations and a depository for the storage of tokens (requests in queueing networks) that have been serviced. While the readability of the models is reduced by the QPN extension, it does increase the expressiveness of the models [50].

Definition 3 *(QPN)*

A queueing Petri net (QPN) is an eight-tuple QPN = (P, T, C, $I-$, $I+$, $M0$, Q, W) where

1. CPN = (P, T, C, $I-$, $I+$, $M0$) is the underlying colored Petri net.

2. Q= ($Q1$, $Q2$, ($q1$, . . ., q $|P|$)) where

• $Q1 \subseteq P$ is the set of timed queueing places,

• $Q2 \subseteq P$ is the set of immediate queueing places, $Q1 \cap Q2 = \emptyset$ and

• q_i denotes the description of a queue taking all colors of $C(p_i)$ into consideration if pi is a queuing place or equals the keyword ‘null’ if pi is an ordinary place.

3. W= (W1, $W2$, ($w1$, . . ., w $|T|$)) where

• $W1 \subseteq T$ is the set of timed transitions,

• $W2 \subseteq T$ is the set of immediate transitions, $T = W1 \cup W2$, $W1 \cap W2 = \emptyset$ and

• $w_i \in [C(t_i) \rightarrow$ R+] such that $\forall c \in C(t_i)$: $wi(c)$ is interpreted as the rate of a negative exponential

distribution specifying the firing delay due to color *c* if t_i is a timed transition or a weight specifying the relative firing frequency due to color *c* if t_i is an immediate transition.

Here below a related paper [57], which uses a queueing Petri net (QPN) modeling approach, it considers an automobile manufacturing company that plan to use e-business technology to support its order-inventory, supply-chain and manufacturing operations, it presents a practical performance modeling methodology in a step-by-step fashion showing how each step is applied to the considered scenario in figure 3:1 below. The objective was to predict the performance of the system under peak operating conditions.

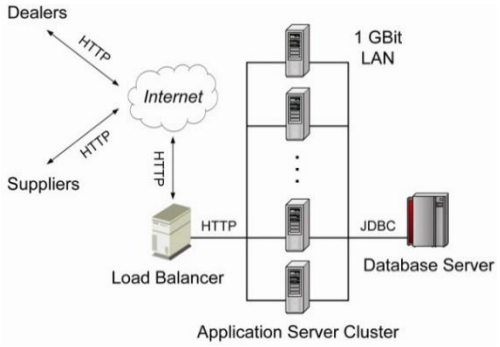


Fig 3.1 Deployment environment [57]

Table 3.1 System component details [57]

Component	Description
Load Balancer	Commercial HTTP Load Balancer 1 x AMD Athlon XP2000+ CPU 1 GB RAM, SuSE Linux 8
App. Server Cluster Nodes	WebLogic 8.1 Server 1 x AMD Athlon XP2000+ CPU 1 GB RAM, SuSE Linux 8
Database Server	Oracle 9i Server 2 x AMD Athlon MP2000+ CPU 2 GB RAM, SuSE Linux 8
Local Area Network	1 GBit Switched Ethernet

A QPN model of the system under study was built and then customized to the concrete configurations of interest. All token service times at the queues of the model are assumed to be exponentially distributed. Composite transactions are modeled using the following token types (colors): 'B' for Browse, 'P' for Purchase, 'M' for Manage, 'W' for Work Order and 'L' for Large-Order. For the sake of compactness, the following additional notations were used:

Symbol 'D' will denote a 'B', 'P' or 'M' token, i.e., token representing a dealer transaction.
Symbol 'd' will denote a 'b', 'p' or 'm' token, i.e., token representing a dealer sub transaction.
Symbol 'o' will denote a 'b', 'p', 'm', 'w' or 'l' token, i.e. token representing a sub transaction of arbitrary type. Upper-case tokens represent transactions, whereas lower-case tokens represent sub transactions. In the initial marking, tokens exist only in the depositories of places C1 and C2, and

other components of the model described further in figure 3:2 as shown below, other detail of the work was omitted to save space of this document and to give only an insight on QPN definition.

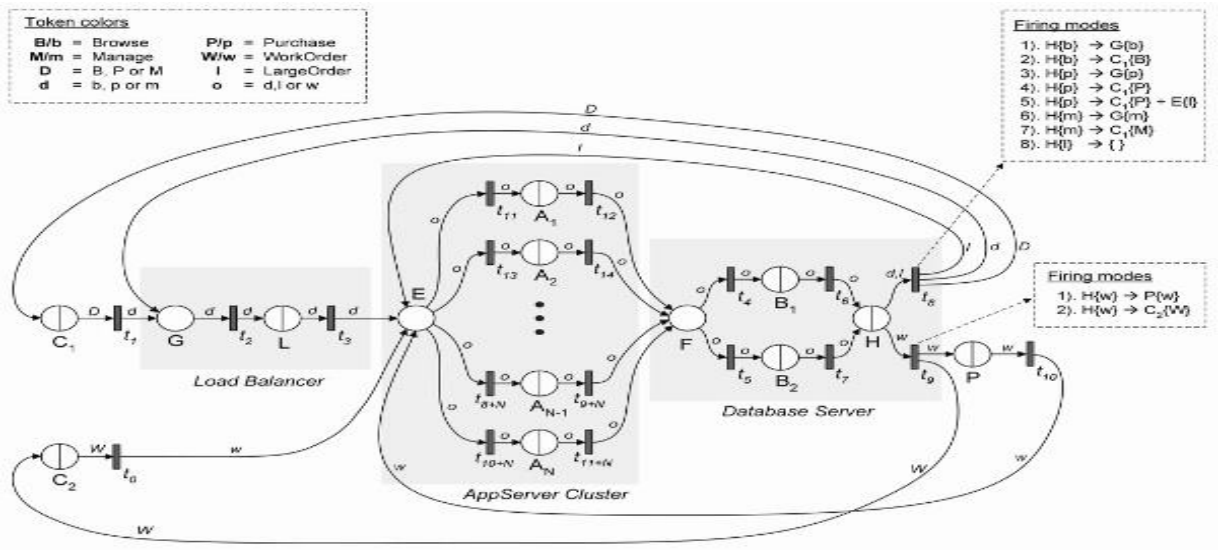


Fig 3:2 QPN model of the system [57]

All transitions of the model are immediate. Their firing modes, except for transitions t0, t1, t8 and t9, are defined in such a way that whenever they fire they simply move a token from their input place to their output place. The model was validated by comparing its predictions against measurements on the real system. The model predictions were verified for a number of different scenarios under different transaction mixes and workload intensities. The model was analyzed by means of simulation using SimQPN. For all metrics, the error of estimates was less than 20.6% of the respective mean value.

Summary

This chapter discussed on the papers which are related with the main concept of the research. ESB related performance problems of a company like ethiotelecom, the queue petri net modeling technique can be a foundation for performance prediction works, these models can be used to represent the performance of critical ESB infrastructure components and the services running on the ESB.

Chapter Four: The Proposed Solution Approach

4.1 Introduction

Numerous performance modeling approaches have been proposed to evaluate the performance (i.e., response time, throughput, resource utilization) of enterprise applications. These models can be used as input for analytical solvers and simulation engines to predict performance. Performance models are especially useful when scenarios need to be evaluated that cannot be tested on a real system. Scaling a system up or down in terms of the available hardware resources (e.g., number of CPU cores) are examples for such scenarios [55].

The performance characteristics of an ESB are mainly determined by its capacity for content-based routing and message transformation. Extra ESB infrastructure functions such as message flow monitoring and message security management can also incur additional overhead. In this thesis, it will be focused on the performance overhead of ESB routing and transformation. This can simplify the performance analysis approach and establish a model, which can be further extended to incorporate more complex ESB features such as monitoring and security management. The Chapter is organized as follows: Section 4.2 Methods used, Section 4.3 ESB Implementation in ethio telecom, Section 4.4 The Business Case and Section 4.5 Proposed Techniques.

4.2. Method

Interviews and company document reviews were done to investigate the existing system environment. Similar to the production environment of the company's ESB infrastructure, a small-scale system is established, and workloads were generated to this system to observe the impact of message traffic growth. Model simulation results were compared with results obtained from Jmeter measurements of the small scale system. Confidence intervals for the sample mean are an interval estimate of the real mean. Interval estimates are usually desirable, since the estimate of the mean varies from sample to sample. For both the measurement and simulation results, the output data was checked for the confidence intervals weather they are valid or not, and the minimum confidence value was 95% or not. To keep this result, enough number of samples were used during

each measurement steps. Document review also was conducted and a specific SIM card replacement process was selected.

4.3. ESB Implementation in ethio Telecom

Ethio telecom is moving towards becoming more efficient operationally and launching new services by collaborating across traditional marketing and IT departments to become more responsive towards its customer's needs. The company faces an increased pressure to form an effective service delivery. The service delivery is not only meant to bring public services effective but is also focused mainly in reducing overall operational costs by transforming the service delivery into an organization that generates both social and economic values effectively. This involves implementation of various information systems such as management support systems, operations support systems (OSS), transaction processing systems, enterprise resources management system (ERP), electronic messaging and collaboration system, and National electronic single window system for various billing services. The company has been started GTP programs investing in telecom infrastructure with different vendor specific infrastructure and applications of technologies. This multivendor program introduces new heterogeneous technologies with a complexity of service integrity and quality assurance challenges. The company has implemented BSS/OSS and other electronic systems in various times, including the ERP system to integrate business process and automate resource management.

There was a challenge in the integration of these systems due to their heterogeneity and lack of interoperability brought about by the diverse data formats, program languages, data storage methods and vendor's system access. Before some years, the company implemented ESB to address the integration issue.

The company's integrated services on the ESB:

- ✓ Business applications
- ✓ Banks
- ✓ Credit services
- ✓ Many web-based services and

Some of the functions that are activated on the ESB are:

- Message Routing
- Message Mapping
- Message Forwarding
- Message Transformation

4.3.1 ESB Interface and Integration

Figure 4:1 shows the interface and integration of ESB in ethio telecom. It has an interface from business applications (CBS, CRM and IPCC) and third-party systems. Most integrated applications are internal, and some are external. For example, bank system is an external system, CBS, CRM and IPCC are business applications, but the remaining systems are from third party.

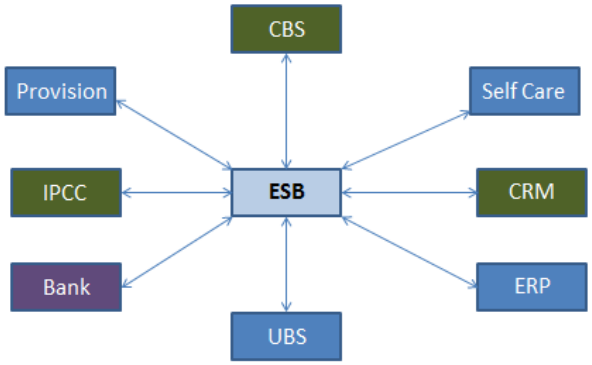


Fig 4:1 ESB Interface [1]

Successfully implementing SOA requires applications and infrastructure that can support the SOA principles. Applications can be enabled by creating service interfaces to existing or new functions that are hosted by the applications. The service interfaces should be accessed using an infrastructure that can route and transport service requests to the correct service provider. As organizations expose more and more functions as services, it is vitally important that this infrastructure should support the management of SOA on an enterprise scale [2].

4.3.2 Routing with message flows

Routing a message involves sending an incoming message to a destination based on some criteria. The destination can be predefined (static) or based on information that is obtained at the time of the message flow (dynamically). A common routing pattern includes a dynamic lookup of the destination based on the incoming message type and routing the message to that destination. This routing pattern typically consists of the steps shown in Figure 4:2.

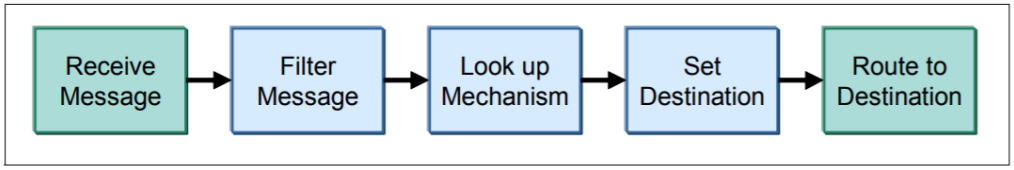


Fig 4:2 A typical routing mechanism

The following alternative ways are used to implement routing in message flows and the nodes that are available to achieve it,

- Routing using Java Compute Out and Alternate terminals
- Routing using the RouteToLabel node
- Routing use a database

Using a database table is the recommended way to store any form of routing or transformation data for the following reasons [42]:

- It removes business data from the ESQL or Java code in the message flow.
- It allows for updates in the routing data without message flows having to be changed and redeployed.
- It makes ESQL or Java easier to read by removing the need to have large If ... then ... else type of code structures.
- It allows routing information to be stored in one place but used by many different message flows and possibly other applications

Figure 4:3, use a database as a means to find routing information:

- Directly accessing the database table

Fig 4:3 Database lookup of routing information[42]

- Figure 4:4, using an in-memory cached version of the database table

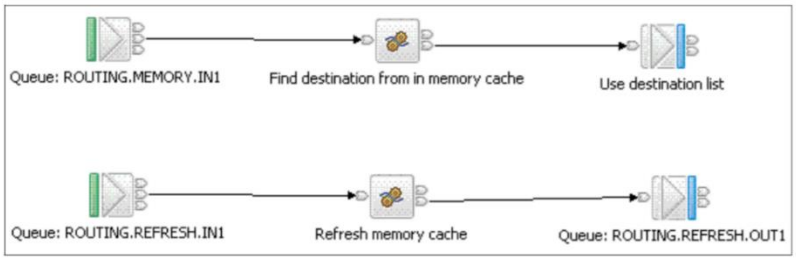


Fig 4:4 Routing with lookup using shared variables and refreshing of cache [42]

4.4. The Business Case

Data-driven models show the sequence of actions involved in processing input data and generating an associated output. In the study of this research, it is reasonable to consider one of the operational processes of the company, SIM card sales business process is selected which is one of the processes related to the ESB's enterprise applications integration implementation in the company.

Some of the reasons to consider SIM card sales business process in the study,

- The process completion time has direct impact on the company's service delivery
- The process has been impacted by ESB performance and
- Many applications involved and interact via ESB to complete this process

Figure 4:5, covers the description and components involved in SIM card replacement business process.

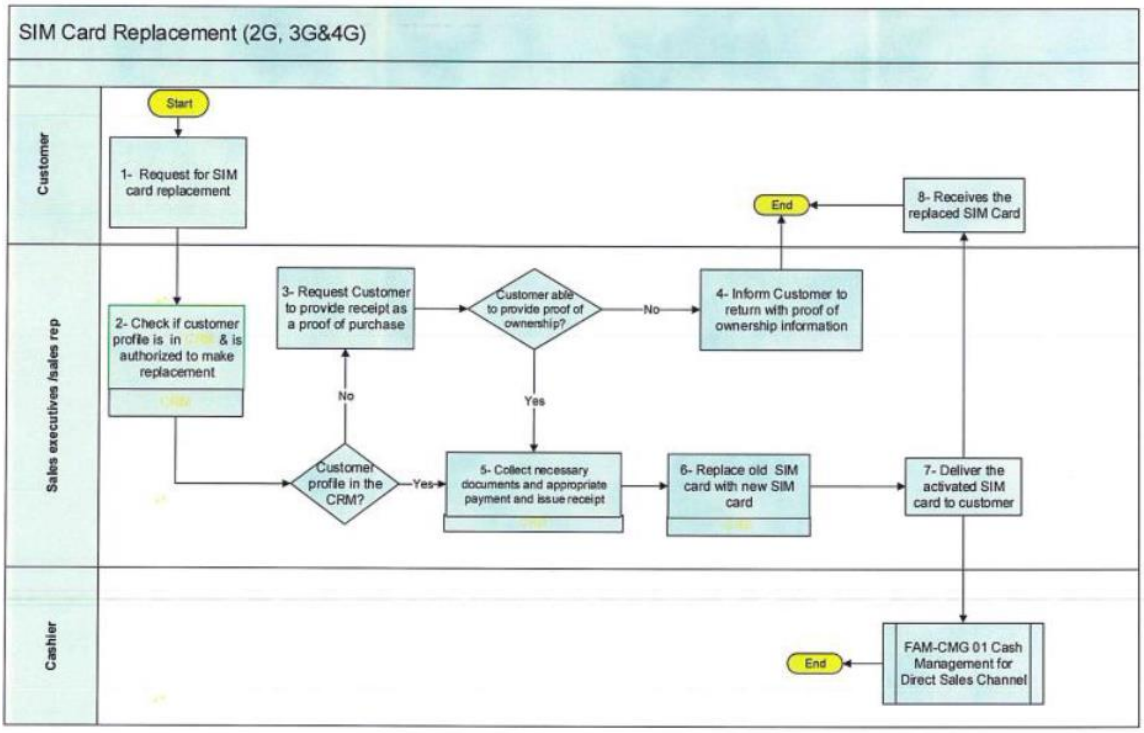
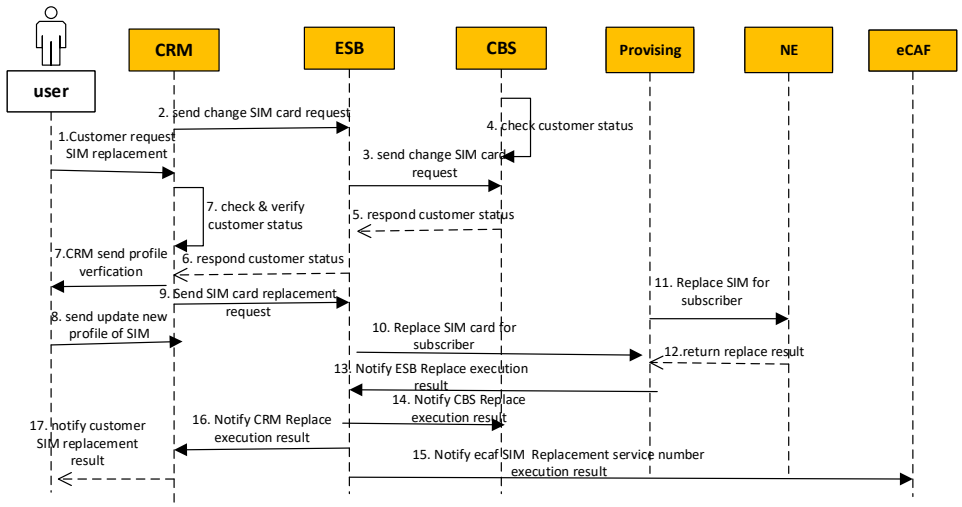


Fig 4:5 SIM card replacement process activity diagram [35]

The activity diagram in Figure 4:5 above shows the SIM card replacement business process implemented in ethiotelecom sales shops. The activity diagram is based on ethiotelecom SIM card replacement business process and detail can be found in the sales manual document.

4.4.1 SIM Card Replacement Process sequence diagram



4:6 SIM Card Replacement Sequence diagram [22]

The process steps in sequence diagram in Figure 4:6 are,

1. Customer in the sales office requests a SIM card replacement.
2. CRM sends the SIM card replacement request to the ESB
3. ESB sends the SIM card replacement request to CBS
4. CBS checks customer status
5. CBS send the response to ESB
6. ESB sends the response to CRM
7. CRM captures and validates the response
8. The customer profile is updated with new SIM information.
9. CRM sends the replaced SIM information to ESB
10. ESB sends replace SIM request to Provisioning
11. Provisioning sends SIM replacement request to NE
12. NE returns SIM replacement result to provisioning
13. Provisioning sends the SIM replacement result to ESB
14. ESB notify the SIM replacement to CRM
15. ESB notify SIM replacement information to CBS
16. CRM confirms the SIM replacement to the customer

4.4.2 Publish /Subscribe Scenario in SIM Card Replacement Process

The interacting applications taken from Figure 4:6,

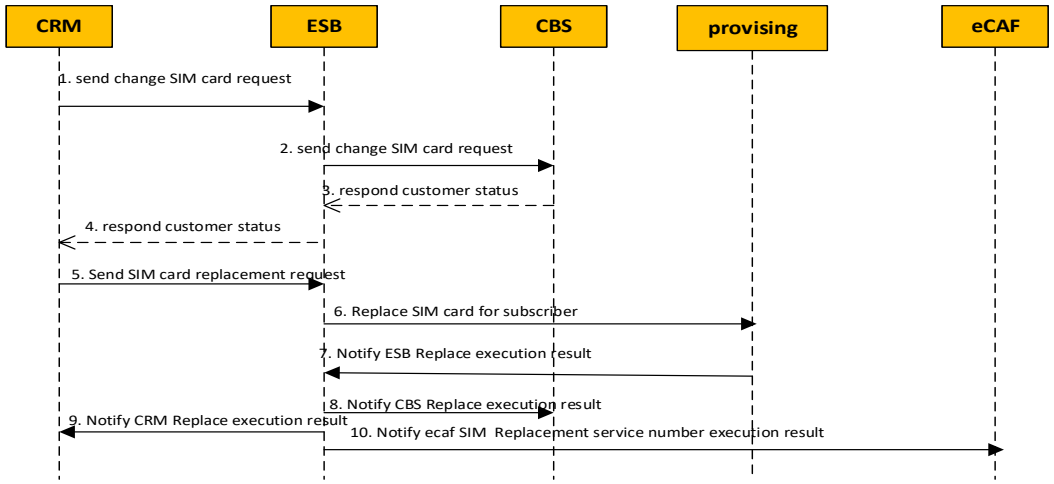


Fig 4:7 ESB & Application's interactions on SIM Card Replacement Sequence diagram

From Figure 4:7, four applications publish/subscribe interactions,

1. CRM sends the SIM card replacement request to the ESB
2. ESB sends the SIM card replacement request to CBS
3. CBS send the response to ESB
4. ESB sends the response to CRM
5. CRM sends the replaced SIM information to ESB
6. ESB sends replace SIM request to Provisioning
7. Provisioning sends the SIM replacement result to ESB
8. ESB notify the SIM replacement to CRM
9. ESB notify SIM replacement information to CBS

4.5 Proposed Technique

As indicated in the problem statement (Section 1.2) of this research document, performance problem and business risk were identified. To solve or manage the performance problem and the performance bottleneck risk, a performance predictive model is proposed. A generic distributed event-based system (DEBS) is normally composed of a set of nodes deployed in a distributed environment and exchanging information through a set of JNDI communication infrastructure. Clients of the system are either publishers or subscribers depending on whether they act as producers or consumers of information. Publishers publish information in the form of events which are commonly structured as a set of attribute-value pairs. Subscribers express their interest in specific events through subscriptions. The detail concepts for the proposed technique is presented in below sections.

4.5.1 Communication Pattern Selected

Here for this study, a publish/subscribe communication pattern is chosen to investigate the performance of the ESB.

Publisher/Subscriber Scenario

Simple operation an application puts (publishes) a message to the broker, another application gets (subscribes to the topic) the message. Publish/Subscribe is a configuration of messaging application in which the providers of information (publishers) have no direct link to specific consumers of that information (subscribers), but the interactions between publishers and subscribers are controlled by pub/sub brokers. A typical publish/subscribe system has more than one publisher and more than one subscriber. In Figure 4:8 and Figure 4:9, communication steps are shown,

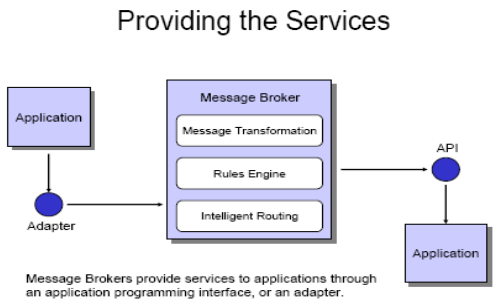


Fig 4:8 ESB as message broker [17]

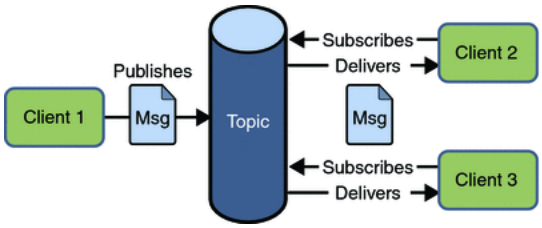


Fig 4:9 publisher/subscriber messaging using topics [21]

Publish/Subscribe Messaging Application Flow

- The publisher generates a message that it wants to publish and defines the topic of the message.
- A subscriber registers a request for a publication by specifying the topic (or topics) of the published messages that it is interested in.
- Subscriptions to an MQ pub/sub broker might include the following information:

    - The subscription point from which it wants to receive publications.
    - The content filter that should be applied to the published message.
    - The name of the queue/topic (known as the subscriber queue) on which publications that match the criteria selected are placed. This queue can be a cluster queue, so that publications can be distributed to clustered subscribers.

4.5.2 Performance Models

Performance is a quality attribute of a software system and is crucial to the acceptance of a developed system both by users and IT operations. A performance model captures the performance characteristics of the entire system: software, operating system, middleware, and hardware, as it captures all the system resources and the competing demands made on resources when executing different scenarios under different workloads. The results of a performance model are system-level measures such as end-to-end delay, response time, throughput, utilization, etc. There are many performance-modeling methods and tools available, based on different formalisms such as queuing networks, stochastic Petri nets, stochastic process algebra, and stochastic automata. All formalisms can be used for modeling software systems. Two categories of performance modeling approaches are analytical and simulation techniques. The paper [50], summarizes the difference between the two as: "Analytical techniques use theoretical models. Simulation techniques emulate the functionally of the application using a computer simulation whose performance can be probed". However, no clear distinction can be made between simulation and analytics. Some constructed models may be evaluated both analytically and by using simulation.

Compared to simulation, analytical techniques require more simplifications and assumptions, and this is especially true for models employing queues. Analysis techniques for performance models that are directly based on the states and transitions of a system generate a state space. Often these techniques cannot be used to model systems of realistic size due to the so-called state space explosion problem, in which the set of states to be analyzed grows exponentially with the size of the model. On the upside there are improvements to numerical solution methods for state spaces and the approximations they use. Simulation techniques allow for more detailed studies of systems than analytical modeling. However, building a simulation model requires both strong software development skills and comprehensive statistical knowledge. Also, simulation models often require (much) more time to develop than analytical models [50].

The performance of a system can be described by multiple metrics. The following metrics are relevant to the understanding of this paper:

Response time Specifies the amount of time the user must wait to get the response from the application. In terms of performance testing, this is the time between the user's requesting response

from the application and a complete reply, arriving at the users workstation. This information collected using the model for configuration scenarios with additional message traffics because of the growth of new integration .

Throughput Defines the number of things we can accomplish in a time period. In JavaEE terms, request throughput is several requests a system can service in a second. The goal is to maximize request throughput and to measure it against the number of simultaneous requests. Request throughput reflects the response time.

Resources utilization defines the percentage of the capacity of the resource is being used by the application. This result help to analyze the work of application and help to find the root cause of performance degradation, if any. The most important metrics are CPU usage and memory usage. Among others for Java applications are thread pools, JDBC connection pools, caches, JMS servers and others [43].

The integrations growth is defined in terms of more publishers and subscriber applications and investigating the impact of number of messages arriving to the modeled ESB system. After model simulation the performance information can be collected for different integration scenarios and possible message traffic growths. These metrics can be easily generated using the model instead of instrumenting the real ESB operational environment.

4.5.3 The Modeling Process

A model is an abstraction of a system, an attempt to distill, from the mass of details that is the system itself, exactly those aspects that are essential to the system's behavior. Once a model has been defined through this abstraction process, it can be parameterized to reflect any of the alternatives under study, and then evaluated to determine its behavior under this alternative. Using a model to investigate system behavior is less laborious and more flexible than experimentation, because the model is an abstraction that avoids unnecessary detail. It is more reliable than intuition, because it is more methodical, each approach to modelling provides a framework for the definition, parameterization, and evaluation of models. Of equal importance, using a model enhances both intuition and experimentation. Intuition is enhanced because a model makes it possible to "pursue hunches" - to investigate the behavior of a system under a wide range of alternatives. (In fact, to

devise quantitative models, which accurately reflect the performance measures of a system, an equally effective guide to intuition can be provided by less detailed qualitative models, which accurately reflect the general behavior of a system but not necessarily specific values of its performance measures) [51].

The first step of a model-based performance evaluation consists of the formalization process, during which the modeler generates a *formal* description of the real-world system. If the system is to be represented as a queueing network, the modeler applies a routed job "flow" modeling paradigm in which the real-world system is conceptualized as a set of service stations which are connected by edges through which independent entities " flow" through the network and travels in the queues and servers of the service stations. Figure 4:10, below describes model-based performance evaluation process.

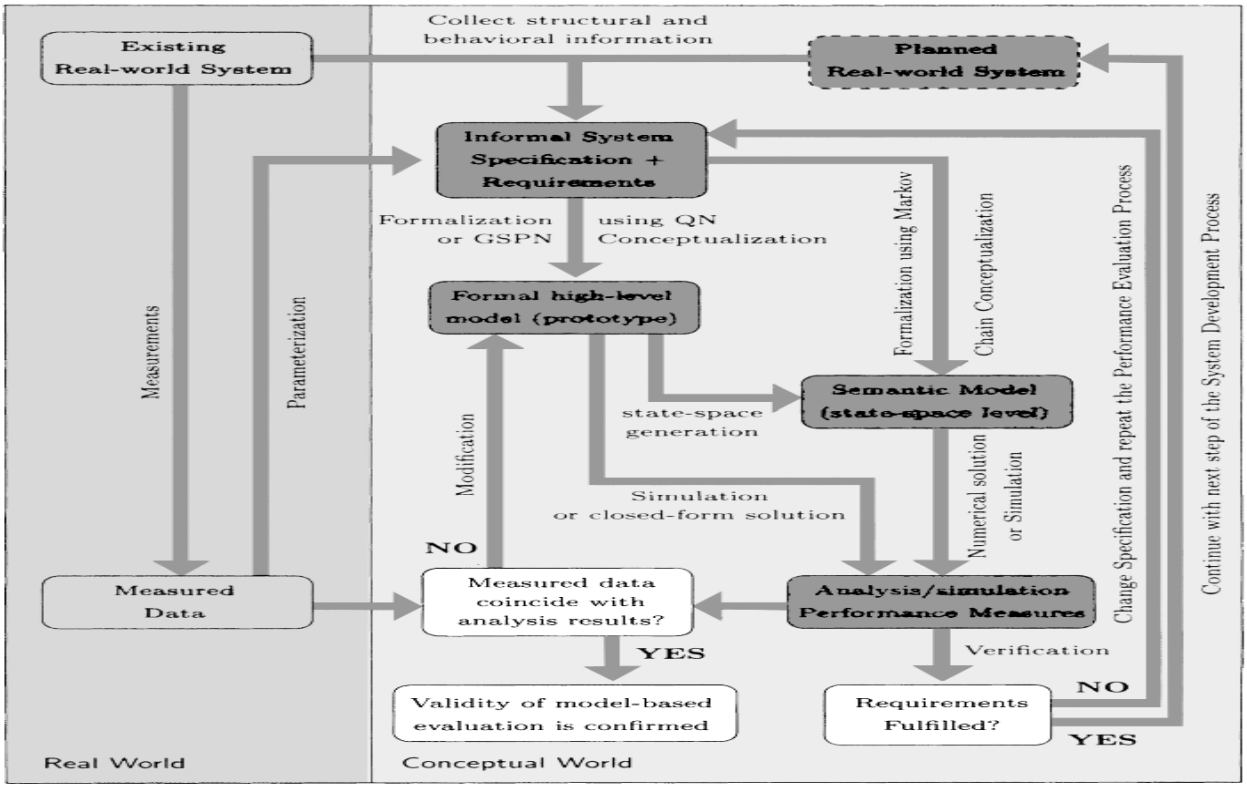


Fig. 4:10 Application of model-based performance evaluation [15]

The software performance engineering (SPE) process includes the following nine steps,

1. Assess performance risk
2. Identify critical use cases
3. Select key performance scenarios
4. Establish performance objectives
5. Construct performance models
6. Determine software resource requirements
7. Add computer resource requirements
8. Evaluate the models
9. Verify and validate the models

4.5.4 Performance Measurements

ESB system is an integration backbone system consisting of different integration of applications and services, number of **concurrent users** and configuration scenario are inputs of the set-up environment. CPU, I/O and memory are performance factors to be measured. Concurrency is the ability to run several programs or several parts of a program in parallel. It enables a program to achieve high performance and throughput by utilizing the untapped capabilities of underlying operating system and machine hardware. e.g. modern computers have several CPU's or several cores within one CPU, program can utilize all cores for some part of processing; thus, completing task much before in time in comparison to sequential processing. The backbone of java concurrency are threads. A thread is a lightweight process which has its own call stack but can access shared data of other threads in the same process. A Java application runs by default in one process. Within a Java application you can work with many threads to achieve parallel processing or concurrency [33]. According to Woodside the approaches to performance engineering can be

divided into two categories, measurement-based and model-based. The former is most common and uses testing, diagnosis and tuning once a running system exists that can be measured. It can thus only be used towards the end of the software development cycle. The model-based approach focusses on the earlier development stages instead and pioneered with the Software Performance Engineering (SPE) approach. Like the name suggests, in this approach models are key to make quantitative predictions on how well an architecture may meet its performance requirements. Performance testing applies measurement-based techniques and is usually done only after functional and load testing. Load tests check the functioning of a system under heavy workload. Whereas performance tests are used to obtain quantitative figures on performance characteristics, like response time, throughput and hardware utilization, for a system configuration under a defined workload [50].

Estimation of the Event Service Times: to determine the service times of events at the system resources includes all resources used by the system to deliver published events. The two types of resources are the CPUs of the system and the network nodes, secondary storage devices at the system nodes (e.g., disk drives) might have to be considered if they are used by the event-based middleware. There are different approaches to estimate the service times of events at the CPU of a system node. First, the system node can be instrumented to directly measure the CPU usage when processing events. Another approach which does not require instrumentation is to estimate the service times based on measured CPU utilization and event throughput [54].

Types of performance testing

There is no single approach to the definition of types of performance tests. The following areas are distinguished, load tests, stress tests, endurance or stability tests, configuration tests, smoke tests.

Load testing Load testing is the classic form of performance testing, where the application is loaded up to the specified level but usually not further. Load testing is usually performed in order to evaluate the behavior of the application on a given expected load. This load can be, for example, the expected number of concurrent users of the application, creating a specified number of transactions per time interval.

Stress testing Stress testing is used to measure the upper limits, or the sizing of the infrastructure. Thus, a stress test continues until something breaks, e.g. no more users can log in, application becomes unavailable, etc. It is important to know the upper limits of the applications, especially, if future growth of the application traffic is expected.

Configuration testing The aim of the configuration tests is to define how different types of system's configuration can influence the productivity of the system. This test can be also combined with load, stress or stability tests [43].

Performance Testing Steps Used

According to [34], standard steps to do the test,

- Write performance requirement
- Attach a test case defining the workload and performance test
- Capture an automated performance test for the feature to be tested
- Run the test case which links to the automated performance test
- Results of the performance test are automatically recorded
- Repeat the performance test in every iteration's regression suite

These guideline concepts that mentioned above are fundamental of the testing activity of this performance predictive model work and used to show impacts of service integration grows.

4.5.5 Tools Used

JMeter tool

Is an Open Source testing software. It is 100% pure Java application for load and performance testing. Jmeter is designed to cover various categories of tests such as load testing, functional testing, performance testing, regression testing, etc., and it requires JDK 5 or higher. Jmeter framework include test plans, listeners, functions, and regular expressions functions [32].

The protocols supported by Jmeter are,

- Web − HTTP, HTTPS sites 'web 1.0' web 2.0 (ajax, flex and flex-ws-amf)
- Web Services − SOAP / XML-RPC
- Database via JDBC drivers

- Directory − LDAP
- Messaging Oriented service via JMS
- Service − POP3, IMAP, SMTP
- FTP Service

JMeter simulates a group of users sending requests to a target server, and returns statistics that show the performance (functionality of the target server) application via tables, graphs, etc.

QPME (Queuing Petri Net Modelling Environment)

In the previous sections we have seen about queue Petri nets general overview, QPME is a performance modelling environment based on the Queuing Petri Net (QPN) modelling formalism. QPN formalism has several advantages over conventional modelling formalisms such as queuing networks and stochastic Petri nets. By combining the modelling power and expressiveness of queuing networks and stochastic Petri nets, Queuing Petri Nets (QPNs) enable the integration of hardware and software aspects of system behavior into the same model. If the system is to be represented as a queueing network, the modeler applies a "routed job flow" modeling paradigm in which the real-world system is conceptualized as a set of service stations which are connected by edges through which independent entities "flow" through the network and sojourn in the queues and servers of the service stations [15].

SimQPN is a discrete-event simulation engine specialized for QPNs. SimQPN simulates QPNs using a sequential algorithm based on the event-scheduling approach for simulation modeling. Being specialized for QPNs, it simulates QPN models directly and has been designed to exploit the knowledge of the structure and behavior of QPNs to improve the efficiency of the simulation. SimQPN offers the ability to configure what data exactly to collect during the simulation and what statistics to provide at the end of the run. This can be specified for each place (ordinary or queuing) of the QPN. The user can choose one of the four modes of data collection. The higher the mode, the more information is collected, and the more statistics are provided [12]. Figure 4:12 shows SimQPN's main simulation routine.

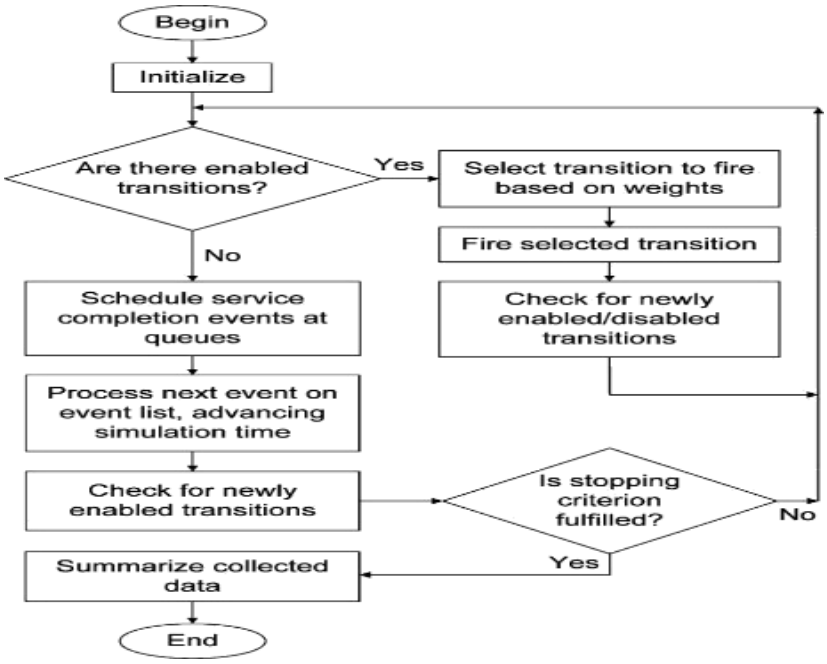


Fig. 4:12 SimQPN's main simulation routine [11]

Steady State Analysis

The Poisson and exponential distributions are used to model both arrival times and service times respectively. As mentioned, the Poisson and exponential distributions are mathematically related. If the number of service completions per unit of time, when there is a backlog of customers waiting, has the Poisson distribution, then service time has the exponential distribution. It's conventional to think of service in terms of the length of service time. The standard simple queuing model assumes the following points,

1. Arrivals have the Poisson distribution

2. Service times have the exponential distribution

3. Arrivals and service times are all independent.

(Independence means, for example, that: arrivals don't come in groups, and the server does not work faster when the line is longer). Resource utilization factor (ρ) must be less than 1 for there to be a steady state. Otherwise, customers are tending to arrive faster than they can be served. The

queue will grow more. The formulas or computations give impossible results if ρ is 1 or bigger [52].

SimQPN supports two basic methods for estimation of the steady state mean residence times of tokens inside the queues, places and depositories of the QPN. These are the well-known method of independent replications (IR) (in its variant referred to as replication/deletion approach) and the classical method of non-overlapping batch means (NOBM).

Summary

In this chapter the proposed solution approach and related concepts are explained, and the business case which is an appropriate case to show ethiotelecom business application integrations and helps to understand performance problems as their number grows. The company business critical systems are integrated as publish/subscribe scenario with IBM web sphere ESB and SIM replacement process is selected for the study, it needs a performance analysis and modeling practice to address ESB performance issue that already impacting the service delivery. This modeling development process uses SPE guidelines and QPN modeling technique.

Chapter Five: The Model

5.1 Introduction

In this thesis Queuing petri net (QPN) modeling technique is used, which is a kind of extended petri networks. The queueing Petri net (QPN) paradigm provides several benefits over conventional modeling paradigms such as queueing networks and using QPN the real system can be represented or built easily at any level. Using queueing Petri nets (QPNs), one can integrate both hardware and software aspects of system behavior into the same model. The chapter is organized as follows: Section 5.2 System Set up description and Section 5.3 The Model description.

5.2 System Set up description

Publish/subscribe service implementation scenario (it was discussed in Chapter 4) is used, as a business logic to explain the elements of ESB environment and how they work together. An API for business logic of publish/subscribe environment with Jmeter JMS (java messaging service) client was used, and messaging & routing tests was done, and instrumentation configurations made on the ESB (Jmeter connected with IBM MQ 9.1 and IBM WebSphere 7.0 application servers) also a performance server agent tool was used for CPU utilization measurement at each scenario test. In each experiment, a subset of the overall set of publishers are emulated by generating sample events. The event publication rates need not be equal to the real system publication rates λ, and they should be chosen in such a way that the load injected does not exceed the capacity of the test environment.

Laptop with the following specification: Intel® Core i7 – 8650U CPU @ 4GHz,16.00GB (15.8 GB usable) memory, 64-bit Operating system, x64-based processor, Java SE Development Kit 8 and Windows 10 Enterprise. By using ethio telecom as the case study, four different scenarios are selected (to be discussed latter in the next section).

Table 5:1 Jmeter empirical test configuration for each test scenario type

Input Items Configured in Jmeter	Description
Thread/user	Number of publishers/senders in Jmeter represented by equivalent thread numbers. Eg. 10 users with 10 threads
Arrival rate	Poisson process & exponential inter arrival rate (λ) distribution
Message size	Transaction (of 1kB)
Topic (s), queue, non-persistence, non-durable etc.	Configuration on IBM WebSphere
JMS connection factories	Object a client uses to create a connection to an IBM MQ service provider.
JMS destinations	Object a client uses to specify the target of messages it produces and the source of messages it consumes. In the PTP messaging domain, destinations are called queues. In the pub/sub messaging domain, destinations are called topics
JMS Message Listeners	Object that acts as an asynchronous event handler for messages
JMS Connections	Encapsulates a virtual connection with a JMS provider. A connection could represent an open TCP/IP socket between a client and a provider service daemon.
JMS Message Selectors	Which allows a message consumer to specify the messages it is interested in.
Message Headers & Message Bodies	Message formats or message types published

The sequence diagram of the publish/ subscribe system in Figure 5:1 shows interaction of applications , Figure 5:2, describes the empirical set up instrumented by Jmeter JMS AP which was done at the test server and also Figure 5:3 shows the JMeter user interface on which test configurations set up was done respectively.

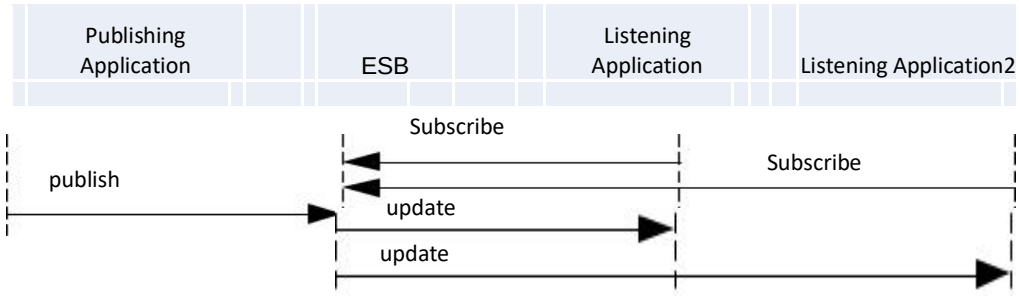


Fig 5:1 sequence diagram of the publish/ subscribe system

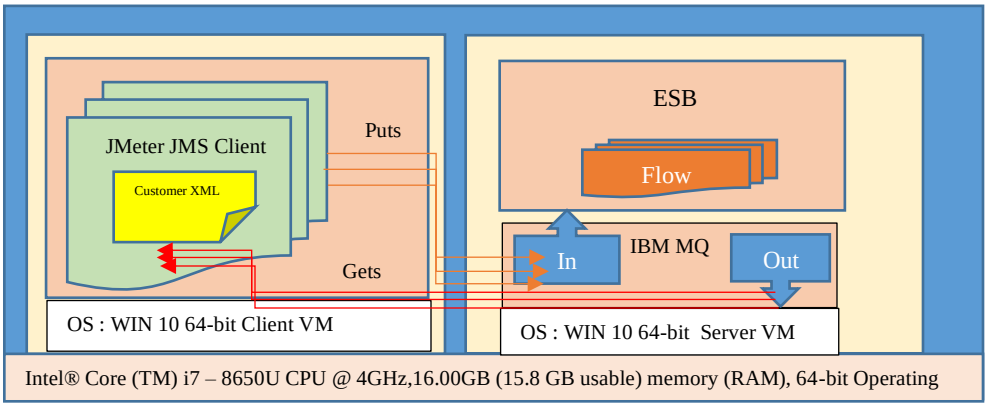


Fig 5:2 Graphical representation of the test prototype setup/topology

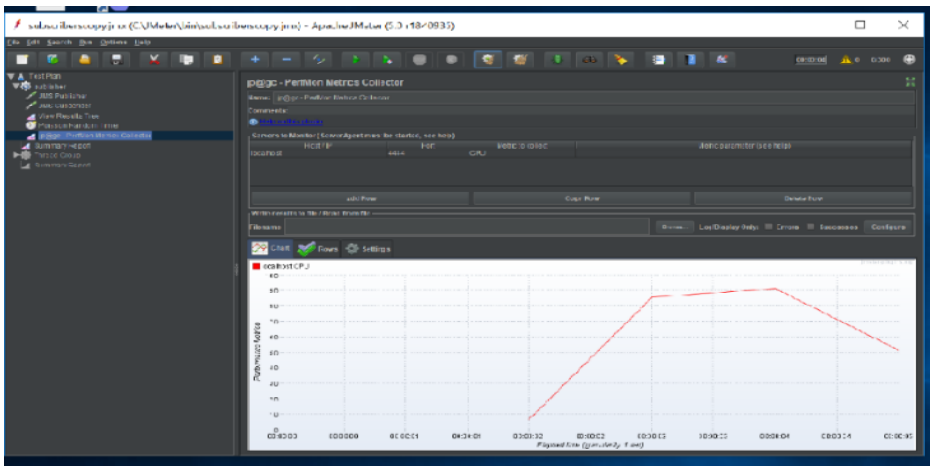


Fig 5:3 Graphical representations of Jmeter with sample output data

Key activities in the empirical test

- ✓ Construct request message using Jmeter JMS API which relates to IBM MQ 9.1 and IBM WebSphere 7.0 application server
- ✓ IBM MQ 9.1 is configured for the test scenarios, topic-based publishing and non-persistent messaging with related queue managers.
- ✓ Communication infrastructure between Jmeter and the servers is using JNDI setup, The JMS specification for Connection Factory and Destination objects to be administered objects.

- ✓ An administrator creates and maintains administered objects in a central repository, and a JMS application retrieves these objects using the Java Naming Directory Interface (JNDI). In this model, we ignore the network, assuming that network delays are negligible
- ✓ Send requests then measure and collect system response times, utilization & throughput
- ✓ Request messages arrive at the ESB with Poisson arrival process (Jmeter Poisson timer used to user request arrivals)
- ✓ CPU utilization measured using Jmeter performance monitor plugin and server agent package
- ✓ Output results, response time, CPU utilization and throughput are collected after checking their validity with standard deviation and confidence intervals.

Test Scenarios

Scenario 1 (message coping): it is investigated the impact of the number of subscribers on the server throughput and system response times.

Scenario 2 (Publish/Subscribe with Message Bus): One topic is used for all publisher/ messages, i.e. the configuration parameter should be set to one central topic.

Scenario 3 (Publish/Subscribe with Multiple Topics): For each message/publisher events, a separate topic is used, for different selectors.

Scenario 4 (Pub/Sub with Multiple Topics & CPU utilization): number of topics and their CPU utilization impact observation.

The scenarios differ mainly in terms of the behavior and function they provide. Scenario 1 is easy to implement and each new subscriber load must be set up with topic name. In contrast, Scenarios 3 and 4, which only use multiple topics, provide more flexibility and uses stress test in addition to load test. In Scenario 3, a reconfiguration of the ESB server is necessary only when introducing new message types. Scenario 2 doesn't require reconfiguration at all since a single topic (message bus) is used for communication.

5.3 The Model Description

The model captures the performance characteristics of the ESB system software and hardware as it captures all the system resources and the competing demands made on resources when executing different scenarios under different workloads. During workload characterization, four workload scenarios were identified. All of them, represent publish/subscribe transactions and are modeled using the following token types (colors): ‘purple’ for publisher events, ‘Green’ for subscriber notifications, ‘brown’ for connection pool, also ‘blue’ for publish/subscribe token flow control and here below just a sample view from the work, the model elements in the figure 5:4 are used in all scenarios and in figure 5:5 sample model configuration window. Both were used during the modeling process.

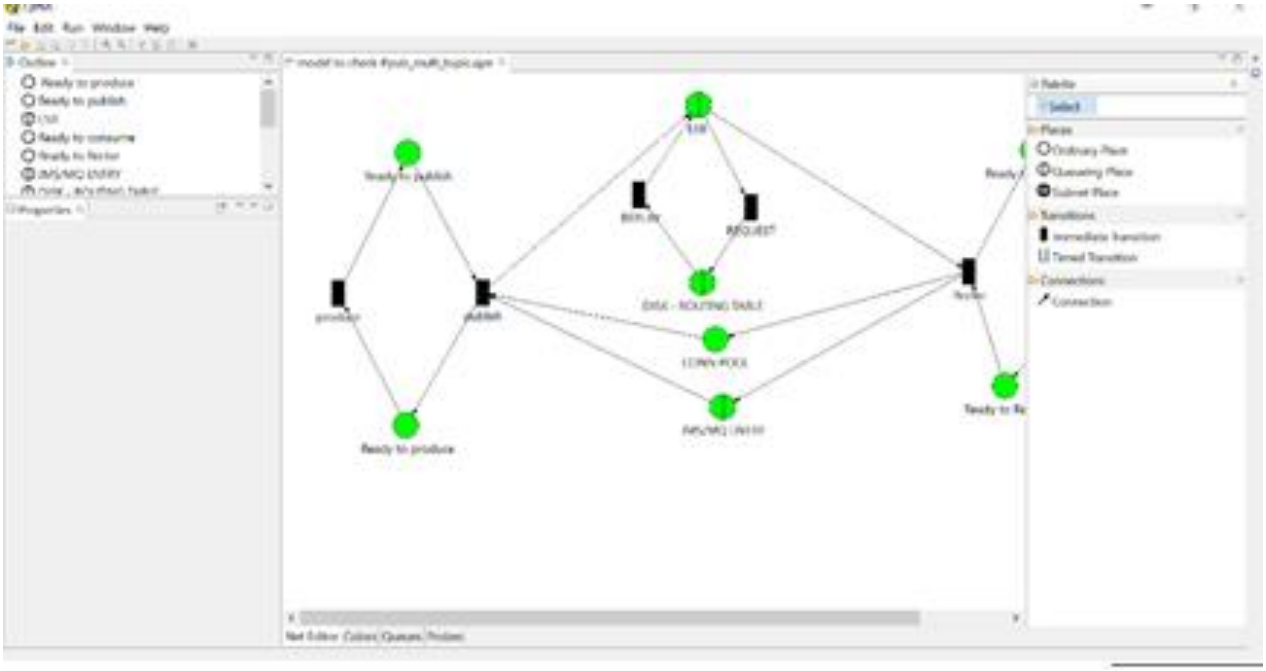


Fig 5:4 QPE Main Window

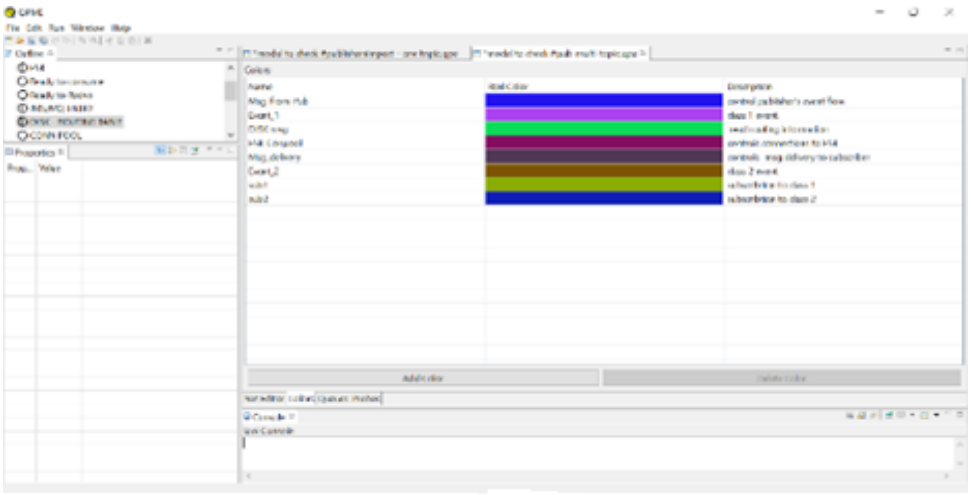


Fig 5:5 Graphical user interface of sample color naming configuration

QPE (Queueing Petri net Editor), the first major component of QPME, provides a graphical tool for building QPN models. QPE is based on the Eclipse Rich Content Platform (RCP) and the Graphical Editing Framework (GEF). QPE is written in pure Java and runs as a standalone RCP application on all operating systems officially supported by the Eclipse platform. A characterizing feature of QPE is that it allows token colors to be defined globally for the whole QPN instead of on a per place basis.

5.3.1 Queueing Petri Net Modeling Environment

Performance modeling environment is based on the Queueing Petri Net (QPN) modeling formalism (it is described in Table 5:2), in many similar studies which involve queueing network models, they recommend to begin by determining which system devices/components should be represented as service centers, and what the service demands at these centers should be. With these parameters as an input, the computational algorithms in QPM, use Little's law to calculate the effect of resource contention, and yielding performance measures such as utilizations, throughputs, residence times, and queue lengths.

Queueing network model that we have used throughout this study consists of service centers representing the CPU and the individual disk devices.

Table 5:2 Description of the model assumptions

Model Configuration items\assumptions	Description
Queue place	ESB server CPU & Disk service center representation
λ, μ and ρ	Arrival , service rate distributions independent input variables & utilization dependent output variable respectively
FCFS, IS, and PS	Service scheduling algorithms, with configuration values: Processor sharing (PS) for CPU, first come first served (FCFC) for disk and infinite server (IS) for clients.
✓ M/M/m (KENDALL NOTATION) ✓ Normal & FIFO	✓ Queue model notation: (M- Markov process, Poisson inter-arrival; the second M- exponential arrival and service distributions; m is number of available parallel servers ✓ Scheduling discipline for departure types either normal or FIFO departures after getting service at the service stations
A place	To store token which has no time delay (immediate transition)
Token	To symbol data and its flow in the model place, queue component or jobs, and initialized during simulation stage
Color	Color identifications used to uniquely identify token types

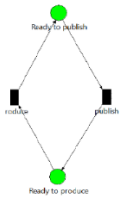
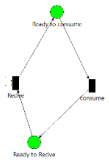
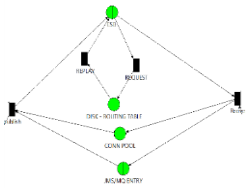
Important assumptions,

- λ "lambda" is the average customer arrival rate per unit time is constant. The expected arrival rate doesn't change over the period.
- μ: "mu" is the average customer service rate (when customers are waiting)
- μ= 1/(average service time)
- ρ"rho" is λ/μ, also called the server utilization factor
- ρ must be less than 1 for there to be a steady state
- Service times are according to a distribution
- Arrival rate must be less than service rate i.e. stable system
- Three queueing places are used, the first two representing the node CPUs and the third one representing the node's disk I/O subsystem. Events are modeled using tokens and transitions are used to move events among nodes as they are routed in the system. *(see Fig 5:5* QPE Main Window above sections)

Key activities in the model

- ✓ Build the model using queue petri net (QPN)
- ✓ Simulate the model with simQPME tool (a tool and methodology for analyzing queueing Petri net models by means of steady state simulation)
- ✓ Validate the model using simulation results against Jmeter measurements
- ✓ Evaluate the model for further performance analysis

Table 5:3 Description of the model elements

No.	Model element	Description
1	Publisher 	This component controls the producer event generation at local state
2	Subscriber 	This component controls the subscriber event consumptions at local state
3	ESB container 	This part of the model responsible for integrity of publish/subscribe system, and messaging and routing behaviors are mapped here. Also, the maximum concurrent connections pool of applications buffers for producer entry (JMS/MQ interfaces) and consumers are mapped.

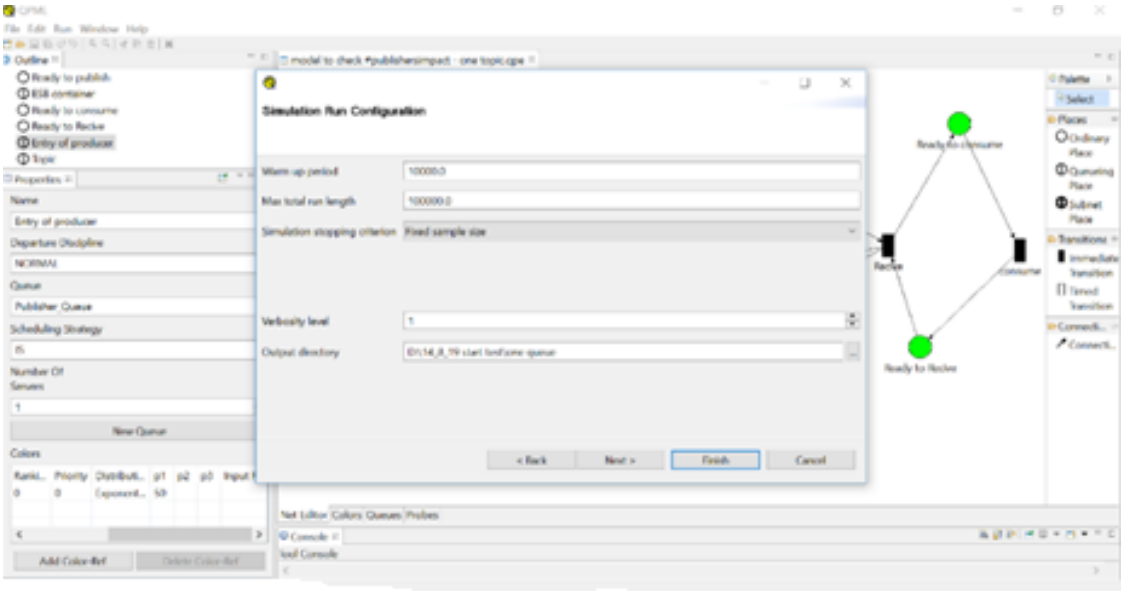


Fig 5:6 Graphical user interface for simulation run configurations of QPME

The model is collection of three M/M/1 queues and,

- Arrivals occur at rate λ according to a Poisson process and move the process from state i to i + 1.
- Service times have an exponential distribution with rate parameter μ in the M/M/1 queue, where 1/μ is the mean service time.
- A single server serves customer requests one at a time from the front of the queue, according to a first-come, first-served discipline. When the service is complete the customer leaves the queue and the number of customers in the system reduces by one.
- The buffer is of infinite size, so there is no limit on the number of customers it can contain.
- In an M/M/1-PS queue there is no waiting line and all jobs receive an equal proportion of the service capacity
- The average response time or travel time (total time a customer spends in the system) does not depend on scheduling discipline and can be computed using Little's law as $1/(\mu - \lambda)$. The average time spent waiting is $1/(\mu - \lambda) - 1/\mu = \rho/(\mu - \lambda)$. The distribution of response times experienced does depend on scheduling discipline.

Chapter Six: The Proposed Model Evaluation

After building the QPN model, it is validated by comparing its response time, throughput and utilization results with Jmeter measured values for the same metrics. Workloads considered with different user /event numbers and four scenarios with possible configurations are proposed.

Assuming Poisson event arrivals process, both inter arrival and services times are with exponential distributions, the number of concurrent users/events was varied from 1 to 100 per second, for normal workload configurations where all tasks were running on a single processor. The maximum normal load input is decided to be 100 events. The reason is the system gets very saturated with arrival of events with more than 100 concurrent events. And for stress load test also, 1000 is as maximum arrival of events. And each scenario is tested with specific configuration values which are appropriate to that specific scenario.

Quantitative resource demands for each step was given in the test configurations (see annex also). Each scenario is executed by a workload, which is driven by a closed queue workload with a given number of concurrent users or jobs. Since it was tried to emulate the end user's random pattern of request behavior, the "Poisson random generator" is used in the model and Jmeter. The service times have exponential time distribution. Also multiple request events are sent out randomly during every second. To make the environment identical this was done on both model QPME 4.5.1 and analytical Jmeter 5.0 test tools.

For the first series of tests with 1 to 100 concurrent users, the comparison between QPN model results and Jmeter measurements is given in the next section (e.g. as it can be seen in scenario 1 below at 100 users the throughput error was 3.82% but at 1000 users/events the error increases to 17.78%). From experiment and model results in section 6.2, it can be seen that the model and JMeter measured values are very close, but they begin to diverge for more than 100 users, when the system is very saturated. Figure 6:1 shows the general prediction and simulation process. And also, in section 6.2, tables 1 to 5 show the comparison of the model and measurement results. Publishers' events arrive following Poisson arrival process with inter arrival of an arbitrary time 1 second.

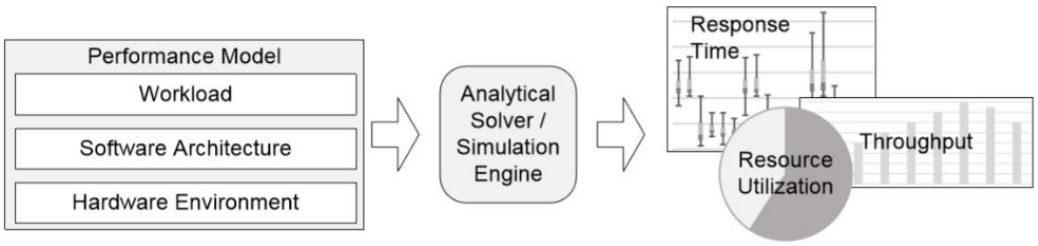


Fig 6:1 Performance modeling and prediction [20]

6.1 Using Steady State Analysis and Confidence Intervals for Data analysis

The QPN model was analyzed by means of tool SimQPN, and then validate the results with respect to correctness and accuracy. The guidelines were followed, and the model was considered with different size and complexity, the simulation outputs with different input parameters are discussed in Chapter -5.

The simulation results were compared with results obtained using other methods, i.e. analytical techniques (e.g. Little's law), approximation techniques and mainly Jmeter measurements on the system modeled. In some cases, confidence intervals for the sample mean are an interval estimate of the real mean. Interval estimates are usually desirable, since the estimate of the mean varies from sample to sample. The interval estimate provides a sign of how much uncertainty there is in the estimate of the real mean. The narrower the interval, the more accurate is our estimate. Confidence intervals are built at a confidence level selected by the user, for example 95%. For both the measurement and simulation results were checked if the confidence intervals are valid, and the minimum was 95%. To keep this result, enough number of samples during each test steps were used.

And, the batch size of the sample was 200 (as per the rule/standard of SimQPN) and as in any simulation, results vary from run to run, to measure this variation and to evaluate the confidence interval coverage, multiple replications of the above simulation run were made. As soon as enough data was available it was stopped. The report data from both Jmeter and SimQPN tools shows the standard deviation of point estimates did not exceed 2.5% of the mean value. And also in all cases the estimated coverage of confidence intervals was 95% and above. The results are discussed in section 6.2.

6.2 Experiment Results

Comparison of QPN model prediction and Jmeter Performance Measurement for response time/ throughput

Scenario 1 : Impact with the Number of Subscribers/ message replication

Table 6:1 Scenario 1 Impact of the Number of Subscribers/ message replication

	Response Time (ms)		Throughput (req/S)		
#of message replication	**Measured**	**QPN Model**	**Measured**	**QPN Model**	**Error (%)**
10	0.001	0.00135	19.4	21.9	12.89
20	0.008	0.00136	40.2	41.9	4.23
30	0.01	0.00138	58.3	61.71	5.85
40	0.013	0.0014	79.5	81.81	2.91
50	0.014	0.00143	96.2	101.69	5.71
60	0.019	0.00145	117.6	121.6	3.4
70	0.022	0.00147	131.3	141.62	7.86
80	0.023	0.00149	150.7	161.52	7.18
90	0.028	0.00145	171.9	181.57	5.63
100	0.035	0.001453	184.2	201,51	9.4

Fig 6:2 Impact of no. of subscribers Vs throughputs

Discussion:

Scenario-1, is about the impact of subscribers' size or message copying, increasing the number of subscribers increase the throughput of the server but it's impact on service time latency is very small or insignificant, to be cost effective with ESB solution there should be enough subscriber number for better performance benefit and efficiency. The throughput increase from 19.4 to 184.2 for number of subscribers 10 and 100 respectively. Generally, as it is seen Table 6:1 response time data, number of subscribers has little impact on ESB service time performance. And the model maximum throughput error is 12.89%. Which is acceptable and valid to consider the model for further performance prediction work.

Comparison of QPN model prediction and Jmeter Performance Measurement for response time/ throughput with publisher events

Scenario 2 (Pub/Sub with one topic)

Table 6:2 Scenario 2 (Publish/Subscribe with one topic)

	Response Time (ms)		Throughput (req/S)		
#of Publisher event arrivals	Measured	Model	Measured	Model	Error (%)
10	0.01	0.0014	18.5	19.8	7.03
20	1	1	35.84	38.27	6.78
30	1	1	51.41	56.05	9.03
40	1	1	65.95	71.48	8.39
50	1	1	83.68	86.48	3.35
60	1	1	99.3	101	1.71
70	1	1	108.78	114.5	5.26
80	1	1	126.3	127.2	0.71
90	2	1	126.4	139.02	9.98
100	3	2	140.77	150.12	6.64

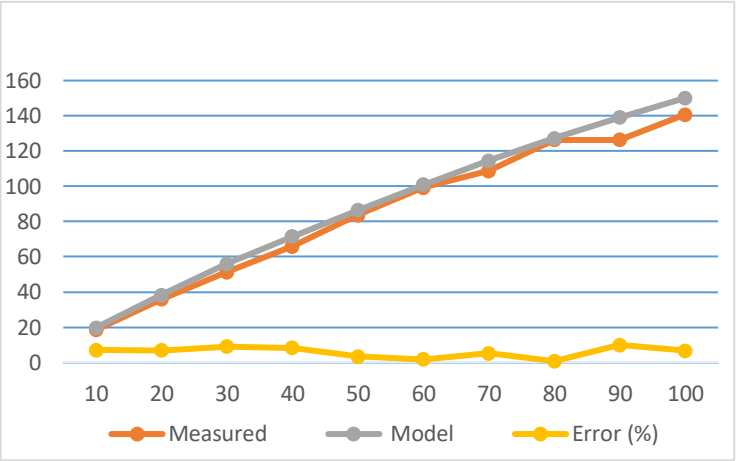


Fig 6:3 Impact of no. of event arrivals Vs throughputs

Discussion:

Implementation of publish/subscribe with one topic is the simplest configuration, following Poisson's arrival process per second time duration, producers and consumers interact with one single location, the result as it can be seen in Table 6:2, both throughput and service time will increase, as we increase the number of integration growth/ publishers events but very small service time changes were observed, due to the single topic nature, performance degradation can occur as we increase the traffic (we will see it below at scenario 4 from stress test) , for instance for 90 publisher sample events, service time and throughput were 2 ms and 126.4 respectively. Due to maximum concurrency, for 100 samples simultaneously the service time and throughput starts to increase, which was not true for small number of samples (you may see in the table above for samples 10 to 80). The model throughput prediction error was between 0.71% to 9.98.

Comparison of QPN model prediction and Jmeter Performance Measurement for response time/ throughput with multiple topic numbers

Scenario 3 (Pub/Sub with Multiple Topics)

Table 6:3 Scenario 3 (Pub/Sub with Multiple Topics)

	Response Time (ms)		Throughput (KB/S)		
#of Publishers	Measured	QPN Model	Measured	QPN Model	Error (%)
10	0.5	1	6.2	7.2	16.13
20	0.6	1	12.65	14.2	12.25
30	0.7	1	18.74	20.70	10.46
40	0.9	1	24.22	26.85	10.86
50	1	1	30.43	32.72	7.51
60	1	1	36.17	38.25	5.75
70	1	1	40.58	43.45	7.07
80	2	1.5	51.47	48.38	6
90	3	2	52.65	53.05	0.76
100	3	2	56.95	57.51	0.98

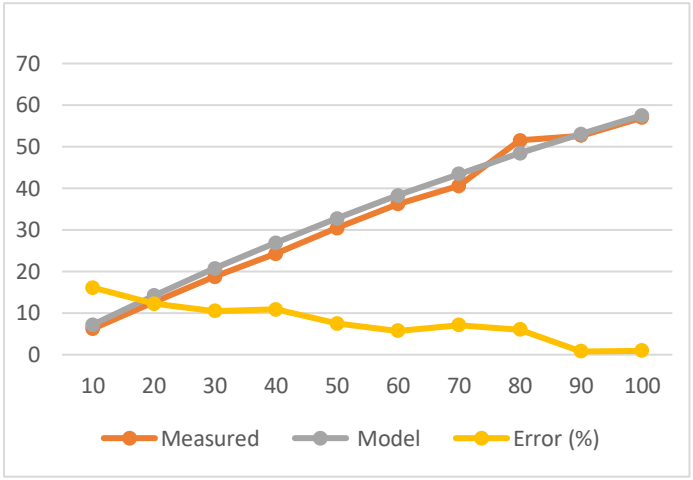


Fig 6:4 Impact of no. of event arrivals in two topics Vs throughputs

Discussion:

A multi-threaded process has one arrival entrance, one finite waiting queue, and one departure exit, and can execute multiple jobs simultaneously at a time. Many topics and environments exist in an actual production environment of enterprise applications. Such production environment requires the message-oriented middleware to guarantee the service stability and expected service quality in the coexistence of multiple topics. The testing step to measure the handling capacities of WebSphere ESB on multiple topics when message sending, and subscription coexist. As you see Table 6:3 above, the throughput increases as fast as the number of events increases while with small service time increments, also the model and Jmeter throughput measurement values close (0.98%) as the number of events increase but due to system warmup issue the error is high at the very beginning which is 16.13%. Also comparing the impact of single versus multiple topics on the performances of the ESB's CPU, it is important to see the effect of adding a new service integration which has huge message traffic (please see scenario 4 in next section).

Comparison of QPN model prediction and Jmeter Performance Measurement for response time, CPU and throughput with one & two topics peak conditions /stress test

Scenario 4 (CPU usage, response time and throughput with topics)

Table 6:5 Scenario 4 (CPU usage, response time and throughput with topics)

	#of Events	Throughput Measured	Throughput Model	CPU % Measured	CPU % Model	time measured	time model
1- Topic	50	83.68	86.48	7	11	2	2.29
	100	140.77	150.12	34.9	29.7	8	8.25
	200	139.8	230.21	55.1	46	20	27.41
	300	132.6	278.2	63.3	55.7	67	54.19
	400	125.6	308.9	68.69	61.8	160	88.26
	500	No system response	329.85	No system response	66	_	130.89
2- Topic	50	31	34.5	7.5	4.6	1	0.46
	100	61	63.8	12.7	8.4	2	1.75
	200	112.4	109.7	26.5	14.6	6	6.53
	300	149.3	143.4	37.8	19.1	10	13.97
	400	171.8	168.7	52.4	22.5	18	24.10
	500	188	188.1	63.9	25.1	40	37.32

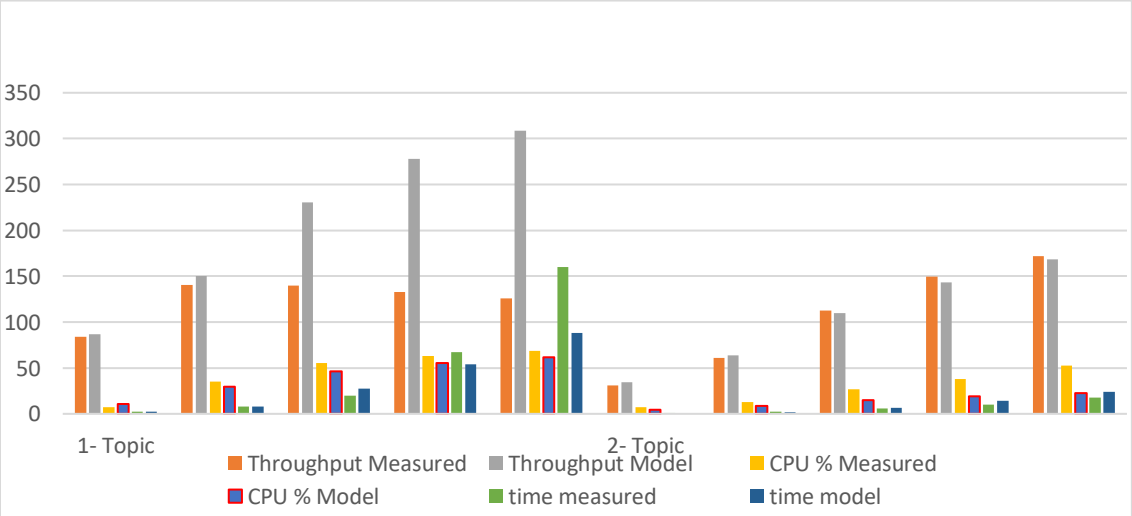


Fig 6:5 Impact of no. of event arrivals in one & two topics Vs throughputs/CPU usage and response time

As it is shown in Figure 6.5, throughput, CPU usages and time measured values in topic-1 grows fast and has higher values. In case of topic-2, the graph values for all the three metrics are small but stable for higher request arrivals or load.

It was compared the impact of the number of topics on the performances of ESB when sending and receiving ends coexist. In this case, stress test used, and as usual the arrival of publisher requests follows Poisson process with inter arrival time of one second and with exponential time distribution. Therefore, the criteria/assumption for the stress test is given below.

Keep increasing the stress on the sending end until the system throughput does not increase, the response time takes longer, and CPU utilization gets very high. As the result, a performance bottleneck emerges on the service end. Obtain the corresponding optimal throughput of the system, ensuring that there are no messages accumulated throughout the process. Topic is an important concept in message-oriented middleware. Every topic represents a message category. With multiple topics, we can categorize and isolate messages.

Note that in Table 6:6, due to parallelism CPU processing nature, multiple topic implementation will increase performance. However, having many topics also may result congestions or performance degradation, therefore the number of topics should be optimal, due to scope limitation reason two topics' implementations only were considered. By means of the QPN model, it is able to show the prediction of the performance of the system under peak operating conditions.

Table 6:6 Impact of topic configurations with stress load test (scenario 4)

Metrics	Single topic implementation	Two topics implementation	Comments
Throughput	Increasing until it reaches 139.8 (at 200 samples) and at 400 users, throughput is falling to 125.6	Slow increasing of throughput, no drop even after request number reaches 500. for small request rates both throughput and the response times also smaller than single topic. But when the request rate increases throughput get increases more, for 400 requests, the total time was 18ms, & 171.8 request/s and at 1 topic implementation the metrics were 160ms & 125.6 request/s	At peak traffic mostly model values higher than measured values. The reason is the actual environment factors on Jmeter but not on model

Response time	For a small increase of throughput, the time grows fast to 160 ms at 400 users (throughput of 125.6). it was 67 ms at 300 users (throughput of 132.6)	Compared to single topic, here smaller response time observed at 500 requests it takes only 40 milliseconds. but in case of one topic it was **nonresponsive** system for the same sample number.	At peak traffic mostly smaller than measured metrics values. The reason is the environment factors on Jmeter
Utilization	It reaches 100% CPU utilization for 500 users.	Unlike to single topic, for more than 500 requests arrivals, CPU utilization was under 63.9%.	At peak traffic mostly smaller than measured metrics values. The reason is the environment factors on Jmeter

6.3 Discussion

The model developed in the previous sections was validated by comparing its predictions against the measurements of the prototype or small scale ESB system. The model predictions were verified for four different scenarios under different transaction mixes and workload intensities.

The model was analyzed by means of simulation using SimQPN. The method of non-overlapping batch means was used for steady state analysis. And, we can predict the performance of a system under peak operating conditions.

Both the variation of point estimates from multiple runs of the simulation and the variation of measured performance metrics from multiple tests were negligible. For all metrics, the standard deviation of estimates was less than 3% of the respective mean value. The metrics considered were transaction throughput, transaction response time and server utilization.

The main objective of this work is to predict the performance impact of adding or integrating new publisher application to the ESB, it is shown particularly in scenario-2 and as we see the maximum modeling error for throughput is less than 10%. But for very high stress loads (ex. 500 and a greater number of samples) the measurement value deviates more and more from the model results; the reason is, the Jmeter measurement is more impacted or saturated by the real environment's performance factors than the model.

For service time or response times (in milliseconds) the difference is very small between the prototype and the model (see the scenarios for the detail) and, since the values are very small, percentage is not appropriate unit to compare like throughput and utilization values.

Generally, varying the transaction mix and workload intensity led to predictions of similar accuracy. Many papers say [51], a prediction error under 30% is still acceptable for capacity planning of a system. In our case for all test scenarios, the maximum error is less than 16%, Therefore, the model is valid and can be used to do preformation prediction and helpful to get an insight on status or health of the ESB during at the capacity planning times or performance assessment works on the existing ethiotelecom service operational environment. Also, the model is flexible which can be modified well for new configurations, and new components can be added with a small change to the model. Detail design and other related information of the model are included in annex part.

6.4 Summary

This chapter focuses in the evaluation, discussion point of the proposed model solution and validation by considering ethio telecom's ESB and the impact of more integrations with the study scenarios. The results show that the performance predictive model can be used as reference work for designers or service operation management of ethiotelecom IT teams.

Chapter Seven: Conclusion, Recommendation and Future Work

7.1 Conclusion

The objective of the thesis is to model the performance of an ESB service integration solution in order to manage service quality and ESB Performance challenges in early service integration stages. ESB integrates services, which means it combines more resources and requires longer execution times as integration grows and introduces further system performance deterioration. Usually service designers and implementers do not have a clear understanding of the performance overheads of service composition and will ignore them in the early system design phases. This thesis addresses this issue by measuring and modeling the following ESB overheads:

- Message performance cost for send and receive for different message amount or event numbers and workloads.
- Performance cost for different publish/subscribe configuration options.
- Performance cost for messaging engine execution and disk access for routing information.
- The number of samples for the measured data was determined by using confidence intervals. The same sample, logical assumptions and scenarios used as parameters to construct an QPN model for the case study system and to validate it against Jmeter measurements taken under different workloads and number of events.

QPN model was built using the same input parameters as empirical Jmeter measurement and showed how the QPN model can be used for further performance analysis of the system, for different deployments and configurations. CPU and I/O contention was demonstrated how some complex aspects of system behavior such as composite transactions, software contention and asynchronous request processing can be modeled. The modeling error for transaction throughput did not exceed 17% for all scenarios , not exceeds 10% for main objective of this paper (scenario -2) and was much lower for transaction response time and resource utilization.

This can be useful to designers and operation teams for assessing the performance effects of different solution alternatives in the early development and early implementation phases, before the system is entirely built and implemented. It has been showed how more detailed performance models based on QPNs can be built and used to provide more further accurate performance prediction for additional requirements.

7.2 Recommendation

While designing and implementing enterprise applications, it is very important to study and consider the nature of the application's service demand required to integrate and work together between other integrated enterprise applications. Otherwise after implementation correcting and finding other workaround for solutions is more expensive in terms of system performance, time, new investments , service quality or customer satisfaction.

The company needs to have a performance practice and management during design and service operations that considers the current and upcoming future applications. The trend so far is costing the company a lot of time, money and effort; because for most projects introduced in the company the existing infrastructure requires redesign and major modification or resource duplication.

The existing Enterprise service bus is not serving all enterprise applications, rather it serves for few applications and some other third-party applications. It would be better for the company to carefully design and building enterprise-level service bus that could mediate all the current and upcoming enterprise applications that can lead to standard SOA technology and environment.

7.3 Limitations of the work

The case study in this thesis is relatively not complex compared to real world ESB-SOA systems. we only consider factors like concurrent users , message size or number of publishers, subscriber and some key configurations, but there are other important factors which needs to be considered like network conditions, load balancing, security functions of ESB.

7.4 Future Work

This work only represents a first step for performance measurements and evaluations of ESB. There are various dimensions for modeling and analysis in the following foreseeable directions:

- Use a variety of SOA systems implementations to model them using QPN and perform a more thorough validation.
- Performance measurements and modeling can be applied to other ESB functions and elements not used in this thesis.
- Study in more depth the effect of network communication and application interfaces
- Performance modeling can be done for other company’s systems which has critical impact on the company business

REFERENCES

[1] Kebede Alemtsehay, “An Improved Technique for Enterprise Service Bus Data Transformation: The Case of ethio telecom”, 2018

[2] K. Martin, et al. “Patterns: SOA with an Enterprise Service Bus in Web Sphere Application Server V6”, May 2005

[3]. Eric Newcomer, "Understanding Web Services XML.WSDL, SOAP, and UDDI", 2005

[4]. W. Jieming and T. Xiaoli, "Research of Enterprise Application Integration Based-on ESB" , 2010.

[5]. H. Gregor and W. Bobby, "ENTERPRIS INTIGRATION PATTERNS", August 2003.

[6]. The blueprint for an enterprise service bus (oracle source)

[7]. M.Swientek, et al., “Service-Oriented Architecture: Performance Issues and Approaches”, 2008

[8] Gunter Bolch , "Queueing Networks and Markov Chains book ", 2006

[9] F. Bause, F. Kritzinger, "Stochastic Petri Nets-An Introduction to the Theory", Vieweg Verlag, 2002.

[10] Dr. Conor McArdle, "EE414 - Performance Evaluation of Queuing Systems Performance Evaluation of queuing Systems"

[11]. S. Kounev, A. Buchmann ,”Performance Evaluation” 63 (2006) 364–394

[12]. "International Journal of Computer Systems (ISSN: 2394-1065) ", Volume 03– Issue 06, June 2016 Available at http://www.ijcsonline.com/

[13] Kai Sachs, Stefan Appel, Samuel Kounev, and Alejandro Buchmann ," Benchmarking Publish/Subscribe-based Messaging Systems ", (2009)

[14] zur Erlangung des akademischen Grades , " Performance Modeling and Benchmarking of Event-Based Systems, (Dr.-Ing.) ", 2010

[15] Gunter Bolch , Stefan Greiner Hermann de Meer Kishor S. Trivedi , " Queueing Networks and Markov Chains Modeling and Performance Evaluation with Computer Science Applications Second Edition "

[16] www.ibm.com/support/knowledgecenter/SSFKSJ_7.5.0/com.ibm.mq.pro.doc

[17] www.cl.cam.ac.uk/teaching/0910/ConcDistS/

[18] Paul A. Jensen, " Closed Queuing Networks " , August 8, 2003

[19] Jun Li, " Performance Measurement and Modeling of BPEL Orchestrations " , 2013

[20] Andreas Brunner, "Continuous Performance Evaluation and Capacity Planning for Enterprise Applications ", 2012

[21] IBM documentation

[22] Worku Melesse, " A Data Synchronization Solution Model for Enterprise Applications Integrating; the Case of ethiotelecom ", 2018

[23] Abdullah AL-Malaise AL-Ghamdi, Farrukh Saleem, " Enterprise Application Integration as a Middleware: Modification in Data & Process Layer " , August 27-29, 2014

[24] {M.Swientek, U.Bleimann and P.S.Dowland}, “Service-Oriented Architecture: Performance Issues and Approaches ",University of Applied Sciences Darmstadt, Germany 2008

[25] {Yan Liu, Ian Gorton, Liming Zhu}, “Performance Prediction of Service-Oriented Applications based on an Enterprise Service Bus ", National ICT Australia, 2007

[26] {Ken Ueno, Michiaki Tatsubori}, " Early Capacity Testing of an Enterprise Service Bus " Tokyo Research Laboratory, IBM Research

[27] {Marzieh Asgarnezhad, Ramin Nasiri, and Abdollah Shahidi}, " A Systematic Method for Performance Analysis of SOA Applications "

[28] {Tomasz Górski, Kacper Pietrasik} “Performance analysis of Enterprise Service Buses”,

Institute of Naval Arms and Computer Science, Polish Naval Academy, 2017

[29] {Tomasz Masternak et al} , " ESB - Modern SOA Infrastructure" , May 2014.

[30] https://nptel.ac.in/course.html

[31] https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.pro.doc

[32] https://www.tutorialspoint.com/Jmeter

[33] Java Concurrency Tutorial

[34] IBM Software Group

[35] Ethiotelecom, SMO-SCA 02 SIM Card Replacement Process Manual, 2015.

[36] Robert R. Rowntree, M. Eng., " Ahead of the Pack – The Source for High Performance WebSphere SOA ", 2007

[37] https://studylib.net/patterns-soa-design-using-websphere-message-broker

[38] Antonina Mitrofanova , " Continuous times Markov chains. Poisson Process. Birth and Death process ", December 18, 2007.

[39] http://www.mathcs.emory.edu

[40] https://tice.agroparistech.fr

[41] https://pdfs.semanticscholar.org/BIRTH AND DEATH PROCESSES AND ORDER STATISTICS

[42] Using IBM WebSphere Message Broker as an ESB with WebSphere Process Server

[43] Elena Medvedeva, "Performance comparison of JBoss integration platform implementations", May 2014

[44] Ferrari et al., "Computer system measurement - An Overview of Queueing Network Modelling" ,1983

[45] Pelayo Medrano Garcia, "Practical relevance of the Petri nets model language for a real-time multi-processor system", December 2014

[46] John Wiley & Sons Ltd, “Queueing Modelling Fundamentals with Applications in Communication Networks”, 2008

[47] { Kai Sachs , Samuel Kounev, Stefan Appel}, “Benchmarking of Message-Oriented Middleware”, 2014

[48] Thanisa Numnonda and Rattakorn Poonsuph, “Publish/Subscribe Architecture with Web Services”, 2011

[49] Oskar Ferm, “Decision Making Support for Systems Integration Choices of Service Oriented Architecture Platforms”, January 11, 2012

[50] Thijmen de Gooijer, “Performance Modeling of ASP.Net Web Service Applications: an industrial case study”, 2011

[51] James Bean, “ SOA and Web Services Interface Design”, 2010

[52] Samuel L. Baker , “Queuing Theory I”, 2006

[53] Roberto Baldoni etal, “Dynamic Message Ordering for Topic-Based Publish/Subscribe Systems”, 2011

[54] Samuel Kounev, “A Methodology for Performance Modeling of Distributed Event-Based Systems”, 2008

[55] Brunnert/Wischer/Krcmar, “performance of Business application”, 2014

[56] Thomas Gschwind, “Software Engineering and Middleware”, September 20, 2004

[57] Samuel Kounev and Alejandro Buchmann, “On the Use of Queueing Petri Nets for Modeling and Performance Analysis of Distributed Systems”, February 1st 2008

Annex A: Model Validation Procedure

This is a guiding procedure to validate the proposed predictive performance solution Model functionality and model correctness. The validation procedure is organized and tabulated in three Sections as shown in (Table 7) below. The procedure in the first Section is to validate the model components proper functionality through the simulation metrics data information. The second Section is to validate the model functionality through comparison of the measured data with simulation data.

Table 7 Validation Procedure

Validation Procedure			
Functionality test			
Preliminary			
1	Run QPME as administrator from program files		
2	Open new design page and test the program with simple example , save		
3	Open the file and test for full functionality of the program		
Section One: Functionality Validation for model components			
	Test Procedure and Inputs	**Expected result**	**Test Result (Pass/Fa)**
1	From the model interface page click entry of events/producer	Arrival/s data input box would be displayed	
2	In the input text box enter the arrival/second value Ex.10 events/s	The arrival rate initialized and would be displayed	
3	Configure service rates at service centers/queues at ESB's CPU & disk or cache e.g.1400 events/s , 450 or 750 events/s respectively How to measure the service rate μ? Note : the mean service time & rates computed using Little's law and utilization law Input to the model: arrival rate- λ , service rate- μ ➢ There are many approaches, depending what aspect of your system you want to model	Configured service centers/queues or (CPU & Disk)	

	E.g., Service rate > arrival rate. $\rho < 1$. use configuration that gives maximum throughput ,sample case, **Input** : λ=10 req/s, μ=1450 req/s Therefore, mean service time=1/service rate =1/1450 = 0.00069 seconds		
4	Initialize publish/subscribe control components Ready to produce PLACE =1 & Ready to consume PLACE =1, entry buffer queue=1	Adjusted local states QPN PLACES and buffer full settings on displayed	
5	On the run option configure the simulation: – Warmup time =10000 seconds – Simulation type = Batch Mean – Batch size=200 – Level of simulation=4 – Console=true – Output directory	Ready setup to start simulation	
6	Click 'Run this simulation configurations' to start	Simulation report	
Section Two: Measurement and Model Simulation Comparison Performance Test			
	Test Procedure	**Expected result**	**Test result (Pass/Fa)**
1	Open Jmeter JMS API with publish/subscribe configured test file, Configured items: – IBM MQ Destination name – IBM MQ connection factory name – Context factory – IP & Port number – Message/payload – Publisher name – Subscriber name – Queue Manager – Poisson arrival rate – Licenser ,threads and report Metrix – Number of test iteration – Utilization measuring tool plugged	Jmeter publish/subscribe configured test data	
2	The same configuration on IBM ESB side	Configured and Ready ESB to accept Jmeter publish/subscribe interaction test	

3	Send Jmeter publish event message to destination name/topic	Publishing	
4	Send subscriber request	Sent requests & receive response	
5	Throughput , response time & utilization data collection	Metrix information report	
6	From the metrics I get the service times using utilization law/Little's Law are computed, later will be used as model input for queue devices service rates.	Service times & service rates of queue devices (CPU or disks)	
7	Report generation for arrival rates 1,10,….100 for normal load testing on both Jmeter and model simulator (QPME)	The same response time, throughput and utilization report results but with small difference results (small & acceptable errors)	
Section Three: Validation for model components with stress and load performance tests			
1	Generating arrival rates 50….1000 for stress loading test both on Jmeter & model simulator (QPME) , ➢ Increasing the load step by step ➢ Observation on utilization changes until it reaches 90%-95%	Stressed system and highly loaded CPU	
2	Collected data checked with confidence interval and standard deviations for each test scenarios	Validated data from both tools	
3	Compare the result with measurement data	The same result but small errors and logical differences were observed	

Annex B: Model Node and Input Design

Table 8 Node model design description

	Property						
No.	Node Name	Departure discipline	Queue /Place Name	Scheduling strategy	Number of servers	Color reference of node	Initial no. of tokens
1	JMS/MQ entry	Normal	Publisher queue	IS	1	PURPLE	1
2	Ready to Publish	Normal	Place / no queue	-	-	BLUE	0
3	Ready to produce	Normal	Place / no queue	-	-	BLUE	1
4	ESB component	Normal	ESB queue	PS	1	PURPLE & BROWN	0
5	Disk -route record	FIFO	Disk -route record	FCFS	1	GREEN	0
6	Connection pool	Normal	Place/no queue	-	-	Red	100
7	Ready to consume	Normal	Place / no queue	-	-	Dark brown	0
8	Ready to Receive	Normal	Place / no queue	-	-	Dark brown	1

Note :

- Colors used to uniquely identifiers of group or individual model element
- Token to control number of events

Annex C. Model Transitions Design

Table 9 Model Transitions and Input Design

	Property				
No.	Name	Priority	Firing weight	Modes	color
1	Produce	0	1	1	orange
2	Publish	0	1	1	yellow
3	Receive	0	1	1	gray
4	consume	0	1	1	lead

Note :

- Colors : used as unique identifier of group or individual model element
- Mode : to control #of transitions
- Priority: order of firing i.e. 0 means equal priority, 1 high priority
- Firing weight : number of tokens to be fired

Transitions are events or actions which cause the change of state. A transition is enabled if all input nodes have the number of tokens equal to or greater than those needed for firing the transition. System behaviors that lead to state change are generally modeled as transitions, and the action of transition fire will cause tokens to disappear or emerge in the places connected with the transition.

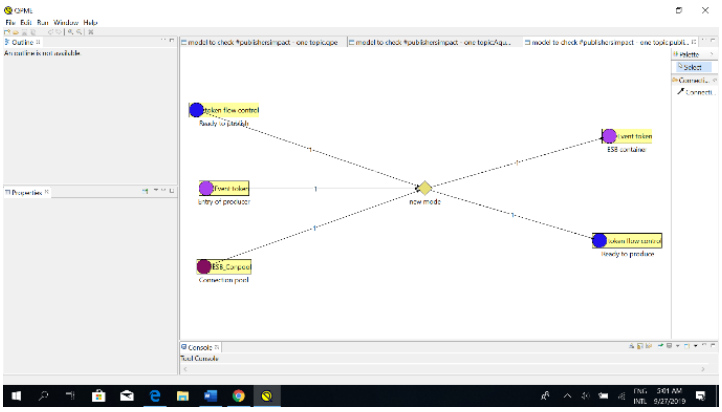


Figure 1 Sample model Transitions Design

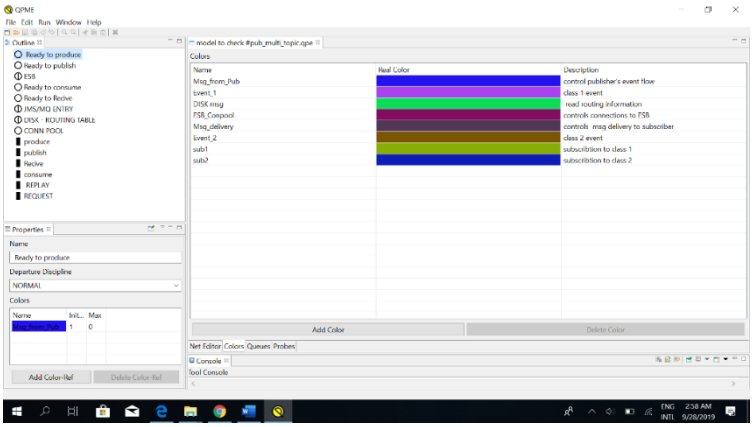


Figure 1 Sample model Node property and input Design

Annex D: Interview Questions

This interview question is a semi-structured interview in which for smooth communication between the interviewee and the interviewer, the interviewer may not strictly follow the questions sequentially. The purpose of this interview is to collect technical and operational related information about integrated enterprise applications implementation and other information in related to ESB performance in ethiotelecom. If not confidential the information gathered from this interview is intended to be used as an input for the research being conducted for the Master of Science in telecom engineering in Addis Ababa University Institute of Technology. The research title is 'Performance Predictive Model of ESB as service Integration Grows: The Case of ethio telecom'. Any confidential information that the interviewee specified as confidential will not be published or used in the research documentation. The interviewees are selected based on their better technical knowledge in the application domain in ethiotelecom.

Performance prediction is the process of estimating or identifying performance status of components among integrated applications. And, performance problem is a situation that prohibits integrated applications from having acceptable service quality in the service operation and delivery.

Name of the Interviewee: _____

Current Position: _____

Roll: _____

Experience in the Position: _____

Gender _____

Technical Interview Questions

1. What are the Enterprise applications integrated to ESB?
2. Your duties in these applications?
3. What kind of performance problems with these applications? Please be exhaustive and
4. provide scenario.
5. What kind of performance problem frequently reported?
6. Which incident type considered as major or critical incident that impacting the business?
7. What is response time and through put status before and after design? At Peak hour time?
8. Which application more impacted from integrations?
9. Any performance follows up practice in service operation departments? Monitoring?
10. What are the business rules and security policies that should be considered in EAI data communication?
11. how many applications integrated to ESB?
12. What functionalities are performed by the ESB?
13. Application concurrency configuration on ESB
14. Is any additional information you want to tell me about performance challenges on ESB?

Declaration

I, the undersigned, declare that this MSc thesis is my original work, has not been presented for fulfillment of a degree in this or any other university, and all sources and materials used for the thesis have been acknowledged.

Solomon Tiku Desta
Name

_____
Signature

Place: Addis Ababa, Ethiopia
Date of submission: Jan 2020

This thesis has been submitted for examination with my approval as a university advisor.

Dr. Mesfin Kifle
Advisor's Name

_____
Signature