Software Requirement Specification For Scalable and Fault Tolerant Architecture for Event Processing Systems

Supervised By: **Mr. Manusha Wijekoon(Ext.)**

Mr. K. Sarveswaran (Int.)

Group Members:	**H.E. Martin**	**060295J**
	D.S. Metihakwala	**060305T**
	D.M.R.R. Sampath	**060428X**
	H.C. Randika	**060398D**

Table of Contents

1. Introduction

1.1 Purpose

Introducing a scalable and fault-tolerant architecture, which can be used as a reference architecture for building distributed complex event processing systems is the main purpose of this project. The project is divided into three phases.
The main purpose of this document is to describe the functional, non-functional and all the other general software requirements for each phase of the project.

1.2 Document conventions

This document is a SRS for the Scalable and Fault Tolerant architecture for complex event processing systems. This SRS is structured in the following format.
Section 2.0 Overall Description
Section 3.0 External Interface Requirements
Section 4.0 System Features
Section 5.0 Other Nonfunctional Requirements
Section 6.0 Other Requirements
Appendix A describes all the abbreviations use in the document.

Throughout this document, the following conventions have been used:-
Font:Times New Roman
Size 16 for Main Headings
Size 14 for Sub Headings
Size 12 for the rest of the document.

The key words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119. These key words mean the same thing whether capitalized or not.

1.3 Intended audience

Intended audience for the SRS can be classified under two groups. They are
Present Stake holders who are actively involved in the project:

- Developers
- Project Evaluators

Future Stake holders:

- Developers involved in design complex event processing systems
- Customers

This document is intended for all the above mentioned audience.
Top down approach is recommended for each audience for reading.

1.4 SRS team members

H.E. Martin	060295J
D.S. Metihakwala	060305T
D.M.R.R. Sampath	060428X
H.C. Randika	060398D

1.5 References

1. http://www.faqs.org/rfcs/rfc2119.html
2. http://en.wikipedia.org/wiki/Complex_Event_Processing
3. http://code.google.com/edu/parallel/dsd-tutorial.html
4. http://en.wikipedia.org/wiki/Simulation
5. http://en.wikipedia.org/wiki/Modeling_language

2. Overall Description

Project focuses on introducing a scalable and fault-tolerant architecture for development of distributed complex event processing system. Since the project is divided into three phases each phase is described below.

Phase one is aimed at the creating a simulation model for the proposed architecture during the research. This is for verification purposes of the proposed system.

The second phase of the project has the capability to act as a fully featured small scale implementation of the proposed architecture with small scale complex event processing system as well.

During the phase three we will convert this proposed architecture in to a framework where third party developers can implement their business logic and use our distributed architecture to create a scalable and fault tolerance complex event processing system.

Following sections will go deep in to all these sections introduced above.

2.1 PHASE ONE

2.1.1. Product Perspective

The first phase of the project aims to build a simulation model of the introduced architecture for verification purposes. The basic features of this includes initial setup of the system to be simulated, random and guided feeding of events, guided addition of triggers and scheduled manipulation of the system attributes during the simulation. This is to demonstrate the validity of the system for scalability, fault tolerance, efficiency and consistency by examining both its safety properties and liveliness properties.

2.1.2. Product functions

This system models all the components and characteristics of the introduced architecture for verifications purposes. This includes components having specific capabilities assigned for simulation of processing and transmission delays, a mechanism to manipulate the system during simulation, a mechanism to feed random and guided inputs with specific weights and a mechanism to collect statistics.

2.1.3. User classes and characteristics

This will be used only by the development team for verifying and validating the introduced architecture.

2.1.4. Operating environment

The model is to be developed using a verification modeling language and a model checker on a Windows or a Linux system with preferably a 2 GHz+ dual core processor and 1 GB+ of RAM. In case of a performance issue or an unavoidable issue of existing verification modeling languages and model checkers, we may switch to an ordinary programming language which will have the same hardware and operating system requirements as mentioned above.

2.1.5. Design/implementation constraints

In a real distributed system, there can be several types of errors, namely halting failures, fail-stop, omission failures, network failures, network partition failures, timing failures and byzantine failures. Many of them are very complex, time consuming and may not be advantageous to specifically represent in a simulation system. Due to their complexity and the timing constraints on this project, we will have to represent some of the mentioned failures as one of the two general types of failures in our simulation system, which will be host(process) and connection failures. These two failures will be represented in the system by removing relevant components from the system.

We will be mainly focussing on collecting required statistics of the simulation system to prove that our architecture satisfies the requirements. Hence, we will not be focusing on developing an interactive system for this phase and instead we will mainly focus on developing a system which can represent the architecture with a reasonable accuracy and to collect its statistics during simulation for analytical purposes.

We will not be focussing on security of the system during simulation phase, that is, attacks to the system and other security threats will not be represented. Those features will be considered in phase 3.

We will not specifically concern about the user friendliness and rather will concern about the accuracy.

2.1.6. Assumptions and dependencies

We assume that the delays caused by simulation platform and hardware are negligible and does not have a big impact on the results of the simulation. This is based on the fact that we are not going to do actual processing on any input fed to the simulated system and hence it won't consume significant amount of resources of the system. This assumption will hold only if the system is executed on a reasonably capable computer as mentioned in section 2.1.4.

2.2 PHASE TWO

2.2.1 Product perspective

The second phase of the project has the capability to act as a fully featured small scale implementation of the proposed architecture. The implementation has the capability to expand by connecting more machines and it has the ability to handle a variable number of events and triggers throughout the time of operation. Machines can be physically disconnected to simulate network failure. The user interfaces would provide the feedback on the performance and fault tolerant behavior of the implementation of the architecture.

2.2.2 Product functions

The system has several main software components, the dispatcher, the result accumulator and a software component for a partition are some of them. The existence or absence of these components is determined by the developed architecture. In this implementation each of the separate components are deployed in separate machines. There are several instances of some components in the system while other components only have a single deployed instance. The scalability of the system is in the capability to change the number of partition to accommodate a variable number of events and triggers. The implementation enables the user to change the number of partitions, change the number of events and triggers and simulate network brake downs. The user interfaces in each of the deployed components gives feedback to the user about the performance of the architecture. The actual event processing would be simulated by a simple test event processing component deployed in the partitions.

2.2.3 User classes and characteristics

The second phase of the implementation is only to be used by the project development team. The small scale implementation is only to be used as a proof of concept of the developed architecture.

The main characteristic of the user group would be that all of the group has a firm and deep understanding of the architecture and the implementation.

2.2.4 Operating environment

The implementation would be in java, so the operation environment has to have the Java Virtual machine installed. The machines for deployment would have the minimum configuration as stated below.

1. Intel 2.00 GHz+ processor
2. Minimum of 2GB RAM
3. Storage of 40GB hard drive
4. Network interfaces of Ethernet IEEE 802.3
5. A Microsoft Windows Operating system

2.3 PHASE THREE

2.3.1 Product perspective

The third phase of the project is aimed to implement a framework for Complex event processing systems designers. This framework will consist of the functionalities to handle data flow for Events and Queries, handling incoming events and queries, connectivity between distributed processes and query persistence. The third party developer has to implement the business logic for the complex event processing system. Depending on the research, phase 1 and phase 2 there will be few parameters available for third party developer which are related to distributed system architecture configuration. So this framework simplifies the complex event processing system designers tasks by handling the connectivity related tasks of the distributed system. The scalability and fault tolerance will be handled by the framework itself according to the parameters inserted by the developer. The number of hosts in the network is an example for such parameters since scalability and fault tolerance depend on the number of peers. Query persistence will be handled by storing the incoming queries in a reliable data source such as hard disk drive.

2.3.2 Product functions

Although the functionalities supported at this phase are highly dependent on the architecture and algorithms that we are going to use in the first two phases, we can outline the basic functionalities that we are going to integrate to the framework. Since this framework is for complex event processing designers, most of these functionalities are based on distributed systems and complex event processing systems.

- Scalability
- Fault tolerance
- Query Consistency
- Configurable
- Security
- Extendability
- Maintainability
- Low over head

2.3.3 User classes and characteristics

This framework is specifically designed for Complex event processing system development. Hence the intended users of the framework are programmers who have a sound knowledge about complex event processing systems. This framework will handle the other tedious tasks for these developers so that they just need to implement the complex event processing system. The connectivity between peers of the distributed system and data flow will be handled by the framework as explained earlier.

The main characteristic of the intended user group is they have a sound knowledge about complex event processing. They might have knowledge about distributed systems as well. But it is not necessary to have thorough knowledge on Distributed Systems. The Framework will handle the distributed architecture based on the developer assigned parameters for the framework's configuration parameters.

2.3.4 Operating environment

This framework will be implemented in Java, C# or C++ with suitable middle-ware support. If we decide to implement this in Java, then the operating environment will require Java Virtual Machine. Since Remote Method Invocation (RMI) is coupled with Java Run-time Environment itself, there will be no extra software required. If the system is implemented in C# or C++, then it will require .NET framework for its operations. Depending on the middle-ware we select for the system, it will require some other software such as Cobra supportive software.

When considering the hardware requirements, applications developed on this framework will host on normal desktop computers with following hardware.

1. Core 2 Duo 2.00 GHz processor
2. Minimum of 2GB RAM
3. Storage of 40GB hard drive
4. Network interfaces (E.G.: Ethernet IEEE 802.3, Wi-Fi IEEE 802.11)
5. Microsoft Windows or Windows compatible Operating system

2.3.5 Design/implementation constraints

The third phase of the project, the framework implementation has large number of constraints. At the moment almost everything are uncertain since we have not finalized our algorithms, architecture or system components. All these things will be decided after a careful consideration of the results of the research that we are planning to carry out before involve in any design work. At the research we have to come up with an algorithm for distributed system which supports scalability and fault tolerance. Based on that algorithm, we will decide the components. Depending on the qualities such as user friendliness, robustness, extendability, etc, we will decide the language we are going to implement the system. The underlying distributed system architecture and the configuration for the third party developer are also depending on the architecture we use.

2.3.6 Assumptions and dependencies

Third phase is highly dependent on the results of the research that we are planning to carry out. Since this framework is developed to simplify the tasks of a complex event processing system developers and designers, we have to assume that the system user will use the system appropriately for their developments. We assume that the business logic developers have some idea about the resource allocation as well as they to specify the configuration for the distributed system that we are running.

3. External Interface Requirements

3.1 PHASE ONE

3.1.1 User interfaces

The system will have console based and file based interfaces for the user inputs and outputs.

3.1.2 Hardware interfaces

The system will not have specific hardware interfaces.

3.1.3 Software interfaces

The system will run on a verification model checker and thus will be written according to a suitable verification modeling language specification. This will provide necessary software interfaces to the verification model checker for providing the features mentioned in section 4.1 for simulation.

3.1.4 Communication interfaces

This will not have any communication interfaces.

3.2 PHASE TWO

3.2.1 User Interfaces

The second phase implementation of the project has command line user interfaces. There are distinct user interfaces depending on the component running on the deployed machine.The information shown in each console window of each machine depends on the component. The components show information related to the architecture such as the number of partitions connected, the number of total events received up to now, the number of events routed to each partition, the total number of events that have been completed up to now, the list of triggers assigned to each partition etc.. The exact details shown in each interface heavily depends on the actual configuration of the architecture that would be developed.

3.2.2 Hardware interfaces

There will be many hardware interfaces in the second stage implementation. All the components have interfaces to communicate with network and internal file storage devices. The functions and requirements of the hardware interfaces differ depending on the component.

3.2.3 Software interfaces

The second phase implementation is to be done most likely with java, hence each system requires Java Virtual Machine installed in them. Java RMI would be used to communicate with other components in the system.

3.3 PHASE THREE

3.3.1 User interfaces

This is the third phase of the project. In this phase our aim is to implement a distributed architecture framework for complex event processing system developer so that they can use our framework to create a distributed system to host their complex event processing system core engines. Hence this phase of the project does not contain any Graphical user interfaces as it is a scalable and fault tolerance distributed application development framework. The output of this phase will be in form of Java Archive (jar) or Dynamic Link Library (Dll). Since other developer are able to implement their application on top of this framework, they should be able to know the status of the underlying core components. So there will be Command line interfaces such as Debug consoles with complete logging services with hierarchical logging support such as INFO,DEBUG, etc. with professional logging services provider log4J or log4net.

3.3.2 Hardware interfaces

As this is a Distributed System application development framework, there will be lot of hardware interfaces interacting with the system such as network interfaces, system units, hard drives and etc. Network interfaces will be used to connect two or more processing unit in order to synchronization purposes, data transmissions and information transferring purposes, etc. A Hard drives will be used to retrieve data required for the basic operations.

3.3.3 Software interfaces

Depending on the requirements there will be different types of software interfaces will interact with the system. Currently we are planing to implement the core components in C or C++ and the higher layers in Java. Hence system requires Java Virtual Machine installed in the system. And to communicate between the C/C++ native program and higher layer, system requires Java Native Interface.

Since each processing unit running the same application, they need to communicate between these interconnected computers/processing units. If we use Socket connections,as it is very primary the system functionalities will slowdown. Hence we are planning to use a middle-ware for inter process communications. If we use Java for implementation, we will use Remote method invocation and if We use C++ we can use DCOM or Cobra. If we use C# for the system implementation, then we can use Remote Procedure Calls functionality in Dot Net Framework for that.

3.3.4 Communication protocols and interfaces

In a distributed system, communication is an essential part. In this project we are using inter process communication via middle-ware such as Java RMI, C# RPC , etc. In Java RMI, the protocol used to communicate between processes is called as RMI Wire Protocol which is consists of two well known protocols. They are Java Object Serialization and Hyper Text Transfer Protocol commonly known as HTTP, RMI Multiplexing protocol and RMI Transport protocols. Each of them have a specific tasks to be executed for a successful data transfer.

4. System Features

4.1 PHASE ONE

4.1.1. System configuration

4.1.1.1. Description

This feature is to:
1. allow the user to setup a system for simulation with each component and links having specific limitations such as processing powers, storage limits and delays.
2. change the system during simulation. The change can be either random, scheduled or interactive depending on the applicability. This will be used to reflect scaling of the system and failures of components such as host and network link failures in the system.

4.1.1.2. Action/Result

Defining the components with capabilities and logical topology – The system shall start simulation based on a user defined model. This includes distributed components(virtual hosts) to have limited processing power and storage capabilities and links to have delays.
Addition and removal of components – The system shall be able to continue its operation taking newly added components for simulation and should reflect the consequences due to newly added and existing components.
These actions shall be performed in a way which assures that the system remains in confirmation with the proposed architecture and preserves essential characteristics such as scalability and fault-tolerance.

4.1.1.3. Functional requirements

REQ-1: Receiving inputs specifying components and their limitations, connections between them and their limitations(delays).
REQ-2: Receiving inputs specifying components and links to add with their limitations.
REQ-3: Receiving inputs specifying components and links to remove.
REQ-4: Output specifying the status of system configuration operations.

4.1.2 Input feeding

4.1.2.1. Description

This is for feeding guided or random inputs to the system. This includes both triggers and events, each having assigned weights which reflect the required amount of processing to be done on them and other possible characteristics.

4.1.2.2. Action/Result

Feeding triggers – system shall accept triggers with weights falling within a predefined range and continue its operation reflecting the consequences of both newly added and existing ones. The maximum number of triggers will be limited depending on the capabilities of the simulating system.

Feeding events – system shall accept events with weights falling within a predefined range and continue its operation reflecting the consequences of newly added ones as well as the existing events. The system shall accept a limited number of events depending on the capabilities of the simulated system.

4.1.2.3. Functional requirements

REQ-1: An input specifying events and their weights added during a specific time interval.
REQ-2: An input specifying triggers and their weights added at a moment.
REQ-3: Output specifying the status of input feeding operations.

4.1.3. System statistics collection

4.1.3.1. Description

This feature will be to store the state of the system at scheduled or arbitrary time interval for analytical purposes.

4.1.3.2. Action/Result

Configuring state observation time span: This shall be done initially before the execution of the simulation system starts. The system shall collect state of the simulated system according to the initially specified time span and store them in a file according to a predefined format.

4.1.3.3. Functional requirements

REQ-1: An input specifying the time span to collect system statistics.
REQ-2: An input specifying a location and a file name to save statistics.
REQ-3: Output of statistics to a file.

4.2 PHASE TWO

4.2.1 Event Generator

4.2.1.1 Description and Priority

Since the second phase of the project is a scaled down implementation of the architecture, in order to test the system a set of events has to be given to be processed. The usage of these events are essential in order to test the system for proper behaviour.

4.2.1.2 Action/result

The events for this test would be a set of text files each containing a different set of strings. Each text file would be read and passed to the system to be processed through the set of triggers. When a certain event is passed to the system the system gives a feedback indicating the successful processing of the event. If a trigger is matched in an event the system gives out an alert.

4.2.1.3 Functional requirements

REQ-1: A component of the system which reads text files and sends them to the processing system.
REQ-2: A user input via console to determine the rate on which the events are sent to the system.
REQ-3: Display the total number of events sent to the system.
REQ-4: Resend an event if a feedback response does not come from the system after a fixed time period.

4.2.2 Event processor

4.2.2.1 Description and Priority

Since the project does not focus on the aspects of actual complex event processing, the second phase implementation has a simple event processor intended for testing purposes. The event processor communicates directly with the partition component of the system.

4.2.2.2 Action/result

The event processor does a simple text matching on the received text file with the set of assigned triggers for that specific partition. It does a separate text matching for each of the triggers. If a match is found it it gives a feedback to the system.

4.2.2.3 Functional requirements

REQ-1: A text matching component to search the text files for the triggers.
REQ-2: A console user interface to show the total events processed and their status.
REQ-3: A method to send alerts to the system of any hits.

4.2.3 Architecture Implementation

4.2.3.1 Description and Priority

The implementation of the architecture is the main focus of the second phase. Since the second phase acts as an demonstration of the architecture, it shows all the features and the characteristics of the architecture.

4.2.3.2 Action/result

The architecture implementation handles with the assignment of triggers to the partitions and routing the events through the partitions such that every event is exposed to all of the triggers. The Architecture implementation depends strictly on the architecture itself.

4.2.3.3 Functional requirements

REQ-1: Interface to receive events from the event generator.

REQ-2: User interface to receive the list of triggers before and at the time of operation.

REQ-3: Interface to pass the events and triggers to the event processor.

REQ-4: Interface to give feedback about complex event processing state to the event generator.

REQ-5: The capability to add/remove partitions at the time of operation.

REQ-6: User interface to show details of the performance of the architecture (depends on the architecture developed)

4.3 PHASE THREE

4.3.1 Handling Data flow for Events and Triggers

4.3.1.1 Description and Priority

This framework is working as a platform for complex event processing business logic developers. So the handling the data flow of the system is done by the components with in the framework itself. The data flow refers to the incoming events and triggers from the external sources. Handling of these events and triggers accordingly is a main task of the framework so that the business logic developer do not need to worry about those event and query distribution tasks. Framework handles all the load balancing for triggers and events.

4.3.1.2 Action/ Result

After implementing the business logic and inserting the configuration parameters of the framework, developers can start feeding the triggers and events. Then these triggers will be added to the system's trigger base in a manner such that system can support scalability and fault tolerance. Incoming events also will be distributed in a balanced manner with load balancing ability of the framework.

4.3.1.3 Functional requirements

REQ-1: Developer has to enter valid values to framework configuration parameters
REQ-2: Implement the business logic for complex event processing system
REQ-3: Integrate the framework and the business logic correctly for message passing
REQ-4: Physically Connect the peers correctly

4.3.2 Connectivity between processes in separate peers

4.3.2.1 Description and Priority

The scalability and fault tolerance qualities of the architecture are achieved by distributing the complex event processing task over few peers. This distributed system require to communicate with each other peers to complete their tasks. These connections and message passing between peers will be handled by the framework. These separate peers will run same process. So via the connections between peers these processes will communicate with each other using specific middle-ware such as Remote Method Invocation and protocols such as Java Object Serialization and HTTP.

4.3.2.2 Action/ Result

After correctly setting up the system with configuration in a single host, it will start communicating with other peers according the role assigned to this host. In this communication, process running in this host will contact the processes in other hosts using middle-ware. These communications will be depending on the architecture we are using.

4.3.2.3 Functional requirements

REQ-1: Configure the framework correctly for the distributed system
REQ-2: Availability of physical connections

4.3.3 Query Consistency

4.3.3.1 Description and Priority

In a complex event processing the queries are stored with in the system and incoming events are matched against those stored queries. So complex event processing systems are the upside down version of the relational database systems where data stored within the system and incoming queries are matched with stored data. So in this framework we have to support this typical requirement to store the queries in a reliable place where the system can easily retrieve them back if needed. This functionality is achieved by storing the queries received to a single process in the hard disk drive of the same host which is considerably reliable source. The mechanism used to do this will depend on the architecture we are going to use for the framework.

4.3.3.2 Action/ Result

After setting up the system correctly start feeding the triggers to the system. Then these triggers will be stored in hard disk drives decided by the architecture.

4.3.3.3 Functional requirements

REQ-1: Configure the framework correctly
REQ-2: Connect the network connection correctly

5. Other Nonfunctional Requirements

5.1 PHASE ONE

5.1.1. Performance requirements

The system models the introduced architecture as it is and thus it shall be able to reflect the characteristics of the architecture with a reasonable accuracy. This requires it to have high performance during execution without unwanted delays.

5.1.2. Security requirements

The system does not have specific security requirements as it will be available to and used only by the development team.

5.1.3. Software quality attributes

The main aim is to prove the validity of the introduced architecture. The required attributes will be to provide mechanisms to configure the model which conforms with the architecture, to feed variable inputs, to observe states and to collect statistics for the analysis.

5.1.4. User documentation

This does not require a user documentation as the system will be used by the development team for short term verification purposes.

5.2 PHASE TWO

5.2.1 Performance requirements

Individual component performance is not a main focus on the second stage of the project. The main objective is practically proving the success of the developed frame work. Performance would only be analyzed based on the expandability and fault tolerant nature of the implementation.

5.2.2 Security requirements

The security of the system will not be focused in the implementation.

5.2.3 Software quality attributes

The main objective of the second implementation phase is proving the concept of the architecture hence the main aim is to implement all the features of the architecture. User friendliness is not considered.

5.3 PHASE THREE

5.3.1 Performance requirements

Objective of the framework is to provide predictable performance to the complex event processing system deployed on the proposed distributed architecture. This implies that the system should have ability to provide desired responsiveness in a timely manner.This responsiveness can be through put that system processes or in some other form.

5.3.2 Security requirements

Security requirements include in the framework will support to build the distributed system which have authenticate the accesses to data and services.Also it will provide flexibility to end system developers to add their own security to the system. Communication between processes will be able to use encrypted as well as non encrypted communication channels.

5.3.3 Software quality attributes

The main quality attributes would be providing usability and availability of the framework to use by end system developers without any difficulty. Not only usability and availability but also extendability, maintainability, low over head will be availability in the framework.

5.3.4 User documentation

We will provide a manual which describes all the functionality of the framework to the end system developers.

Appendix A:Glossary

Abbriviation	**Description**
SRS	Software Requirement Specification
CEP	Complex Event Processing
RFC	Request for Comments
HTTP	Hyper Text Transfer Protocol
RMI	Remote Method Invocation
RAM	Random Access memory
IEEE	Institute of Electrical and Electronics Engineers
Wi-Fi	Wireless Fidelity
RPC	Remote Procedure Call
COM	Component Object Model
DCOM	Distributed Component Object Model
JDK	Java Development Kit
JRE	Java Run time Environment
JAR	Java Archive
Dll	Dynamic Link Library
GB	Gigabyte
GHz	GigaHertz
WWW	World Wide Web