SIMVA-SoS Requirement Specification

(working draft)

1. Introduction

1.1. Document Conventions

This SRS follows the IEEE formatting requirements. Basic font type used for SRS is **Arial** and the font size is 11 throughout the document for text. **Headings and subheadings are represented by predefined styles in GoogleDoc.** Document text is single spaced and maintains 1″ margins. Italics are used for comments, and special issues are noted by bold faces. Key aspects of the requirement are listed by bullet points in discussion.

1.2. Background

(General SoS) A System-of-Systems (SoS) is a collection of systems, which consists of independent constituent systems to achieve higher-level goals. As an SoS changes constantly due to external and internal factors, dynamic reconfiguration and evolutionary development must be performed effectively. To manage an SoS that has these characteristics, SoS managers and engineers need to model and verify an SoS that accommodates changes.

(Modeling) For the simulation and verification, an executable (i.e., simulatable) model should be generated. In SIMVA-SoS, the executable model is generated in the form of the source code, which contains the information of a target SoS and its environment. The information is first specified in several input models, and they have specific SoS-level entities, CS-level entities, and environmental entities. A user of SIMVA-SoS will import the models and it will be converted into the executable JAVA source code. Simulation scenario needs to be imported to run simulation and specified properties can be imported for the statistical verification.

(Simulation and Verification) A key aspect of SoS is that how can we achieve whole SoS-level goals by orchestrating the interconnection between otherwise independent CSs. Thus, naturally, it is necessary to study approaches to verify whether the SoS-level goals are achieved or not. To do this, we consider a simulation-based statistical verification, or Statistical Model Checking (SMC), for SoS. While SMC only provides

probabilistic guarantees about the correctness of the answer, it overcomes the critical state explosion problem.

(Slicing) Since the size and complexity of SoSs are too great to apply existing verification techniques, an efficient verification method for dynamically changing SoSs is required. If we can detect the change-related goals of an SoS and the corresponding change-affected parts in an SoS model, we can slice the SoS model in our area of concern and verify the SoS model efficiently. The slicing module could save the cost of verifying the whole of an SoS model by an efficient statistical model checking for SoSs.

1.3. Project Scope

*This section describes the **scope** of the SIMVA-SoS. (total one paragraph, 1~3 sentences for each module: included or not)*

Modeling is in the scope of the main functionality because there will be a seperate modeling tool based on the ADOXX.

Simulation and simulation-based statistical verification is in the scope of the main functionality of SIMVA-SoS.

Slicing is in the scope of SIMVA-SoS as an optional functionality for efficient verification.

1.4. Intended Audience and Document Overview

This SRS document is intended for all individuals participating in or related with the *SIMVA-SoS* project,. Readers who want to know a brief overview, motivations, and definitions of the *SIMVA-SoS* project should focus on Introduction part, as well as Overall description part of the document which shows the product perspective and functionality.

This document includes an overall description of *SIMVA-SoS*, which provides the purpose and functionalities, analysis of user characteristics and operating environment of the system. Furthermore, it describes the design, implementation constraints, some factors we assumed in whole process of the project. Also, specific requirements for *SIMVA-SoS* such as external interface, functional and behaviour requirements are described in this document. Finally this document covers non-functional requirement of

SIMVA-SoS which includes performance requirements, safety and security requirements, and software quality attributes.

For the intended users of *SIMVA-SoS*, a recommended order of reading this document is to begin with the definitions, Acronyms and Abbreviations for who is not familiar with the terminology through overall description (Section 2) and specific requirement (Section 3). For the SoS engineers, a suggested order of reading this document is to start with the overview section and to proceed through the rest of the document.

1.5. References

[1] IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998

2. Overall Description

2.1. Product Features

SIMVA-SoS is a software for simulating and verifying the SoS. SoS models are converted to executable SoS models and those become the inputs of simulation engine. Another input for simulation engine is SoS scenario data which is based on a target SoS. Also from SoS goal, verification property can be derived. Verification properties can be used for statistical verification.

For the simulation and verification of an SoS, a user needs to generate executable models, which are simulatable in SIMVA-SoS. Thus, SIMVA-SoS shall receive several input files such as SoS models, verification properties, simulation scenarios, and policy models. These input models are generated using external tools and they are formatted as XML, JSON, or TXT files. In SIMVA-SoS, a user can import the model files and SIMVA-SoS interprets and converts the input models into executable (simulatable) models. For better extensibility, SIMVA-SoS will also support on-demand generation or configuration of some models (e.g., simulation scenarios, policies).

With the given (or generated) executable models, SIMVA-SoS simulate the executable models. SIMVA-SoS conduct the simulation based on the discrete event simulation and multi-agent based simulation. Time formalism is discrete event formalism, so that the simulation is conducted by simulation logical time called a Tick which represent a unit of

time in running simulation. Multi-agent simulation represents the SoS comprising multiple constituent systems which has their own goal, functions, etc. Depending on the executable input models, simulation may have uncertainty and non-determinism.

With the simulation feature, another important feature is verification of SoS. The verification feature of SIMVA-SoS is based on the statistical model checking to consider the complexity and size of the SoS model. SIMVA-SoS verifies the input verification properties to the simulation result that is generated by the simulation module. The verification results contain pass or fail properties.

To perform the efficient verification using SIMVA-SoS, model slicing module extracts the core parts of model which is used in verification. Based on the change-related properties, an SoS model or an executable SoS program will be sliced. SIMVA-SoS verify the subject sliced model faster and efficiently. The verification result of the original model and the sliced model should be same.

2.2. Operating Environment

2.2.1. Hardware

2.2.1.1. desktop or laptop

2.2.1.2. without external GPGPU for high performance computing

2.2.2. OS

2.2.2.1. Windows 10 (or above 8)

2.2.3. installed JVM

2.3. Development Environment

2.3.1. IntelliJ

2.3.2. Gradle 3.1, JUnit 5.0

2.3.3. GitHub (https://github.com/SESoS)

2.3.4. Java Coding Standard

2.4. Constraints and Assumptions (for developers)

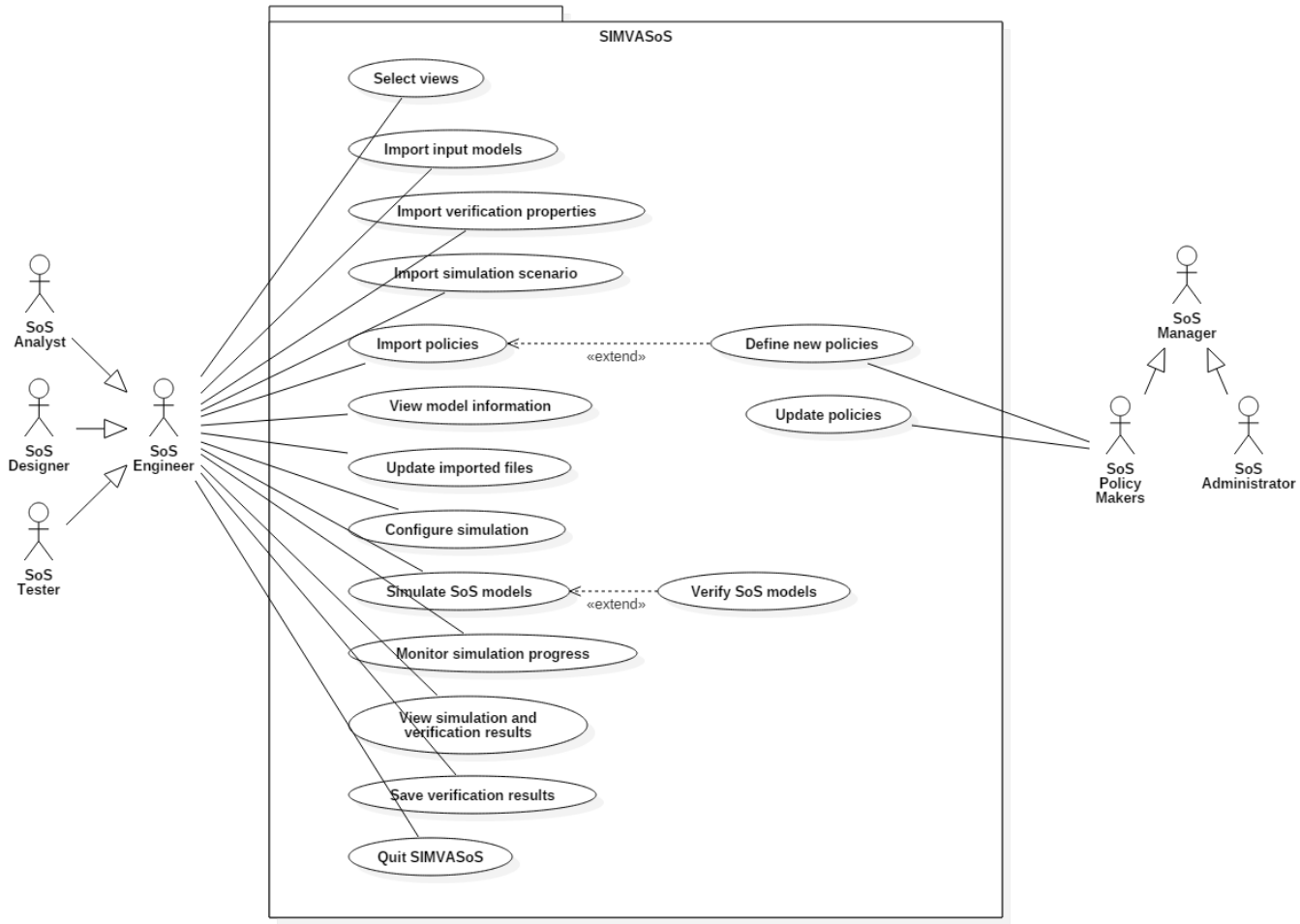
We have a limited resource. No money. No parallel system. No more developers. SIMVA-SoS will be used for the SoS experts who are also familiar with programming and verification.

For now (as in the requirement document), we assume SIMVA-SoS is domain-general.

3. System Features (use case)

3.1. Use case diagram

3.1.1. Overall use case diagram



- Select views
- Import input models
- Import verification properties

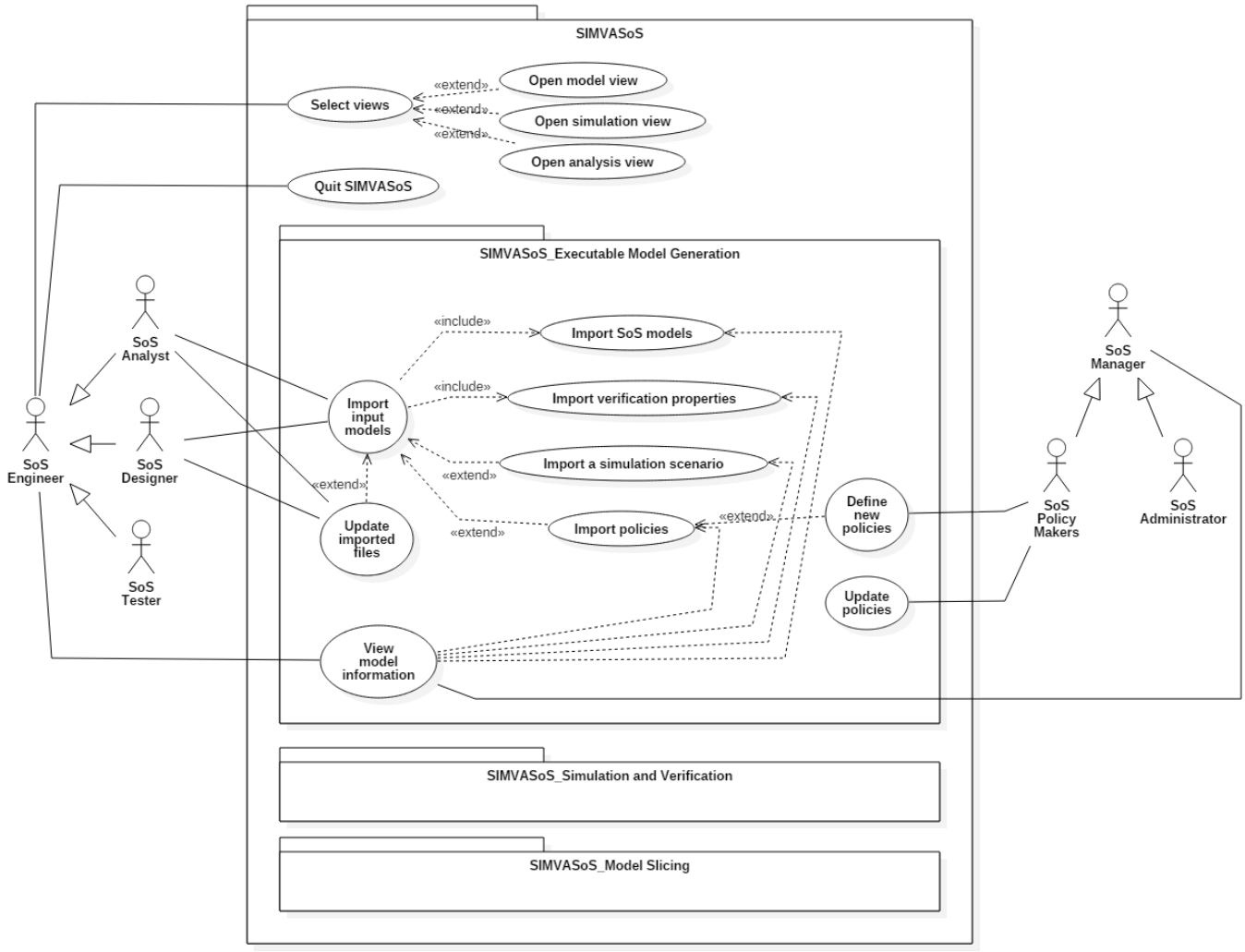
- Import simulation scenario
- Import policies
- Define new policies
- Update policies
- View model information
- Update imported files
- Configure simulation
- Simulate SoS models
- Verify SoS models
- Monitor simulation progress
- View simulation and verification results
- Quit SIMVASoS

3.1.2. Actor

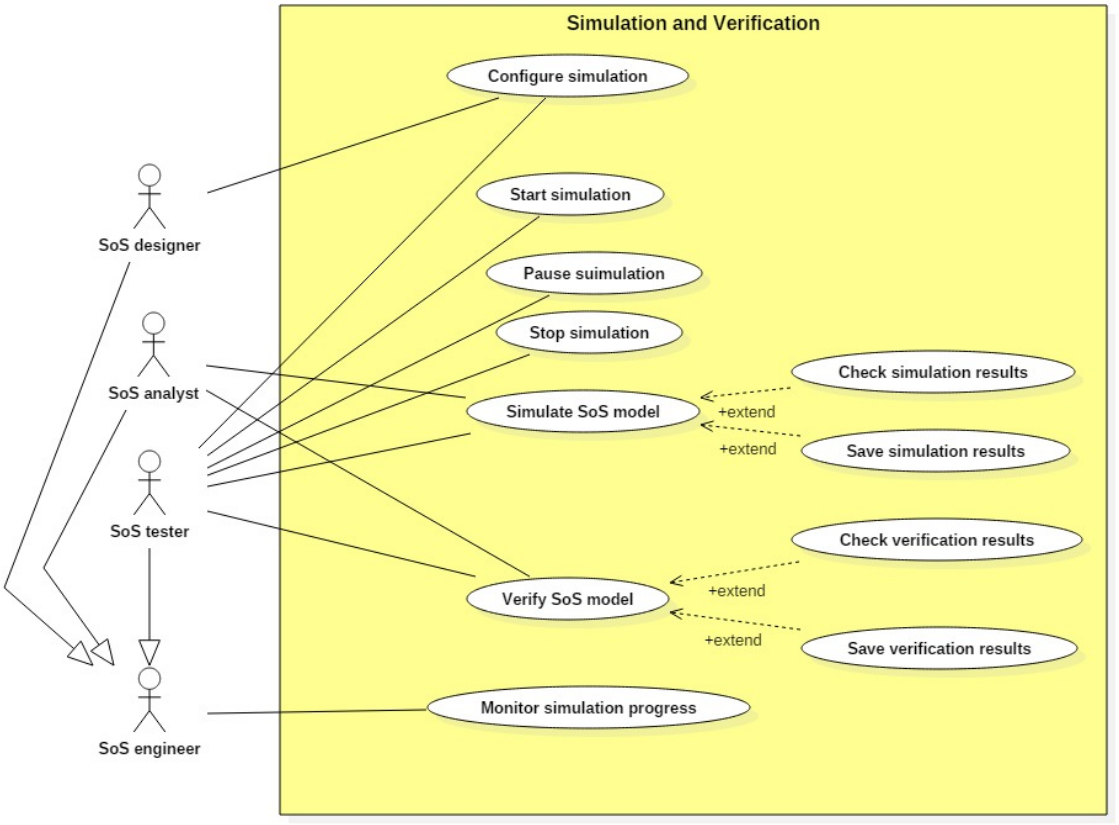
Actor	**Description (role)**
SoS engineer	
- SoS analyst	
- SoS designer	
- SoS tester	
SoS manager	
- SoS policy maker	
- SoS administrator	

3.1.3. Use case diagrams for SIMVA-SoS modules

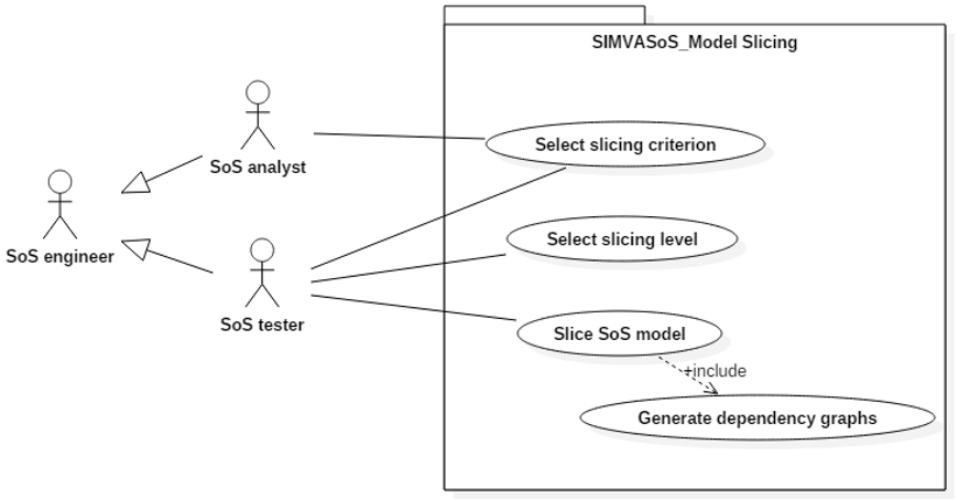
3.1.3.1. Executable model generation



3.1.3.2. Simulation & Verification



3.1.3.3. Model slicing



3.2. Use case description

https://docs.google.com/spreadsheets/d/1PZHPZO7qzlz05j5BhtJWdpsDSpSz6eaFXpHzX2vB57o/edit#gid=0

3.3. Functional Requirements

Functional requirements are derived from the use cases described in Section 3.2. The functional requirements should be listed in the form of table similar to the table for use cases.

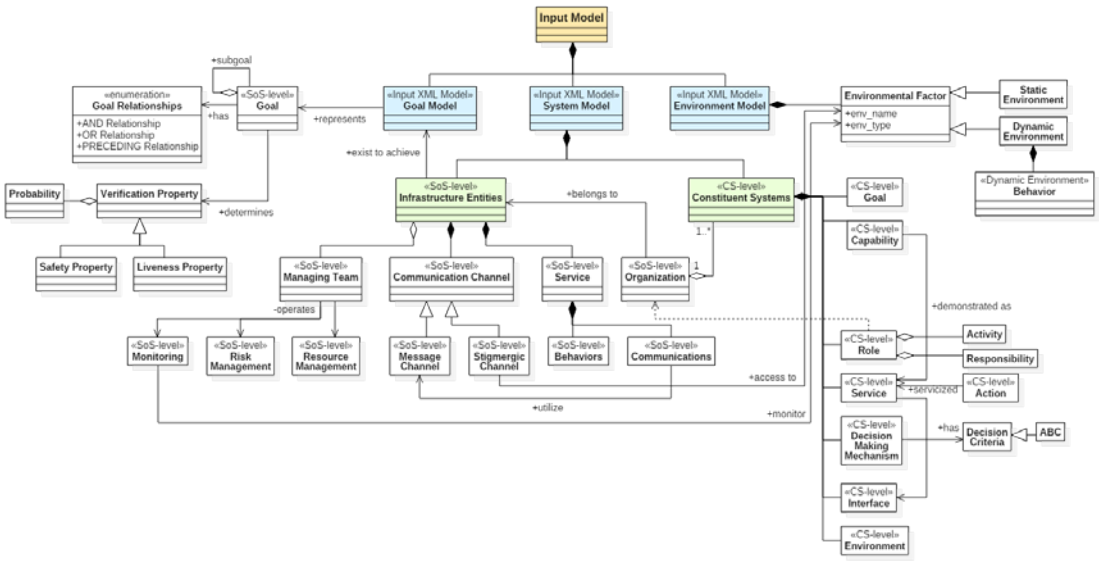
https://docs.google.com/spreadsheets/d/1PZHPZO7qzlz05j5BhtJWdpsDSpSz6eaFXpHzX2vB57o/edit#gid=0

3.4. Inputs and Outputs

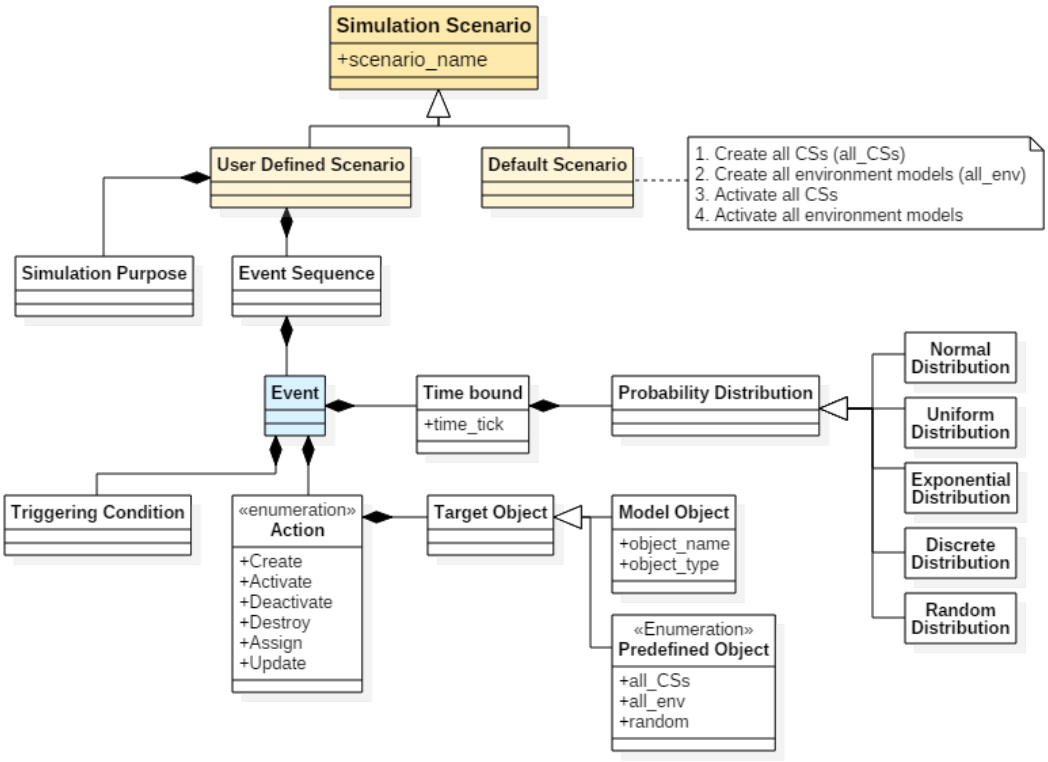
3.4.1. Overall input model

* Input = Goal model + System model + Environment Model.

The following figure shows the contents of the input model.



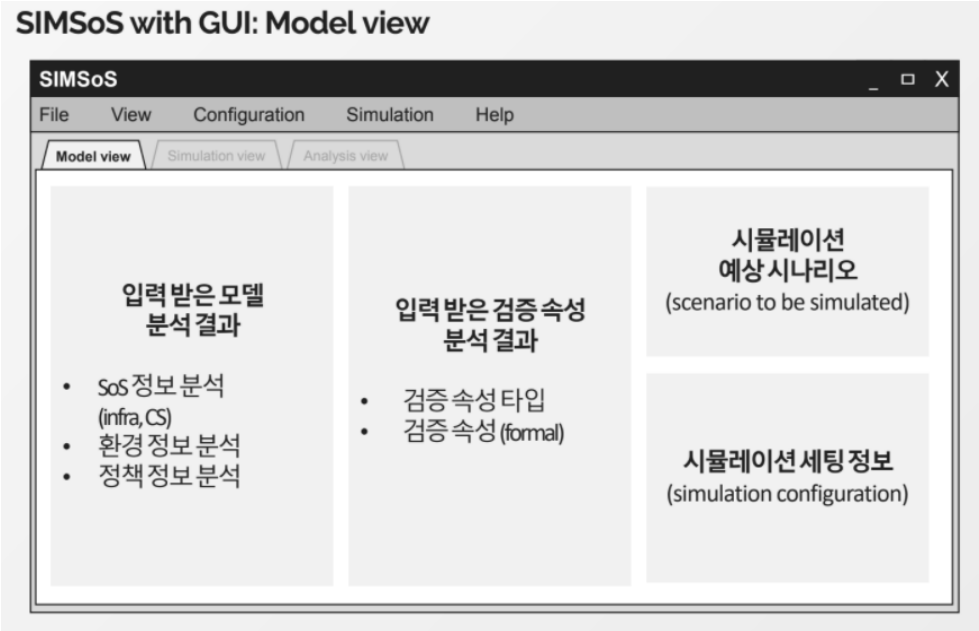
3.4.2. Simulation scenario



3.4.3. Policy model (not yet defined)

4. External Interface Requirements

4.1. Graphical User Interfaces



5. Nonfunctional Requirements

5.1. Product requirement

5.1.1. Usability

SIMVA-SoS should be easy to learn and use from interface perspective and functionalities included. Interactive messages and computation results should be easy to understand and interpret.

5.1.2. Performance

Average response time should not exceed 8 seconds without displaying information about the computation progress.

5.1.3. Efficiency

SIMVA-SoS should operate on average memory, processor and disk of personal computers.

5.1.4. Reusability

SIMVA-SoS should be able use input data from related systems, and its output should be readable by other related system for further analysis, for example with modelling tools

5.1.5. Portability

SIMVA-SoS should be executable on different platforms including operating systems, and processors vendors

5.1.6. Robustness

In the presence of fault inputs, SIMVA-SoS should perform basic operations with less complain ,and provide possible causes of incomplete operations.

5.1.7. Scalability

SIMVA-SoS should provide services when input size becomes above threshold values, with little performance compromise

5.2. Process oriented requirement

5.2.1. Maintainability

SIMVA-SoS Should be modularized and easily maintainable (depends on programming language we use)

5.2.2. Readability

Documentation should be consistent and readable, follow common standard of coding

5.2.3. Testability

Easy of testing and error reporting measure-mut

5.2.4. Understandability

Design, architecture and code should be as clear as possible to convey message in a simple way

5.3. Operation oriented requirement

5.3.1. Access security

Should support different user level of access to prohibit unauthorized access and modification

5.3.2. Availability/dependability

Should be dependable during normal operation

5.3.3. Survivability

During failure, system state should be intact, data consistency should be kept

Apart from functional requirements (what SIMVA-SoS must do) as discussed in previous section, there are also nonfunctional requirements it should satisfy.

Nonfunctional requirements can be defined as

- How well the system must do its functional requirements [Thayer 2000]
- Constraints on the services or functions offered by the system. They don't usually just apply to individual system features or services [sommerville 2007]
- A property or quality that the product must have, such as an appearance, or a speed, or accuracy property[Robertson 2006]
- Nonbehavioural requirements that describe the required overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understand ability, and modifiability. [Davis 1993]

However, we choose the following definition of nonfunctional requirement because it is most comprehensive and embodies other definitions.

A nonfunctional requirement is a specification of how well a software system must function[Roxanne E.Miller 2009].

Identifying nonfunctional requirement is not an easy task. There are well developed techniques and approaches to help in identifying functional requirements, but not applicable for nonfunctional requirements. There are numerous contributing factor or reasons that the nonfunctional requirements are difficult to identify. Three major factors are discussed as follows

(1) Subjective – they can be viewed, interpreted and evaluated differently from user to user.

(2) Relative – the interpretation of relevance and importance might vary depending on the specific system under consideration, as well as the products and services produced by a business.

(3) Integrated – the goal of nonfunctional requirement can conflict with one another.

There are various efforts to make nonfunctional requirement elicitation easier, systematic, and more understandable. For example, Boehm's software quality characteristics tree, Sommerville's nonfunctional requirement type, Withall's

software requirement patterns, and Miller's user focused approach for nonfunctional elicitation.

We considered the Sommrvillie's and Miller's approach for our purpose.

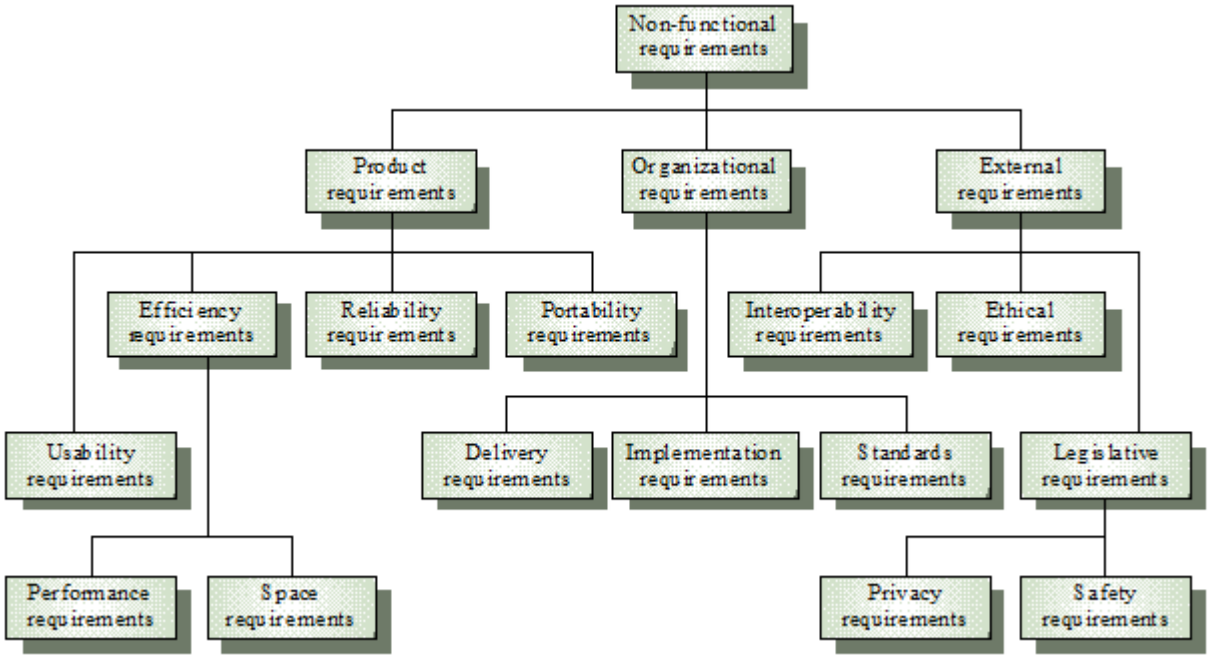


Figure 1. Types of nonfunctional requirements (Gerald Kotonya and Ian Sommerville)

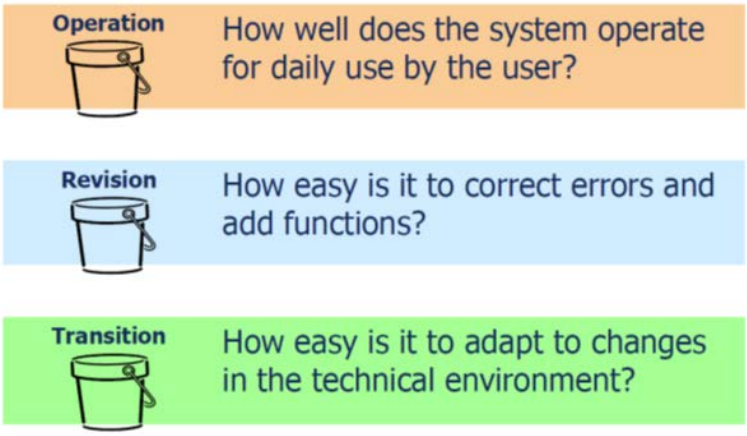


Figure 2. Where are the users need ? (Roxanne E. Miller)

	User Concern	Nonfunctional
Operation	How well is it safeguarded against unauthorized access?	Access Security
	How dependable is it during normal operating times?	Availability
	How fast, how many, and how well does it respond?	Efficiency
	How accurate and authentic is the data?	Integrity
	How immune is the system to failure?	Reliability
	How resilient is the system from failure?	Survivability
	How easy is it to learn and operate the system?	Usability
Revision	How easy is it to modify to work in different environments?	Flexibility
	How easy is it to upkeep and repair?	Maintainability
	How easy is it to expand or upgrade its capabilities?	Scalability
	How easy is it to show it performs its functions?	Verifiability
Transition	How easy is it to interface with another system?	Interoperability
	How easy is it to transport?	Portability
	How easy is it to convert for use in another system?	Reusability

Figure 3. A user focused classification (Roxanne E. Miller)

Based on Sommrvillie's and Miller's approach of eliciting nonfunctional requirement we identified SIMVA-SoS nonfunctional requirement as follows. We included the classification type to clear out terminologies confusion as used differently from different perspective.

i.

Classification	Nonfunctional requirements	SIMVA-SoS specification
Product requirement	**Usability**	Should be easy to learn, use and interpret
	Performance	Average response time should not exceed 3 second,
	Efficiency	Should operate on average memory, processor and disk of personal computers
	Reusability	Should be able use input data from related systems, and its output should be readable by other related system (with the modelling tools)

	Portability	Should be executable on different platforms including operating systems, and processors vendors
	Robustness	In the presence of fault inputs, it should provide basic operation with less complain
	Scalability	Should provide services when input size becomes above threshold values, with little performance compromise
	Modifiability	It should support addition of new functionalities
Process oriented requirement	Maintainability	Should be modularized and easily maintainable
	Readability	Documentation should be consistent and readable, follow common standard of coding
	Testability	Easy of testing and error reporting measure-mut
	Understandability	Design, architecture and code should be as clear as possible to convey message in a simple way
Operation (user focused classification)	Access security	Should support different user level of access to prohibit unauthorized access and modification
	Availability/dependability	Should be dependable during normal operation
	Integrity	Data should always be accurate

	Reliability	Except hardware failure, system should be immune from failure, and consistent in its operation
	Survivability	During failure, system state should be intact, data consistency should be kept
Transition(user focused classification)	Interoperability	Should easily interface with other related systems, for example modelling tools, and repositories

Other Requirements