Software Requirements Specification

for

PhysioMIST

Mark Caral, Sara Cummins, BarbaraJoy Jones, Joshua Lee

September 23, 2009
EECS 393

Contents

1 Introduction

This document describes software functional and nonfunctional requirements for the PhysioMIST model integration interface. This document will be used by the project team to implement and verify the system.

1.1 Project Scope and Product Features

The PhysioMIST model integration interface will allow users to integrate physiological models and produce simulation results. The *Vision and Scope Document for PhysioMIST* provides a detailed description of the project's features and release priorities.

1.2 References

1. *Vision and Scope Document for PhysioMIST*
2. *PhysioMIST.* `http://robotics.case.edu/modeling_simulation_biological_systems.html`

2 Overall Description

2.1 Product Perspective

The PhysioMIST software is a new interface that eliminates the need to manually link different mathematical models when studying human physiology. In its first release, the system will link models on a one-to-one basis and provide integration results through a streamlined user interface. Subsequent releases will include the ability to link models on a many-to-one or one-to-many basis, and also provide a graphical representation of integration results.

2.2 User Classes and Characteristics

Researcher The Researcher is the intended user of the PhysioMIST system. She will be working in a field concerning human physiology, and have multiple mathematical models in need of integration. She will choose the models that she wishes to integrate, import them into PhysioMIST, and receive useful results as output.

2.3 Operating Environment

The PhysioMIST software is supported only on Microsoft Windows systems.
The computer on which PhysioMIST is to be run must have .NET 2.0 runtime files installed.

2.4 Design and Implementation Constraints

The PhysioMIST software is to be designed in Microsoft Visual Studio 2005 utilizing the .NET 2.0 libraries.

2.5 User Documentation

The system shall have a user manual or tutorial to help the new researcher become acquainted with the software.

As the project will be released under an open source license, there will also be a developer's manual, generated with Doxygen.

2.6 Assumptions and Dependencies

This project makes the assumption that the existing PhysioMIST codebase it depends on is easily extendable and capable of performing the integration and simulation processes.

3 System Features

3.1 Import Existing Models

3.1.1 Description and Priority

The system will be able to import models that have been previously created in the JSim MML standard.
Priority = High.

3.1.2 Stimulus/Response Sequences

Stimulus: User locates a JSim MML file he or she wants to import.
Response: System converts the file to its preferred format for further use.

3.1.3 Functional Requirements

Import.check: Determine if the file is valid in the JSim MML format.
Import.parse: Read the contents of the file and parse according to the JSim MML standard.

3.2 Save Anatomical Information

3.2.1 Description and Priority

The system will save anatomical information associated with model components.
Priority = High.

3.2.2 Stimulus/Response Sequences

Stimulus: User requests to save anatomical information.
Response: Information is saved on user's hard drive.

3.2.3 Functional Requirements

Save.Request: Once the user has clicked the save button in the GUI, the system will save the anatomical information for the currently displayed model in a pre-determined location on the user's hard drive.

3.3 Integrate Model Components on a One-to-One Basis

3.3.1 Description and Priority

The system will provide a graphical interface for users to choose components from two models to integrate on a one-to-one basis.
Priority = High.

3.3.2 Stimulus/Response Sequences

Stimulus: User inputs two models to the system.
Response: System integrates the components of these models and displays the result.

3.3.3 Functional Requirements

OnetoOne.Integrate: The system will use the one-to-one algorithm to integrate the two models.

3.4 Query for Related Model Components

3.4.1 Description and Priority

The system will provide a graphical interface for users to search for related model components while integrating models, based on associated anatomical information.
Priority = High.

3.4.2 Stimulus/Response Sequences

Stimulus: User selects a model component with associated anatomical information and the type of query to perform.
Response: The system performs the query and displays the related model components.

3.4.3 Functional Requirements

Query.query: System selects related model components using the anatomical ontology. Query.results: System displays the selected results.

3.5 Integrate Model Components on a One-to-Many Basis

3.5.1 Description and Priority

The system will provide a graphical interface for users to choose components from two models to integrate on a one-to-many basis.
Priority = Low.

3.5.2 Stimulus/Response Sequences

Stimulus: User inputs two models to the system and specifies the manner in which components should be integrated.
Response: System integrates the components of these models as specified and displays the result.

3.5.3 Functional Requirements

Onetomany.Integrate: The system will use the one-to-many algorithm to integrate the models.

3.6 Integrate Model Components on a Many-to-One Basis

3.6.1 Description and Priority

The system will provide a graphical interface for users to choose components from two models to integrate on a many-to-one basis.
Priority = Low.

3.6.2 Stimulus/Response Sequences

Stimulus: User inputs two models to the system and specifies the manner in which components should be integrated.
Response: System integrates the components of these models as specified and displays the result.

3.6.3 Functional Requirements

ManytoOne.Integrate: The system will use the many-to-one algorithm to integrate the models.

3.7 Display Simulation Results

3.7.1 Description and Priority

The system will provide a robust representation of the information contained in simulation results.
Priority = Low.

3.7.2 Stimulus/Response Sequences

Stimulus: User specifies the simulation parameters for a model.
Response: Display the results of the simulation graphically and textually.

3.7.3 Functional Requirements

Display.graph: The system will graphically show the results of a simulation.
Display.text: The system will display the simulation results as text.

4 External Interface Requirements

4.1 User Interface

The entire purpose of this project is the user interface. In general, it should behave in a manner expected of users of the operating system platform, conforming to a reasonable set of human interface guidelines. Discoverability is key for making sure users understand what operations are available and how they work, and also for keeping users once they have tried the software. Testing will be used to ensure that the various GUI elements perform the intended function.

4.2 Software Interface

The user interface will interact with two existing components: the PhysioMIST computational engine and JSim's MML (Mathematical Modeling Language) file format.

If the interaction with the existing PhysioMIST codebase is through a library, then it will also need to have a command-line program, to ease testing, scripting, and verification.

The MML import will be tested to ensure it is compatible with the file format of the existing JSim implementation.

5 Other Nonfunctional Requirements

5.1 Performance Requirements

The performance of the underlying software which does the comparison between the models is not under the control of this project. However, it ought to be able to report the time remaining in any long operation, thus an accurate progress bar shall be displayed, with an option to cancel if that makes sense. Whenever possible, a long operation shall not render the entire GUI unusable.

The GUI shall remain responsive when displaying and editing any data set of a size that can be effectively processed by the underlying algorithms.

5.2 Safety and Security Requirements

Data accountability is paramount in a scientific data processing application. Working under the assumption that the existing codebase correctly models the systems at hand, we aspire to create an interface that never obfuscates or misrepresents the operations being performed. Thus, all GUI commands should possess the ability to export to a plain-text script that the user can inspect to ensure that the input to the data layer will be correct. Test suites will be used to confirm the working order of individual functional units.

5.3 Software Quality Requirements

When presented with a corrupt or invalid data file, the software shall not crash but display an error message, possibly attempting to help the user recover the data.