PhysioMIST Software Architecture Design

Mark Caral, Sara Cummins, BarbaraJoy Jones, Joshua Lee

October 23, 2009
EECS 393

Version	Date	
0.1	October 2nd, 2009	Initial rough draft of all sections
0.2	October 10th, 2009	Additional use cases identified
0.3	October 20th, 2009	Revised use cases
0.4	October 21st, 2009	Revised architectural goals and constraints, size and performance, and quality sections
0.5	October 22nd, 2009	Revised logical, implementation, and data views
1.0	October 23rd, 2009	Release for use

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide a detailed presentation of the PhysioMIST architectural design as a guide for implementing the model integration interface.

1.2 Scope

This document provides detailed description of the architecture of the PhysioMIST model integration interface.

1.3 Glossary

MML: Mathematical Modeling Language, an industry standard for representing physiological models based on differential equations
Model: a collection of differential equations that can be used to simulate human physiology
Variable: an input or output of a model's differential equations

1.4 References

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

1.5 Overview

Architectural Representation: describes the views in the following sections
Architectural Goals and Constraints: describes the goals and constraints of the project's architecture
Use-Case View: describes typical use case scenarios and their impact on the architecture
Logical View: describes the one-to-one model integration use-case realizations; contains the Analysis and Design Models
Process View: describes concurrency aspects of the design
Deployment View: contains the Deployment Model
Implementation View: describes implementation details of the project's subsystems and functions
Data View: describes how the system handles persistent data; contains the Data Model
Size and Performance: describes performance issues and constraints
Quality: describes quality of service aspects of the project

2 Architectural Representation

This document details the architecture using the views defined on the "4+1" model [KRU41] but using the RUP naming convention. The views used to document the PhysioMIST application are:

2.1 Logical View

Audience: Designers
Area: Functional Requirements–describes the system's object model and most important use cases
Related Items: Design Model

2.2 Implementation View

Audience: Developers
Area: Software Components–describes implementation details of the project's subsystems and functions
Related Items: Implementation Model

2.3 Use-Case View

Audience: Project Team and end-users
Area: describes use cases and scenarios representing vital functionality
Related Items: Use-Case Model

2.4 Data View

Audience: Developers and end-users
Area: Persistence–describes how the system handles persistent data
Related Items: Data Model

2.5 Process View

Audience: Developers
Area: describes concurrency aspects of the design
Related Items: N/A

3 Architectural Goals and Constraints

This section describes the software requirements and objects that have some significant impact on the architecture.

3.1 Technical Platform

The PhysioMIST application requires the .NET 2.0 runtime. It can be used on the Windows XP, Vista, and Windows 7 operating systems. It may be possible to install and use the PhysioMIST application on MacOSX and Linux via Mono, but these platforms are not actively supported.

3.2 Security

Strict security measures are not necessary. The application does not support or require network access. It does not access any personal information.

3.3 Persistence

Data persistence will be addressed by allowing the user to save and load models.

3.4 Reliability and Availability

The application does not communicate with a remote server.

3.5 Performance

Querying based on anatomical structures will return a limited set of results to prevent negative effects on performance.

3.6 Internationalization

The PhysioMIST application will only be available in English in this release. Additional language support may be added in a future release.

4 Use-Case View

In the following diagrams, square boxes denote entities that interact with the system. Oval boxes represent actions and functionality of the system itself.

4.1 Loading Models

The user selects the “Load” menu item and then selects the file containing the desired model. The file may be a text file in the MML standard format or an XML file adhering to the PhysioMIST schema. The software parses and validates the file then displays the model’s data or gives an error if the file contents are not a valid model.

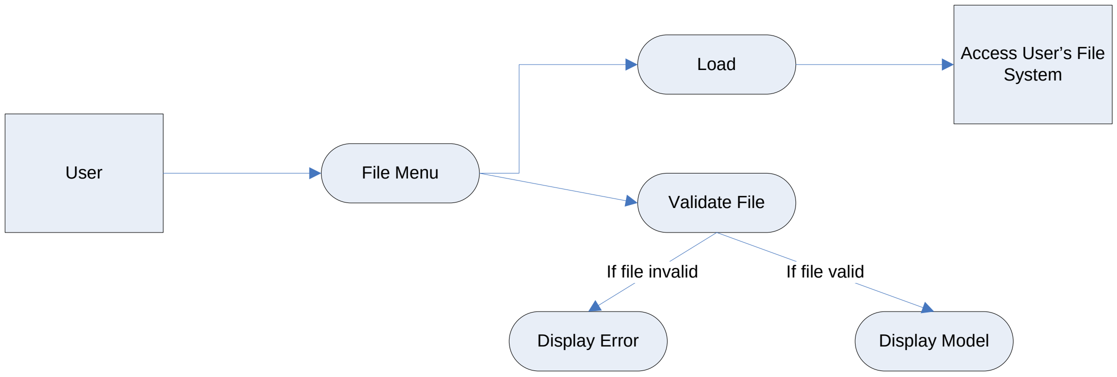

Figure 1: Loading Models

4.2 Saving Models

The user selects the “Save” or “Save As...” menu item. If the “Save” menu item is selected, the model is saved to the same file in the previously selected format. If the “Save As...” menu item is selected, the user must enter a file name and may optionally choose the file location and format. The default format is the PhysioMIST XML format. If the user selects the “Save” menu item for an unsaved model, the “Save As...” dialog is presented.

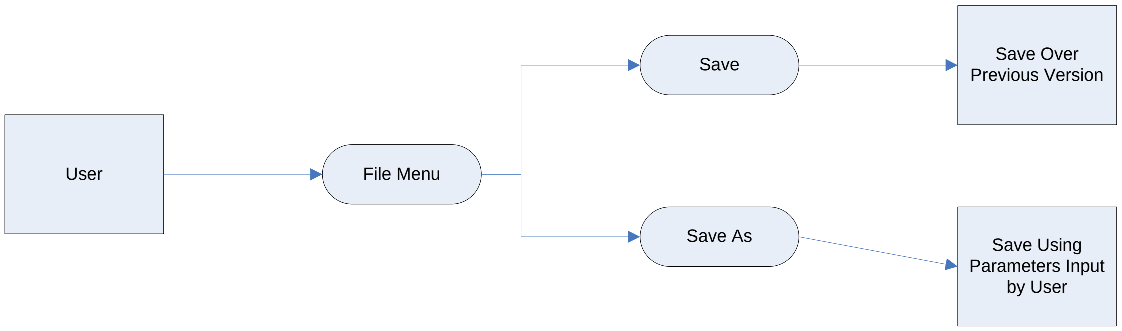
User
File Menu
Save
Save As
Save Over Previous Version
Save Using Parameters Input by User

Figure 2: Saving Models

4.3 Editing Models

4.3.1 Adding Variables

The user clicks the “New” button for the variable or parameter table. A dialog box with fields for the name, formula, value, units, anatomical structure, and description is displayed. The user enters the relevant data, the input is validated, and the new item is added to the appropriate table. Only the name, formula, and value fields are required.

4.3.2 Deleting Variables

The user selects an item in the variable or parameter table and clicks the “Delete” button. The item is deleted from the model.

4.3.3 Modifying Variables

The user selects an item in the variable or parameter table and clicks the “Edit” button. A dialog box as described above is displayed with the appropriate information in the relevant fields. The user modifies the data as needed, the input is validated, and the item is modified.

4.3.4 Associating Anatomical Information

When the user is editing an item in the model, the user selects an anatomical structure in the Anatomical Ontology treeview and clicks the “Associate Structure” button. The structure is added to the item or replaces the previous structure.

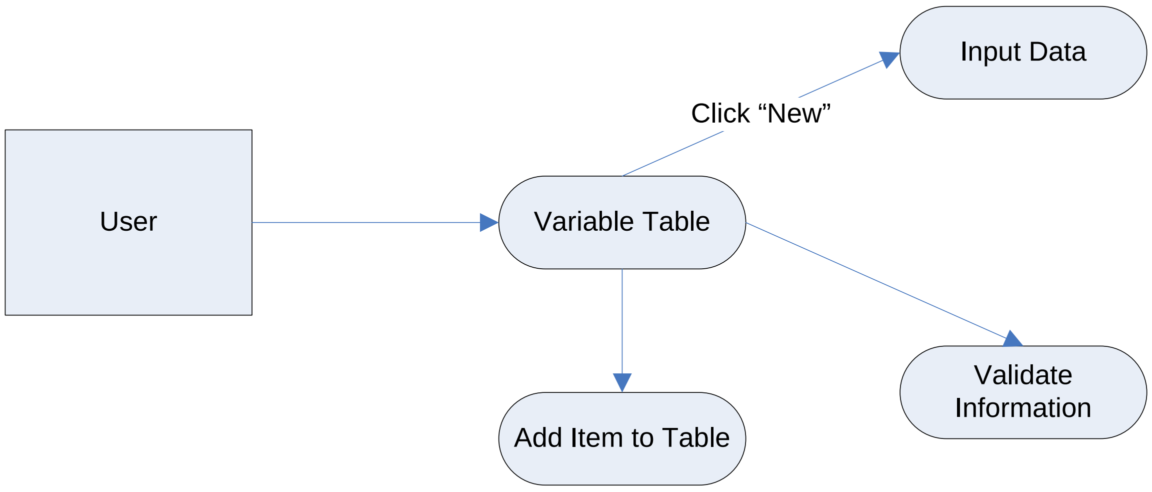

Figure 3: Adding Variables

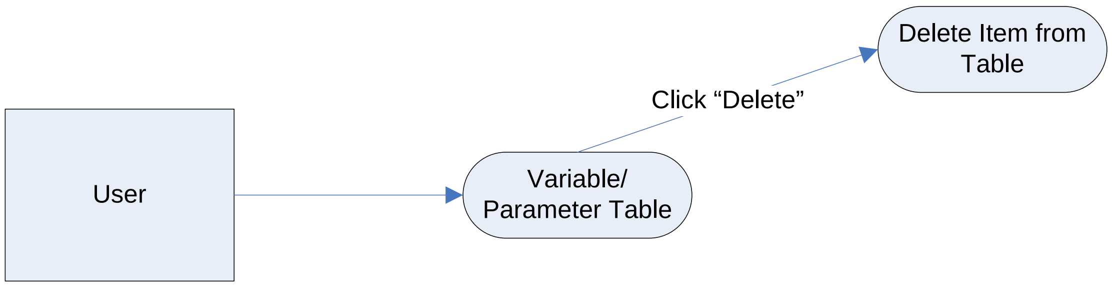

Figure 4: Deleting Variables

4.4 Model Integration

This section is only concerned with integrating two models. The ability to integrate more than two models at a time may be a feature added in a future release.

4.4.1 One-to-One Integration

The user selects two models, which are loaded by the software as described above. The models will be referred to as A and B for clarity. The user then selects an item from each model ($A.x$ and $B.y$) and defines the relationship between them where $A.x$ acts as an input to $B.y$. This can be expressed mathematically as $F(A.x) = B.y$. Note that $A.x$ may be a variable or parameter, but $B.y$ must be a variable. The operations of the integration subsystem will not be addressed within this document.

4.4.2 One-to-Many Integration

This is very similar to one-to-one integration. If a relationship between $A.x$ and $B.y$ exists, the user is not restricted from defining a relationship with $A.x$ as input and $B.z$ as output. If $A.x$ is a

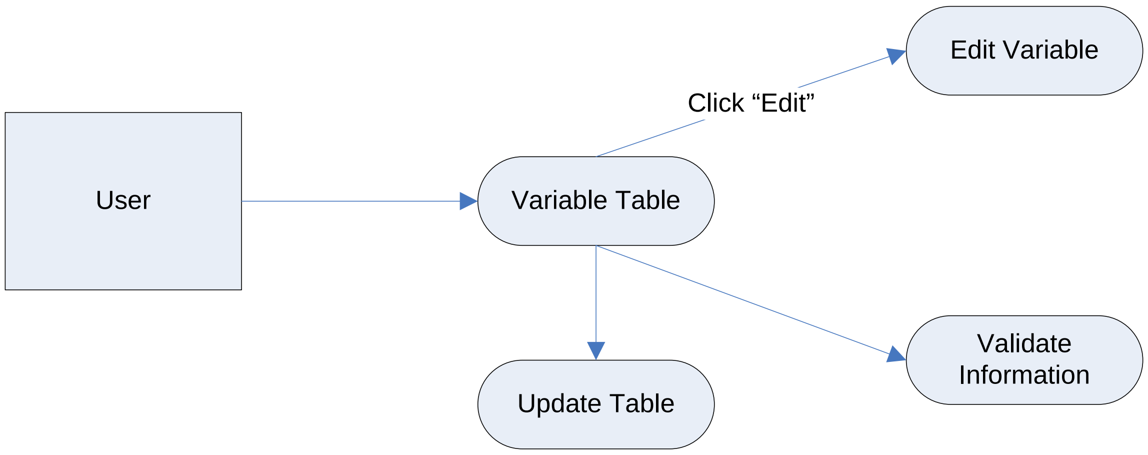

Figure 5: Modifying Variables

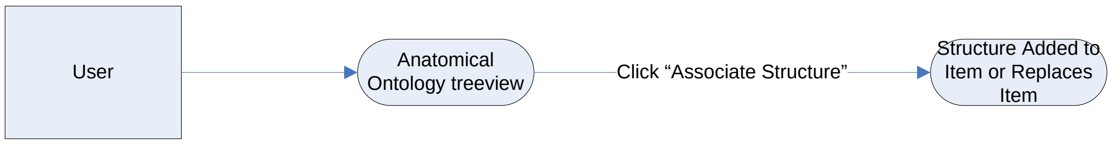
User
Anatomical Ontology treeview
Click “Associate Structure”
Structure Added to Item or Replaces Item

Figure 6: Associating Anatomical Information

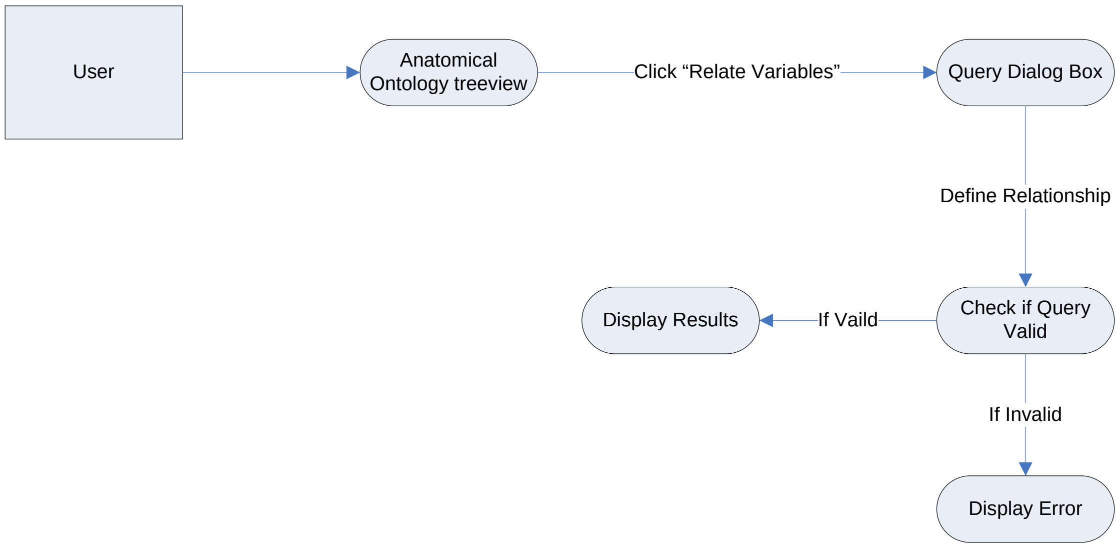

Figure 7: One-to-One Variable Integration

variable and $B.w$ is not dependent upon $A.x$, the user may also define a relationship with $B.w$ as input to $A.x$. Circular relationships are handled by the underlying integration subsystem.

4.4.3 Many-to-One Integration

This feature may be added in a future release.

4.5 Anatomical Queries

Possible query types:

- Develops
- Nerve Supply
- Arterial Supply
- Venous Drainage
- Part
- Attributed Part
- Regional Part
- Regional Part Of
- Constitutional Part
- Constitutional Part Of
- Custom Partonomy
- Custom Partonomy Of

4.5.1 Structure-based Queries

The user selects a structure in the Anatomical Ontology treeview and clicks the "Related Structures" menu item. The user then selects the type of relationship from the query dialog box. The software displays the results of the query (structures with the appropriate relationship to the selected structure) or an error if the relationship type is not applicable for the selected structure.

4.5.2 Variable-based Queries

While integrating models, the user selects a variable/parameter from one model and clicks the "Related Variables" menu item. The user then selects the type of relationship from the query dialog box. The software displays the results of the query (variables/parameters from both models that are associated with structures that have the appropriate relationship to the selected item's associated anatomical structure) or an error if the relationship type is not applicable for the selected item's associated anatomical structure. An error is also displayed if the selected item does not have an associated anatomical structure.

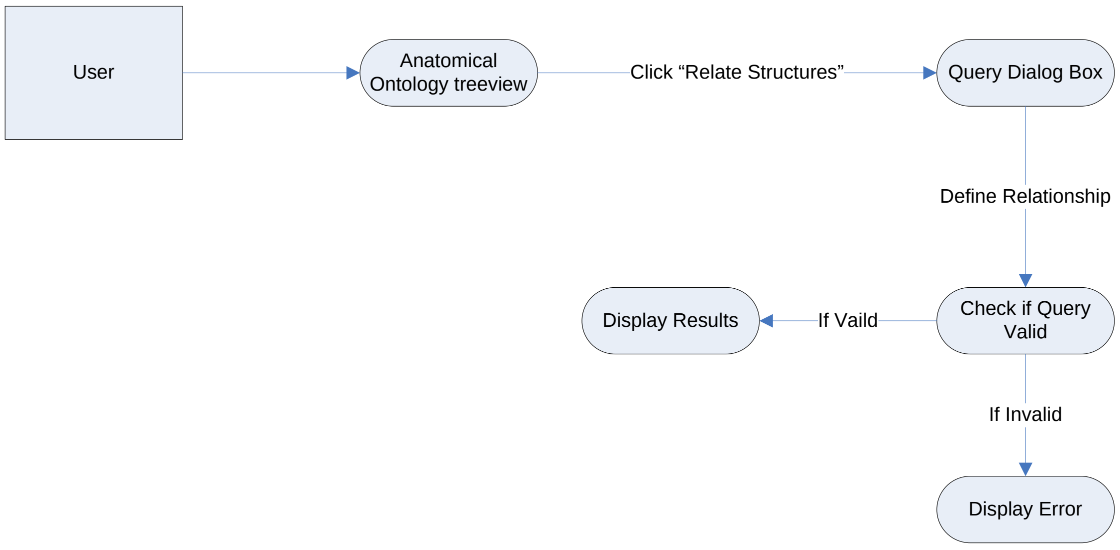

Figure 8: Structure-based Queries

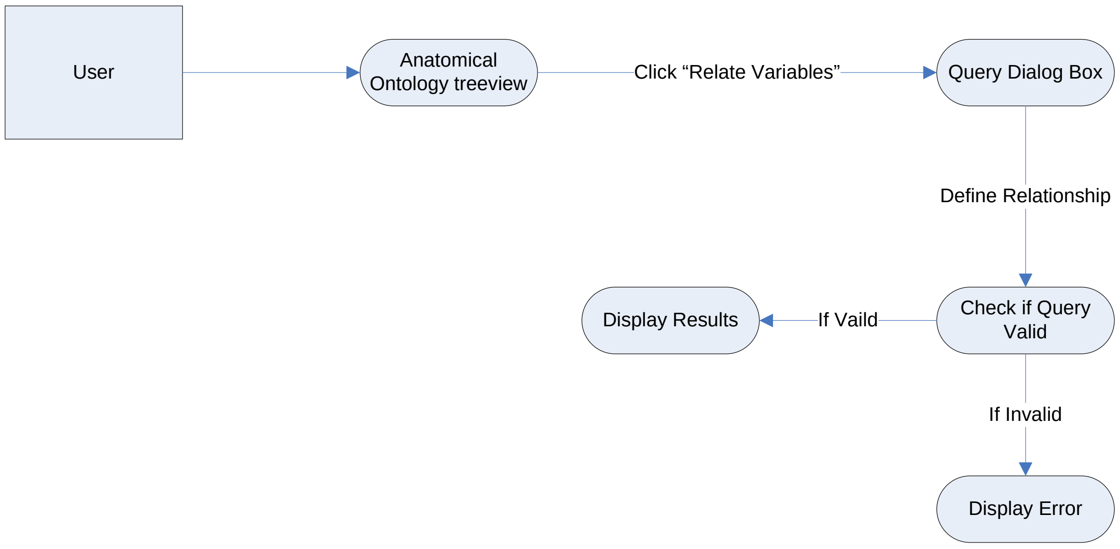

Figure 9: Variable-based Queries

5 Logical View

The PhysioMIST layering model is described in detail on the project website (see References, above). The model integration interface only depends on the query interface, the data layer, and the GUI subsystem.

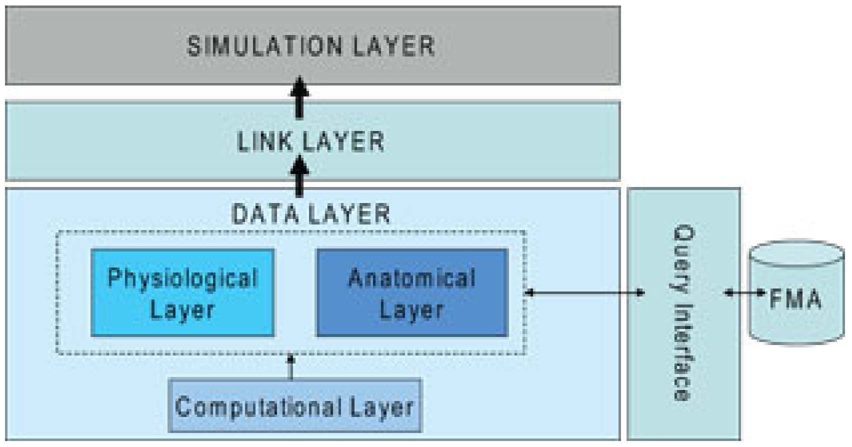

Figure 10: PhysioMIST Layers

6 Process View

The software only executes one process at a time. The software is not multi-threaded.

7 Deployment View

Once a user has the application on his or her computer, the user's computer can run the application without interacting with any external entities. The interactions involved in downloading the application will occur approximately as follows:

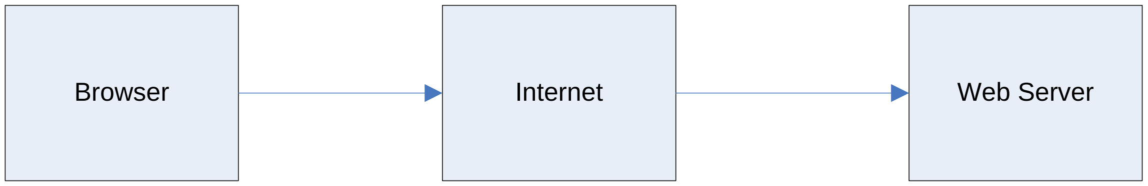

Figure 11: Deployment

8 Implementation View

The implementation view of the existing PhysioMIST application is as described in the layering view. The details of each subsystem, including inheritance and interaction requirements, can be inferred from additional documentation generated from the source code using Doxygen.

9 Data View

The PhysioMIST XML model format is validated by this schema:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="model_schema" targetNamespace="http://tempuri.org/model_schema.xsd"
elementFormDefault="qualified" xmlns="http://tempuri.org/model_schema.xsd"
xmlns:mstns="http://tempuri.org/model_schema.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="model">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="model_name" type="xs:string">
        </xs:element>
        <xs:element name="description" type="xs:string" minOccurs="0">
        </xs:element>
        <xs:element name="variable" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:ID" />
              <xs:element name="formula" type="xs:string" />
              <xs:element name="value" type="xs:float" />
              <xs:element name="units" type="xs:string" minOccurs="0" />
              <xs:element name="description" type="xs:string" minOccurs="0" />
              <xs:element name="anatomical_structure" minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="str_name" type="xs:string" />
                  </xs:sequence>
                  <xs:attribute name="FMAID" type="xs:int" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="constant" type="xs:boolean" use="required" />
            <xs:attribute name="type" type="var_type" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="var_type">
    <xs:restriction base="xs:string">
      <xs:enumeration value="realDomain" />
      <xs:enumeration value="real" />
      <xs:enumeration value="int" />
    </xs:restriction>
  </xs:simpleType>
```

```
</xs:schema>
```

10 Size and Performance

Volumes:
Although we hope that a large number of users will be interested in this application, the exact number of users does not matter since each user will have his or her own computer. The owner of the website on which PhysioMIST is released may to need to account for excessive downloads if the application becomes very popular.
Performance:
The time to integrate models of average size on a one-to-one basis should be less than one hour.

11 Quality

The following quality factors are important to PhysioMIST. As a result, related goals have been identified:
Scalability
Description: System's reaction when user demands increase
Solution: Owner of PhysioMIST's deployment website may need to allocation additional servers for download.
Reliability
Description: System's mean-time-between-failure
Solution: Failure is most likely to occur when integrating large datasets. This will be handled by limiting the size of datasets. Failure could also occur because of errors in the program which will be addressed with functional testing.
Simplicity/Understandability
Description: User's ability to comprehend and utilize the system quickly
Solution: Organize the GUI in a logical manner and include a User's Manual with the system
Efficiency
Description: System's space demands on the user's computer
Solution: Files will be saved as plain text or XML in order to minimize file size