SimTK Modeling Layer:

Requirements for SimTK 1.0

Version 1.0

Ayman Habib & Mike Sherman*

First DRAFT, August 2005

Abstract

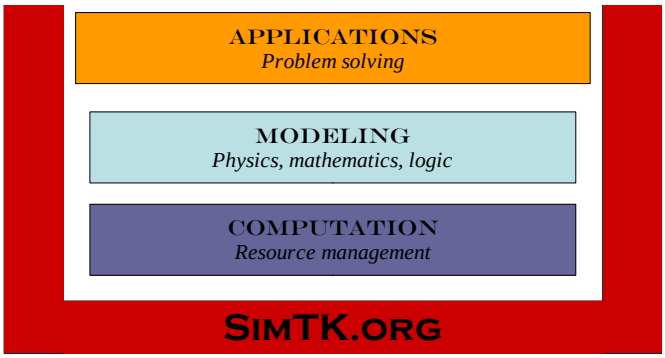
Here we discuss the requirements for the modeling layer in SimTK 1.0. First we outline the purpose of the modeling layer, intended users/user-groups, what was accomplished in SimTK 0.5 and where we are heading in SimTK 1.0.

1 Purpose of this document

This document provides background describing the Simbios needs addressed by the SimTK modeling layer, the challenges to writing a modeling layer that will be widely adopted, the progress that has been made toward writing it and the plans for the modeling layer in SimTK 1.0.

2 Relevance

- The three layer architecture: Applications, Modeling and Computation.



  - Applications are targeted toward end-users/clinicians and will have the biggest direct impact on medical research. Applications are generally narrowly targeted and may or may not depend on the lower layers, though if we were successful in designing the lower layers then more and more applications will use them.
  - The modeling layer targets modelers (researchers with engineering or physics expertise who can program in some form but are not expert programmers).

- Modelers understand the physics behind the applications and provide the logical view of the world along with the mathematics that governs it. Our task is to make it easy for modelers to put their models in a form where they can be correctly & efficiently executed and they can also be shared with others.
- Modelers are the customers of the modeling layer, but we need to keep in mind application developers as well since they will write applications that incorporate models and investigations written by modelers to make end-user applications.
- Algorithm developers work in the computation layer and are interested in making new efficient algorithms that hopefully are plug-and-play interchangeable with other algorithms/implementations. The following interactions between the modeling layer and the computational layer need to be considered:
    1. The algorithmic layout of data in computation layers should be such that any state is maintained external to the computation.
    2. The modeling layer needs to actively support the selection of computational components (via the "manifest" level state variables). SimTK 1.0 will produce some computational layer modules which will work in this form; see [1, 2].
- The modeling layer hence has the potential to have a great impact on the Simbios mission however we need to make sure we don't rush it since we get one chance for adoption and we need to make the best out of it.

3 Current status of the modeling layer

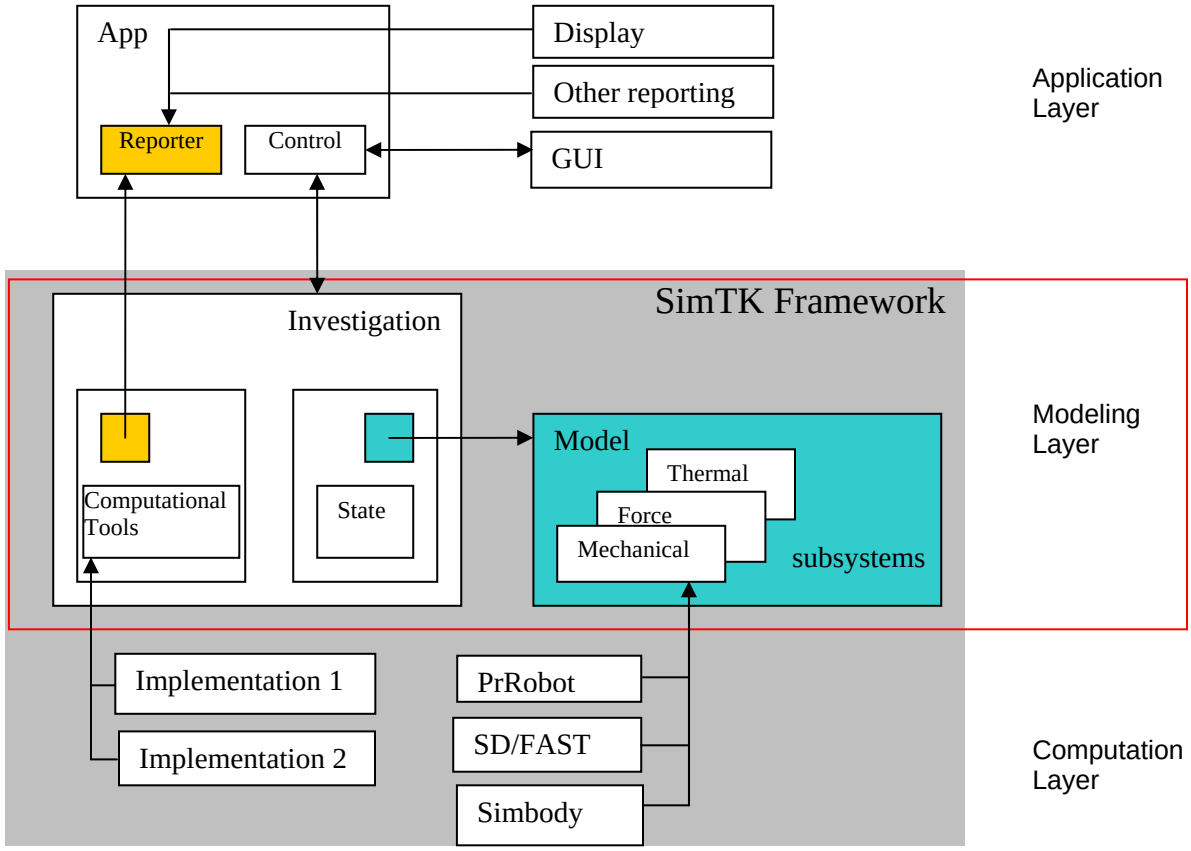
- Abstractions/Classes: The following set of abstractions have been identified and implemented as classes in SimTK modeling layer:

  - Model=Subsystem: A representation of the entities of interest that the user wants to simulate. A model is either made up of one subsystem, or a list of subsystems that interact. Subsystems can be hierarchical and so this definition is recursive. The breakdown of a model into subsystems is logical with every subsystem encapsulating all the knowledge it has about itself and exposing only a limited set of interfaces. A model is a stateless object (has no memory of its own).
  - Investigation: This is the main class driving a simulation. It instantiates the top model and keeps around instances of the state object as needed. The investigation knows nothing about the contents of the state but keeps an opaque reference to it. Investigations hand over the state to the model for interpretation.
  - Reporter: This is a class that writes out or displays results of the investigation at user-defined intervals or as triggered by events. Only a limited set of choices for report triggers have been implemented in SimTK 0.5. Reporters are the standard mechanism for recording results while running a simulation or giving a visual feedback on a simulation while it runs.
  - Stages: Computations are done on stages, to initialize the different subsystems or to do computations up to a specified stage. Different results are available at different stages.
  - Subsystems provide: responses, operators and solvers. These are available at defined stages.
  - State Object: An instance of the state object contains a list of "values" that represent some snapshot of the simulation. The model itself is a stateless entity that knows how to evaluate itself with respect to a specific state object instance. State object serves many functions:
    1. Hide the hierarchical representation of the model from its constituents so that every subsystem is self contained and is coded internally as a top level subsystem.
    2. State object also keeps track of dependencies so that if a state entry becomes invalidated then dependent entries are invalidated as well. If a user of the state object tries to access a state entry then that entry is either correct or the simulation will abort. The intention is to provide mechanisms so that only correct simulations can be produced using the SimTK modeling layer. State can be realized up to a specified stage programmatically by the model writer.

3. To avoid re-computations, the state object keeps a cache of evaluated quantities and guarantees that the cache is up-to-date or is invalidated as needed.

- State Client: Is an interface that needs to be implemented by all the classes that use a State object, or serve states to lower level objects.

- Usage:
  - The common formulation of any simulation process done using the SimTK modeling layer has an "investigation" that tries to analyze a "Model". The investigation refers to an instance of the model and employs one or more reporters to record quantities of interest while performing a simulation.



-

4 Desired features

- The biggest question regarding the Modeling Layer is if it's possible to write one that guarantees correct simulations and is not written in a special purpose modeling language (e.g. matlab/Simulink). This has not been done before in this generality using C++ as a modeling language and the main guarantee we want to provide is that simulations will both work and be correct or will blow-up. Finding out dependencies and making sure that these are correct and up-to-date is the biggest hurdle to making a modeling layer that can be widely adopted. The following list of features need to be observed
  1. Completeness: We need to come up with a set of classes/abstractions that capture, or can be extended to capture, the basic set of problems we plan to address in Simbios. This is a major task considering the disparity of the DBPs that SimTK is trying to address and the

open nature of these problems/DBPs. Significant work has been done on this front in SimTK 0.5 but more work is need.

2. Interchangeability of subsystems of similar nature by designing common interfaces that will be implemented by subsystems. Coming up with a core set of interfaces, for various kinds of subsystems, that the community can build on is a major undertaking that we'll tackle in SimTK 1.0 for the NMBL DBP. For example a "mechanical-subsystem interface" will be designed and that in itself should be helpful to users who want to try different implementations of this interface using different dynamics engines. Eventually we'll have to provide a mechanism for users of the modeling layer to find out available subsystems that implement a specific interface so that users can experiment with different implementations
3. Ease of use: exposing only the abstractions/types that make sense to modelers and hiding everything else.
4. Syntax should be easy to follow at the user level which we're going to assume to be a researcher with reasonable programming skills that can follow guidelines and use a published API.
5. Dependency checking to guarantee that we are not using stale values that lead to incorrect simulations.
6. Efficiency has to be built into the modeling layer so that no copying of data is done.
7. When a simulation fails because it was improperly constructed, the modeling layer will throw exceptions that can be caught by application developers to provide detailed messages about the failures and help modelers identify these problems.

5 What will be accomplished in SimTK 1.0

- Interfaces:
  - Formulate NMBL group needs as Models/Subsystems and Investigations that operate on them. This will lead to a set of interfaces for the different subsystems used by the NMBL group that should be close to sufficient to represent the Neuromuscular DBP's problems of interest.
  - Test convenience of current interfaces to NMBL modelers and application developers. Can we simplify the types/interfaces and or abstractions to fit NMBL needs with less complexity?
  - Work out with one other DBP (likely Molecular) the list of subsystems they use to avoid designing interfaces that are strictly NMBL oriented.
- Functionality:
  - Simplify the inner workings of the modeling layer to avoid complex syntax and minimize the potential for errors.
  - Chopping research problems: can we provide guidelines and/or examples for how to chop up models into subsystems (taking into account that this breakdown is not unique).
  - There's a set of supporting objects/abstractions that do not fit cleanly into the Model/Investigation/Reporter Trio paradigm but still represent concepts that we'd like to model (e.g. Controllers). We will make sure that these concepts can be implemented using the modeling layer abstractions and/or extend the modeling layer abstractions to be able to represent them.
- We will publish a final report summarizing our accomplishments and some guidelines to using the modeling layer at the end of SimTK 1.0 in order to help speed up adoption (internally first among other DBPs and with collaborators and then externally to the rest of the target user community).
- We will provide some executable examples which use the resulting versions of the modeling layer and have these available at least for internal users.

6 What are we leaving out in 1.0?

- Performance tuning: Performance is an essential part of the design of the modeling layer and can't be ignored. We are designing the modeling layer with performance in mind. However, performance tun-

ing and measurement/profiling will be left out of SimTK 1.0 until the interfaces and APIs are ironed out.

- Meta-applications: Meta-applications are applications that will target modelers with the intent of automating parts of the process of model creation. We believe that writing models directly using the modeling layer abstractions/classes should be made substantially stable before we start automating the process using a meta-application. If we succeed in making the modeling layer API meet the set of desired features above, we should have no problem writing a program to automate parts of the model creation process. The experience we gain building models using the API should come in handy if/when we decide to write this meta-application.
- Multithreading the modeling layer: There is a modeling layer-specific issue regarding threading. Because the subsystems are defined to be stateless (meaning that, once constructed, they maintain no persistent memory between calls) they depend on a passed-in state at every call. If threads maintain separate states, then this will provide perfect thread safety. However, if threads share the state we will have to think carefully about thread safety. For SimTK 1.0 we shall avoid this altogether by limiting multithreaded behavior to the computational layer. That is, there would be only a single modeling thread but it might call multithreaded numerical components.

7 Acknowledgments

This work was funded by the National Institutes of Health through the NIH Roadmap for Medical Research,[1] Grant U54 GM072970. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics.

8 References

1. The SimTK Multibody Dynamics Toolset.
2. SimTK 1.0 Requirements: Numerical Methods.

[1] Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics.