GiPSi: An Open Source/Open Architecture Software Development Framework for Surgical Simulation

Tolga G. Göktekin[2], M. Cenk Çavuşoğlu[1],
Frank Tendick[3], and Shankar Sastry[2]

[1] Dept. of Electrical Eng. and Computer Sci., Case Western Reserve University
[2] Dept. of Electrical Eng. and Computer Sci., University of California, Berkeley
[3] Dept. of Surgery, University of California, San Francisco

Abstract. In this paper we propose an open source/open architecture framework for developing organ level surgical simulations. Our goal is to facilitate shared development of reusable models, to accommodate heterogeneous models of computation, and to provide a framework for interfacing multiple heterogeneous models. The framework provides an intuitive API for interfacing dynamic models defined over spatial domains. It is specifically designed to be independent of the specifics of the modeling methods used and therefore facilitates seamless integration of heterogeneous models and processes. Furthermore, each model has separate geometries for visualization, simulation, and interfacing, allowing the modeler to choose the most natural geometric representation for each case. I/O interfaces for visualization and haptics for real-time interactive applications have also been provided.

1 Introduction

Computer simulations have become an important tool for medical applications, such as surgical training, pre-operative planning, and biomedical research. However, the current state of the field of medical simulation is characterized by scattered research projects using a variety of models that are neither inter-operable nor independently verifiable models. Individual simulators are frequently built from scratch by individual research groups without input and validation from a larger community. The challenge of developing useful medical simulations is often too great for any individual group since expertise is required from different fields. The motivation behind this study is our prior experience in surgical training simulators and physically based modeling [10, 11].

The open source, open architecture software development model provides an attractive framework to address the needs of interfacing models from multiple research groups and the ability to critically examine and validate quantitative biological simulations. Open source models ensure quality control, evaluation, and peer review, which are critical for basic scientific methodology. Furthermore, since subsequent users of the models and the software code have access

to the original code, this also improves the reusability of the models and interconnectibility of the software modules. On the other hand, an open architecture simulation framework allows open source or proprietary third party development of additional models, model data, and analysis and computation modules.

In this paper we propose GiPSi (General Interactive Physical Simulation Interface), an open source/open architecture framework for developing surgical simulations such as interactive surgical training and planning systems. The main goal of this framework is to facilitate shared model development and simulation of organ level processes as well as data sharing among multiple research groups. To address these, we focused on providing support for heterogeneous models of computation (e.g. differential equations, finite state machines and hybrid systems) and defined APIs for interfacing various heterogeneous physical processes (e.g. solid mechanics, fluid mechanics and bioelectricity). In addition, I/O interfaces for visualization and haptics for real-time interactive applications have been provided. The implementation of the framework is done using C++ and it is platform independent.

An important difference of GiPSi from earlier object-oriented tools and languages for modeling and simulation of complex physical systems, such as Modelica [8], Matlab Simulink [7], and Ptolemy [3], is its focus on representing and enforcing time dependent spatial relationships between objects, especially in the form of boundary conditions between interfaced and interacting objects. The APIs in GiPSi are also being designed with a special emphasis on being general and independent of the specifics of the implemented modeling methods, unlike earlier dynamic modeling frameworks such as SPRING [9] or AlaDyn-3D [6], where the underlying models used in these physical modeling tools are woven into the specifications of the overall frameworks developed. This allows GiPSi to seamlessly integrate heterogeneous models and processes, which is not possible with the earlier dynamic modeling frameworks [5].

2 Overview

The goal of GiPSi is to provide a framework that facilitates shared development that would encourage the extensibility of the simulation framework and the generality of the interfaces allowing components built by different groups and individuals to plug together and reused. Therefore, modularity through encapsulation and data hiding between the components should be enforced. In addition, a standard interfacing API facilitating communication among these components needs to be provided.

We are developing our tools on a specific test-bed application: the construction of a heart model for simulation of heart surgery. This test-bed model captures the most important aspects of the general problem we are trying to address: *i)* multiple heterogeneous processes that need to be modeled and interfaced, and *ii)* different levels of abstraction possible for the different processes. In the heart surgery simulation, several different processes, namely physiology, bioelectrical activity, muscle mechanics, and blood dynamics, need to modeled. Physiological

processes regulate the bioelectrical activity, which, in turn, drives the mechanical activity of the heart muscle. Muscle dynamics, coupled with the fluid dynamics of the blood, determine the resulting motion of the heart [2]. Models for all these processes need to be intimately coupled: the mechanical and fluid models through a boundary interaction, and the electrochemical and mechanical models through a volume interaction.

The overall system architecture of GiPSi is shown in Fig. 1. The models of physical processes such as muscle mechanics of the heart are represented as Simulation Objects (Sect. 3). Each simulation object can be derived from a specific computational model contained in Modeling Tools such as finite elements, finite differences, lumped elements etc. The Computational Tools provide a library of numerical methods for low level computation of the object's dynamics. These tools include explicit/implicit ordinary differential equation (ODE) solvers, linear and nonlinear algebraic system solvers, and linear algebra support. The objects are created and maintained by the Simulation Kernel which arbitrates their communication to other objects and components of the system (Sect. 6). One such component is the I/O subsystem which provides basic user input provided through the haptic interface tools and basic output through visualization tools (Sect. 4). There are also Auxiliary Functions that provide application dependent support to the system such as collision detection and collision response tools that are widely used in interactive applications (Sect. 5)

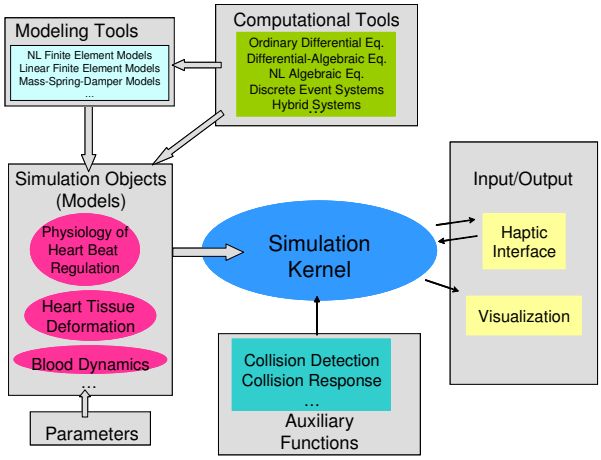

Fig. 1. The system architecture of GiPSi

3 Simulation Objects

In this framework, organs and physical processes associated with them are represented as Simulation Objects. These objects define the basic API for simulation, interfacing, visualization and haptics (see Fig. 2a).

Each Simulation Object can be a single level object implementing a specific physical process or can be an aggregate of other objects creating a hierarchy of models. For example, if we were interested only in muscle model of a beating heart, then we would define the heart as a single object that simulates the muscle mechanics. However, if we were to model a more sophisticated heart with both muscle and blood models, then our heart object would be an aggregate of two objects, one implementing the muscle mechanics and the other implementing the blood dynamics. The specific coupling of these muscle and blood objects would be implemented at their aggregate heart object (see Fig. 2c).

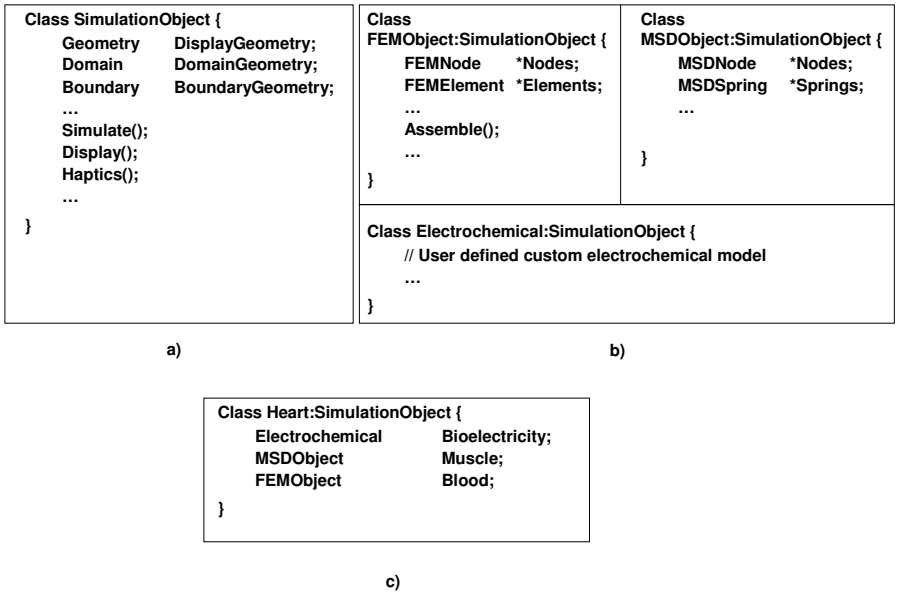

Fig. 2. a) Simulation Object, b) Examples of modeling tool and user defined objects, c) Heart object

The majority of the models in organ level simulations involve solving multiple time varying PDEs that are defined over spatial domains and are coupled via boundary conditions, e.g. a structural model representing the heart muscles coupled with a fluid model representing the blood which share the inner surface of the heart wall as their common boundary. Our goal is to design a flexible API that facilitates the shared development and reuse of models based on these

PDEs. Therefore the focus of our effort is to provide: *i)* a common geometric representation of the domain, *ii)* a library of tools for solving these PDEs, *iii)* a standard API for coupling them.

3.1 Simulation API

The first step in solving a continuous PDE is to discretize the spatial domain it is defined on. Therefore, every object must contain a proper geometry that describes its discretized domain, called the *Domain Geometry*. The definition of this geometry is flexible enough to accommodate the traditional mesh based methods as well as point based (mesh free) formulations. GiPSi defines a set of geometries that can be used as a domain including but not limited to polygonal surface and polyhedral volume meshes. In our current implementation we provide geometries for triangular and tetrahedral meshes.

Second, a method for solving a PDE should be employed such as Finite Element Methods (FEM), Finite Difference Methods (FDM) or Mass-Spring-Damper (MSD) methods. Basic general purpose objects that implement these methods are provided as Modeling Tools, e.g. there is a general customizable FEM object that implements the basics of the finite element method (see Fig. 2b). For example, an FEM based fluid model with linear elements can be modeled as an FEM object with a tetrahedral volume mesh as its Domain Geometry and with Navier-Stokes equations as its user defined PDE. So far we have implemented objects for FEM and MSD methods. GiPSi also provides a library of numerical analysis tools in the Computational Tools that can be used to solve these discretized equations. Our current implementation provides explicit and implicit integrators, some popular direct and iterative linear system solvers and C++ wrappers around a subset of BLAS and LAPACK functions [1].

3.2 Interfacing API

In addition to representing the domain geometry and assigning a method of computation, the simulation API also needs to provide a standard means to interface multiple objects. In the models mentioned above, the basic coupling of two objects are defined via the boundary conditions between them. Therefore, we need to provide an API to facilitate the passing of boundary conditions between different models. First, we need a common definition of the boundary, i.e. each object needs to have a specific *Boundary Geometry*. In our current implementation, we chose triangular surfaces as our standard boundary geometry. Even though the type of the boundary geometry is fixed for every object, the values that can be set at the boundary and their semantics are up to the modeler and should be well documented. Moreover, it is also the developer's task to interface two objects with different semantics on the boundary. For example, a generic fluid object can compute velocities and pressures on its boundary. In order to interface it with a structural object that requires forces on its boundary as boundary conditions, the developer needs to convert the boundary pressure values to boundary forces by integrating the pressure on the boundary.

Use of boundary conditions is not the only interfacing scheme for objects. For example, the coupling between the electrochemical and mechanical models (excitation-contraction coupling) in the heart is through the commonly occupied volume rather than a shared boundary. A more general information passing is provided by a simple Get/Set scheme, i.e. an object can read and write values inside another object by simply using Get(value) and Set(value) methods provided by the object respectively. The set of values that can be get and set by other objects and their semantics are again left to the modeler. In the above example, the electrochemical model sets the internal force values of mechanical model based on the excitation level which in turn result in the contraction of the muscles.

Both interfacing through a surface via boundary conditions and interfacing through a volume (domain) via the Get/Set scheme are achieved by the use of the *Connector* classes. Since the connection of two arbitrary models is application dependent, it is the modeler's task to develop these connectors. Fig. 3 shows two connector classes that interface three basic models contained in the aggregate Heart model. The first connector class provides basic communication between the Bioelectrical and Muscle models through their volumetric domain. It basically *gets* the excitation levels from the Bioelectric models (Domain 1), converts them to stress and *sets* the stress tensor values in the Muscle model (Domain 2). The second connector interfaces the Lumped Fluid Blood model with the Muscle model through their surfaces via boundary conditions. In this example the communication is in both ways. The connector class reads the displacement values on the Muscle boundary (Boundary 1), converts them into velocity and passes the velocities to Fluid model (Boundary 2) as boundary conditions. Similarly it receives the boundary pressure values from Boundary 2, converts them into forces and passes them to Boundary 1 as traction values on the boundary.

3.3 Visualization API

In order to display an object we again need a geometry dedicated for visualization. This geometry is called the *Display Geometry* and can be of any type of geometry defined in GiPSi. Each display geometry has a *Display Manager* associated with it. Display managers convert the data in geometries into a standard format used by the visualization module where the actual display takes place (see Sect. 4.2 for details). This makes the development of visualization tools and development of models mutually exclusive and ensures the modularity and the flexibility of the system.

3.4 Haptics API

Haptic interfacing with the simulation object uses the multi-rate simulation method proposed by Çavuşoğlu in [4]. In this method, each simulation object in haptic interaction provides local dynamic and geometric models for the haptic interface. The local dynamic model is a low-order linear approximation of the full deformable object model, constructed by the simulation object from the full

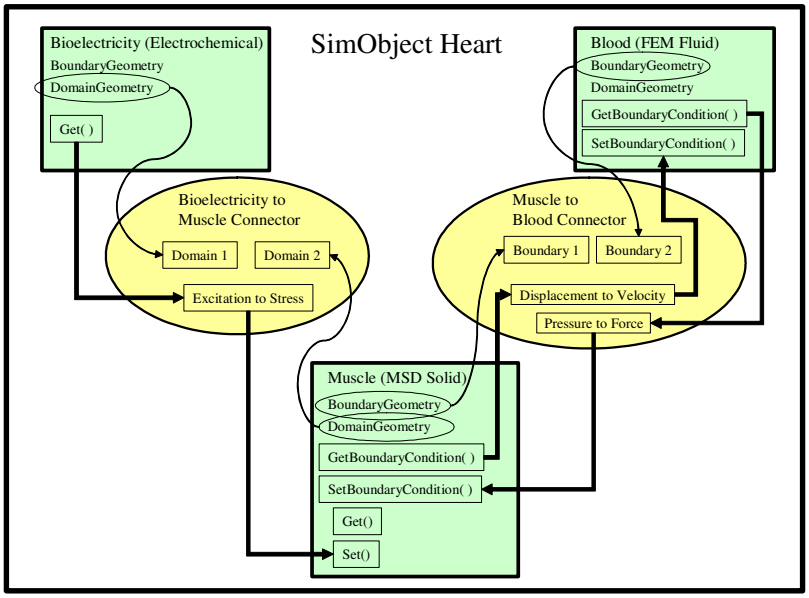

Fig. 3. Connector class example

model at its update intervals, and the local geometric model is a planar approximation of the local geometry of the simulation object at the haptic interfacing location. These local models are used by the haptic interface, running at a significantly higher update rate than the dynamic simulations, for estimating the inter-sample interaction forces and inter-sample collisions.

4 Input/Output Subsystem

The Input/Output subsystem provides basic tools for interacting with the objects. Currently, GiPSi provides haptics tools for input and visualization tools for output. These tools provide modularity and encapsulation of data, and define a standard API for model developers.

4.1 Haptics

Haptic interfaces require significantly higher update rates, usually in the order of 1 kHz, than are possible for the rest of the physical models, which are typically run at update rates in the order of 10 Hz. It is not possible to increase the update rate of the physical models to the haptic rate with their full complexity due to computational limitations, or to decrease the haptic update rate to physical model update rates due to stability limitations. As described in section 3,

GiPSi handles this conflicting requirements using a multi-rate simulation scheme [4]. The Haptic I/O module completely encapsulates the haptic interface and its real-time update rate requirements, and provides a standard API for all of the simulation objects which will be haptically interactive. The interface between the haptic I/O module and the simulation objects is through the local dynamic and geometric models provided by the simulation objects, and the haptic instrument location and interaction forces provided by the haptic I/O module. The instrument-object interaction forces are applied to the objects through the object boundary conditions and the instrument-object collision detections are handled no differently than the regular object-object collisions.

4.2 Visualization

Visualization of an object involves displaying the geometry of the object on the screen. In our current implementation we use OpenGL for display. The geometry to be displayed is defined in the object as discussed in Sec.3.3. However, to assure modularity, the object converts its geometry data into a standard form using the display manager associated with the type of geometry it has. Then the visualization tool accesses this data through the object pool maintained by the simulation kernel and displays it. In our current design, the standard format used is simply the list of vertex positions, vertex normals, vertex colors and connectivity information.

5 Collision Detection/Collision Response

In interactive surgical simulations one needs to detect collisions to prevent penetration between objects in the system, such as organ models and tools used during surgery. Therefore collision detection (CD) and collision response (CR) play a very important role. In our framework, CD module detects the collisions between boundary geometries of different models and the CR module computes the required response to resolve these collisions in terms of displacements and/or penalty forces and communicates the result to the models as displacement or force based boundary conditions. The models process these boundary conditions if necessary and iterate. As a result, the mechanics of contact detection and resolution becomes transparent to the model developer.

6 Simulation Kernel

The simulation kernel acts as the central core where everything above comes together. Its tasks include the management of the top level object pool, coordination of the object interactions, and arbitration of the communication between the components. The part which coordinates the top level objects is provided by the user. This coordination involves specifying the execution order of the models and the specific interfacing between them, allowing the user to properly interpret the semantics of the individual top level objects and the interfacing between

them, based on the specific application that the simulation is being developed for.

7 Conclusion

We have presented an open source/open architecture framework for organ level simulations that facilitates shared development and reuse of models. This framework provides an intuitive API for interfacing dynamic models defined over spatial domains. In addition, it is independent of the specifics of modeling methods and thus facilitates seamless integration of heterogeneous models and processes. Furthermore, each model has separate geometries for visualization, simulation, and interfacing. This lets the modeler choose the most natural geometric representation for each.

We want to emphasize that the framework proposed in this paper is a work in progress. It is intended to be a draft that will be modified according to the feedback we receive from the broader surgical simulation community. As we indicated throughout the paper, the implementation itself is incomplete and is only presented as a proof of concept. If the framework is adopted, the implementation can easily be extended by the community. Therefore, we plan to have a meeting with the interested parties at ISMS to discuss the future of the framework.

8 Acknowledgements

This research was supported in part by National Science Foundation under grants CISE IIS-0222743, CDA-9726362 and BCS-9980122, and US Air Force Research Laboratory under grant F30602-01-2-0588. We also would like to thank Xunlei Wu for his valuable discussions and feedback.

References

1. E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' guide (3rd ed.)*. SIAM, 1999.
2. R. M. Berne and M. N. Levy, editors. *Principles of Physiology*. Mosby, Inc., St. Louis, MO, third edition, 2000.
3. J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *Int. Journal of Computer Simulation special issue on Simulation Software Development*, 1994.
4. M. C. Çavuşoğlu and F. Tendick. Multirate simulation for high fidelity haptic interaction with deformable objects in virtual environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2000)*, pages 2458–2465, April 2000.
5. S. Cotin, D. W. Shaffer, D. A. Meglan, M. P. Ottensmeyer, P. S. Berry, and S. L. Dawson. CAML: A general framework for the development of medical simulations. In *Proceedings of SPIE Vol. 4037: Battlefield Biomedical Technologies II*, 2000.
6. A. Joukhadar and C. Laugier. Dynamic simulation: Model, basic algorithms, and optimization. In J.-P. Laumond and M. Overmars, editors, *Algorithms For Robotic Motion and Manipulation*, pages 419–434. A.K. Peters Publisher, 1997.

7. Mathworks, Inc. Simulink. http://www.mathworks.com/products/simulink/.
8. *Modelica — A Unified Object-Oriented Language for Physical Systems Modeling; Language Specifications 2.0.* The Modelica Association, 2002. http://www.modelica.org/.
9. K. Montgomery, C. Bruyns, J. Brown, S. Sorkin, F. Mazzella, G. Thonier, A. Tellier, B. Lerman, and A. C. Menon. Spring: A general framework for collaborative, real-time surgical simulation. In J. Westwood et al., editor, *Medicine Meets Virtual Reality (MMVR 2002)*, Amsterdam, 2002. IOS Press.
10. F. Tendick, M. Downes, T. Goktekin, M. C. Çavuşoğlu, D. Feygin, X. Wu, R. Eyal, M. Hegarty, and L. W. Way. A virtual environment testbed for training laparoscopic surgical skills. *Presence*, 9(3):236–255, June 2000.
11. X. Wu, M. S. Downes, T. Goktekin, and F. Tendick. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. In *Proceedings of the EUROGRAPHICS 2001*, September 2001.