13 Infrastructure

In previous years, we have included SCIRun/BioPSE as a Core in our Center. The focus of that Core was building the software architecture, software testing framework, software documentation templates, and software distribution mechanisms for our Center’s suite of software applications. With Infrastructure as a separate heading under the new BTTR P41 structure, we have migrated those components of our Center under this new heading. While our current software engineering systems are robust and well-maintained, the DBPs and TRDs included in this proposal bring new software requirements. We believe that these requirements can best be met by extending our Center’s software infrastructure, as described below.

13.1 Objectives

The Infrastructure for the Center will serve three major purposes: (1) to transform technology developed at the Center into usable software products for our collaborators and for the biomedical community (Translational Software), (2) to provide a common framework for the applications developed at the Center (Application Framework), and (3) to develop additional core software infrastructure to support the Center’s TRDs and DBPs (Core Functionality).

13.1.1 Translational software aims:

- **Aim 1**: Transform new algorithms developed in the technical cores into stable, available, documented software projects.
- **Aim 2**: Improve usability of tools for integration in the image-to-simulation pipeline.

13.1.2 Application framework aims:

- **Aim 3**: Render a common software base for (1) prototyping tools, (2) stand-alone dedicated applications, and (3) scriptable tools.

13.1.3 Core functionality aims:

- **Aim 4**: Add provenance, scripting, and database support into all of our core software applications.
- **Aim 5**: Add support for large data sets to all of our core software applications.
- **Aim 6**: Provide infrastructure for Streaming / GPGPU Architectures.

13.2 Rationale

13.2.1 Translational software

As a National Resource, one of the primary services we provide is our suite of software tools. To best serve the scientific community, these tools must be robust, well-documented, and easy to obtain. There is a significant difference between the research / proof-of-concept software produced by labs for internal use and the robust applications that users expect when they download a software product from the web. Once the potential of a new technology has been established, significant amounts of additional software engineering are required to generate a robust application that interfaces well with the rest of the framework and runs stably on all major
computer platforms. The additional software engineering includes: cleaning-up the application’s source code; refactoring the application for integration with the Center’s standard framework components; documentation of the application’s functionality; and a rigorous set of white- and black-box testing to be constructed around the application. Although these software engineering steps do not necessarily add any novel functionality to the programs, they are necessary to ensure the sustainability of the software outside of the Center.

The Center’s technology research and development cores (TRDs) provide novel solutions for image processing, mesh generation, simulation, estimation, and visualization. Ultimately, these solutions must be tested in a clinically relevant setting for a fair evaluation of the results. For example, one can test a meshing algorithm by means of a mesh quality metric, but it must be applied to real physiology for a clinically relevant simulation to be truly validated. Furthermore, the research algorithms we develop at our Center only fill in portions of the image-based modeling, simulation, and visualization pipeline. The Center’s Infrastructure team will implement additional algorithms from the literature to fill in any remaining gaps.

In addition to developing and supporting our suite of SCIRun-based software packages, our Center has also supported the Open Source Teem project. Teem was originally developed by Dr. Gordon Kindlmann when he was a graduate student at the SCI Institute, supported in part by our Center. The Teem tools are a collection of libraries and command-line applications for representing, processing, and visualizing raster datasets. One of the major contributions of Teem is the NRRD file format, a light-weight, flexible file format for storing raster datasets. Because the NRRD file format has been adopted into a variety of national and international tools, including those from our Center, as well as NA-MIC and ImageJ, it ensures that the tools from these different centers can be used in the same framework as our tools. Continuing beyond Dr. Kindlmann’s tenure at Utah, our Center continues to provide ongoing software engineering support for the Teem package. Most recently we have added a regression testing infrastructure for Teem.

13.2.2 Application framework

Each of the driving biological projects (DBPs) in this proposal are motivated, in part, by the use of patient-specific imaging data for the creation of patient-specific models to deliver personalized diagnoses. The creation of such models generally requires a “chain” of tools, e.g. registration tools, segmentation tools, visualization tools, analysis tools, etc. Depending on the DBP, the set of tools our Center develops may serve as the entire tool chain or as part of a pipeline, filling in the gaps for which no readily available solution is present. Thus, the tools need to be exposed through a variety of interfaces, ranging from command-line scriptable tools for projects that are primarily script-based, (e.g. DBP: Bayesian Source Imaging of Pediatric Epilepsy, see section 21), to applications with complete user interfaces, (e.g. DBP: Simulation of Electric Stimulation for Bone Growth, see 18). In order to support these different software paradigms, we will layer our application framework and expose a consistent set of API entry points at each layer in the application hierarchy.

13.2.3 Core functionality

The proposed DBPs specifically require software tools that support both flexible rapid prototyping and efficient batch processing. Patient-specific modeling is one motivating example. As the first step, one prototypes different image processing, meshing, and model generation techniques, generating an initial working patient-specific model; as the second step, one then repeatedly applies the resulting pipeline many times over for different patients to validate the proposed method in a clinical setting. This process suggests the need for quick prototyping tools, as well as tools to keep track of provenance and to convert that provenance information into easy-to-use scripts able to replay a user’s session. Past experience has shown that the tools needed for prototyping, such as our SCIRun framework, are not laid out to perform both tasks efficiently. Therefore, these tools must be augmented by scripting and provenance tools to enable rapid execution of both stages of the process.

We will continue to provide the application framework and the translational tools for migrating newly developed technology into new end-user applications; we will also extend our existing software infrastructure to support the new technology developments of the TRDs. For example, adding support for very large datasets will require additional memory management techniques and strategies for effectively dealing with out-of-core data. In addition to large data support, we will also add infrastructure components for capturing provenance, executing scripts, and managing streaming/GPU architectures.
13.3 Background and preliminary results

13.3.1 Translational software

To increase the impact of our software within the biomedical computing community, we have moved away from a software distribution model, where we only provide source code to users, as it has proven an insurmountable barrier to wider adaptation of our software. In our current distribution model, we provide binary versions of our software (for Windows and Mac OS X) directly from our software portal. Adding support for the Windows platform was another major undertaking that we have completed. The Windows port was especially significant because, even though Unix platforms are prevalent in Computer Science departments, they are almost non-existent in the clinical environment. As part of adding Windows support, our development team was forced to develop part of the software on Windows, which led to a cleaner, more portable, and a more sustainable source code base. This in turn has reduced the maintenance overhead for the center, as our code now largely conforms to today's stricter C++ standards.\(^542,543\)

As software maintenance time has historically consumed a significant portion of the Center's resources, we have made additional improvements to reduce this overhead. For example, we have automated our software process so installable binaries are now automatically generated and posted on the website. This automated process not only reduces the amount of time spent for each release, it also ensures that, whenever a new feature is added or when a bug is fixed, new versions of the software are posted and almost immediately accessible. To ensure that our software is stable and reliable, we have also added a regression testing suite. We currently have eight machines of different architectures that test the software daily and report stability metrics on a public dashboard. To minimize our maintenance overhead for this service, we have host our software statistics on a publicly available dashboard, hosted by Kitware (my.cdash.org). To facilitate the transition to support multiple platforms and daily automated tests, we collect our GNU-make based build system with the cross-platform CMake system (www.cmake.org). CMake allows us to use the same build scripts on all platform, significantly reducing the maintenance overhead for the software development team.

13.3.2 Application framework

Since our last renewal, we made a variety of improvements to the SCIRun system. SCIRun now includes extensions to Matlab for even faster prototyping algorithms. The latest installment of the SCIRun software, version 4.2, boasts a more user-friendly graphical interface and a substantial increase in the datatype flexibility of the internal module algorithms. In addition, the same simulation dataflow network can now be used for meshes based on hexahedral elements as well as tetrahedral elements, allowing for a quickly evaluation different patient specific modeling options.

For our new segmentation and volume rendering applications, called Seg3D and ImageVis3D respectively, we decided that the dataflow paradigm, as used in SCIRun, would not provide an optimally interactive experience for users and was not easy to maintain for developers. Therefore, these new applications were developed with a specific user interface for the targeted audience. A different approach was used for the meshing application, called BioMesh3D. BioMesh3D consists of a series of scripts that tap into the SCIRun core infrastructure and build a mesh using command-line tools.

While distinct from a software engineering perspective, all of these approaches have their own merits. The fast prototyping nature of SCIRun allowed us to design prototypes for DBP: Simulation of cardiac defibrillation (19) and DBP: Simulation of electric stimulation for bone growth (18) within a short time span and the scriptable nature of the meshing pipeline allowed us to easily integrate with the software in the Dr. Warfield's laboratory (DBP: Bayesian Source Imaging of Pediatric Epilepsy, 21). However, we discovered that the SCIRun prototypes were not well suited for processing large amounts of data; similarly, the lack of scripting abilities in Seg3D slowed the segmentations in these projects. To address these shortcomings, we propose to continue developing technology that will allow us to quickly provide different interfaces for the different tools, as each phase of patient specific modeling has its own challenges, each requiring a different interface.

To further improve our software development process, we undertook a substantial reorganization of SCIRun's architecture (which is also the base for the Seg3D and BioMesh3D applications). The main thrust of this reorganization was to separate the algorithms of our system into modules that could be called as separate programs from the command line. It is this modular design that is now used by the BioMesh3D prototype.\(^66,170\) This prototype reuses many of the SCIRun algorithms, but exposes them as standalone programs that can be run from the command line, making it fully scriptable and automated. A further modularization is underway to allow future...
tools to compile only the components that they specifically require, thus dramatically simplifying (and shrinking) the size of each software application that we distribute.

### 13.3.3 Core functionality

In software engineering, the quick pace at which a program's complexity and its maintenance requirements grow are the biggest challenges. To minimize the complexity of our software, we have undertaken to further modularize the pieces. For example, we completely removed a system (called “dynamic compilation”) that compiled and linked source code at run-time in order to obtain nearly optimal performance. Although the technology was interesting from a computer science perspective, it generated too many instabilities in the software and was a huge maintenance/testing burden. We have since employed more conventional software development methods. Stripping out this and similar subsystems has rendered an infrastructure core that is far more modular, more general, and easier to maintain.

Over the past year, we began using this simplified system to generate a first generation of applications that use the new modular core. The first example is the BioMesh3D scriptable meshing pipeline. By combining pieces that were developed separately, such as a particle system that optimizes tessellation on a surface with the modular pieces from the SCIRun framework, we were able to rapidly construct the first implementation of an automated tetrahedral meshing pipeline.

As part of this move toward modular, maintainable components, we developed the ImageVis3D application as a collection of light-weight subsystems, which are easily integrated into other applications. The ImageVis3D infrastructure contains several new pieces of software technology that we intend to leverage throughout our software: (1) its out-of-core volume renderer (called Tuvok) was generated to use recent advancements in volume rendering, such as GPU-based ray-casting and out-of-core methods for rendering very large datasets; (2) it has a new, improved DICOM reader that reads nearly all DICOM formatted image data (each imaging system manufacturer extends the DICOM standard, thus making it difficult to support a fully-general reader).

Beyond our local software development efforts, our Center has been collaborating with Dr. Cooperman at North Eastern University to generate a framework that will allow users to run our software in a virtual VNC shell. This development will provide many exciting new possibilities for stopping and restarting the software at any point during execution, as well as remotely connecting to the software from any platform that supports VNC. Although the current prototype is limited to applications not heavily reliant on OpenGL, we expect that future versions of this technology will provide a fast means for remote access and control of our our Center's applications.

### 13.4 Design and methods

#### 13.4.1 Aim 1: Transform new algorithms developed in the technical cores into stable, available, documented software projects.

When transforming our TRD software into distributable applications that we can support over the long-term, we mainly use a process of software “hardening”. We propose to harden the code in several ways: (1) upgrading the source code to make use of modern C++ functionality such as Boost (www.boost.org) and STL, (2) minimizing third-party dependencies wherever possible, (3) removing platform-specific code, (4) cleaning up the API, and (5) providing documentation in the source code for developers. The next stage in the process of distributable software is ensuring that software tests are available and executed in the regression testing suite, which guarantees that updates to our code or to other code on which we depend (e.g. operating system upgrades, new graphics card drivers, updated third-party libraries, etc.) do not break existing functionality. We propose to keep in place the current CTest-based system (www.cmake.org) for regression testing and to further leverage the CMake tools by using CPack as well. CPack is a light-weight, cross-platform packaging system for building installable binaries (www.cmake.org). The primary remaining component that requires on-going attention is the generation of new tests to exercise and validate any new functionality.

In the process of releasing software, the final piece is the generation of documentation and tutorials. Our past experience with different documentation systems showed that the most maintainable system is embedding the tutorials and documentation within the source code and embedding the documentation building process within the software building process. Not only does this allow us to use one system to keep track of source and documentation, it also means that documentation stays synced with the software development. We have found LateX, specifically pdfLateX, the easiest documentation system to maintain while still producing professional
output. The CMake system ships with macros for compiling PDF documents out of LateX files, thereby providing a simple solution for packaging professional looking documentation with each application. We propose to continue using this process and move our remaining external documentation into our source code repository. By bundling the documentation with the source code as an open-source project, we also open up the possibility for community-contributed documentation, which is under revision control and which can be monitored in tandem with the source code.

13.4.2 Aim 2: Improve usability of tools for integration in the image-to-simulation pipeline.

In addition to the development of new technologies, as highlighted in the TRD cores, an immediate need exists to integrate new features into our tools. Such integration would further streamline the image-to-simulation process enabling these new technologies to be tested within clinically relevant applications. The infrastructure core will provide solutions for gaps in the pipeline where no solution is readily available or where new features in the tools allow for a much smoother overall integration.

The first set of improvements relate to file I/O: (1) we propose to include file readers and writers by bridging the main framework to third-party software, such as those supplied by the Insight ToolKit (ITK) (www.itk.org) and Visualization ToolKit (VTK) (www.vtk.org); (2) we will integrate our recently developed DICOM reader in all our software applications; (3) we will expand file I/O by adding a framework that allows for the selection of formatted data blocks from either ASCII, Matlab, XML, binary, or HDF5 formatted files and that can be programmed to look for similar blocks in files of the same format. The latter construction is mainly intended for importing data for simulations, as there is no commonly accepted standard for simulation data.

A second set of improvements aims to improve the interface of our rapid-prototyping tool, SCIRun, through “tabs”. By adding a top-level set of dynamic tabs, such as those found in modern web browsers, the various steps within the image-to-simulation pipeline can be broken into smaller, distinct dataflow networks, each controlling a part of the pipeline. All these distinct dataflow networks will be linked by a central database within the application, to ensure the capture and coordination of intermediate state. This structure will enable a more intuitive workflow for prototyping and for algorithm exploration in our image-to-simulation pipeline.

A third set of enhancements will be tighter integration with the Insight ToolKit into our core framework, in order to expose a wider set of image-processing filters for SCIRun, Seg3D, and BioMesh3D.

Finally, we propose to insert additional segmentation tools into Seg3D, such as copying segmented slices from one location to a neighboring location in the image stack and interpolating between segmented slices.

The above list will serve as a starting point for improving the usability of the image-to-simulation pipeline. As additional TRD-developed algorithms become available, we will continuously assess whether additional improvements are needed to ensure that the newly developed technologies can be exercised in clinically relevant settings.

13.4.3 Aim 3: Render a common software base for (1) prototyping tools, (2) stand-alone dedicated applications, and (3) scriptable tools.

In order to generate software for a wide variety of applications, we propose to take the various pieces of software the Center currently distributes (i.e. SCIRun, ImageVis3D, Seg3D, ShapeWorks, and BioMesh3D) and integrate them into one common, but very modular, framework; we will then use this framework to drive most of the research for collaborations. Although the list of desired software ranges from rich, graphically-driven end-user applications to scriptable tools, they only differ in the way the user interacts with the underlying software, not in the way that the algorithms actually function. For example, whether a mesh is generated inside an application with an interactive GUI or whether it is scripted from a command-line, the algorithm will generate the same output. Depending on the specifics of the setting in which the tools is to be used, a collaborator with many datasets may find it important to be able to script the tool, whereas another collaborator may feel more comfortable with a traditionally styled application. Hence, the architecture should support the flexibility to quickly generate applications that fit the workflows of specific users or specific fields.

The field of patient specific modeling and, likewise, the Center are entering a realm where many datasets need to be processed in order to validate research goals, requiring some software redesign. A key aspect of the redesign is to making the software aware of state and able to record the provenance of each dataset. This must be accomplished in such a way that the resulting data can be fully recreated from the provenance record. In order to avoid duplication of this information within the program, the information needs to be recorded
at the application’s core level. To meet this need, we propose to design both state and provenance into their objects and to expose them in such a way that state and provenance can tightly interact with all of the other components in the system. Having a state and provenance manager in the core of the software provides the additional advantage of forcing a design where GUI interaction is tunneled to and from the core. With a clear separation of the GUI and the application engine, adding a layer to tunnel events between components becomes much easier to implement.

A common design strategy for recording provenance is the use of the so-called “command pattern”. The command pattern is a design pattern prescribing that every action in the program needs to be recorded and pushed onto a global stack traversable to either undo or redo an action. Although recreating the state of a program from an action list is valid (this functionality is, for instance, needed to load and save a session), with complex and time-consuming algorithms such as image processing filters or linear solvers, this can be an expensive solution. Hence, we anticipate recording the active state of a program as well. In this case, state consists of all the information currently present in the program. A common method for recording state is serializing all the parameter and data objects and writing them to a file; the state of the program can then be recreated by loading the serialized data from disk. In order to generate this functionality, we will rewrite our “Persistent” serialization subsystem of the SCIRun core. The current system only allows for serializing the data objects in the program such as meshes, image data, or matrices, but does not have support for light-weight (“session”) parameters, such as those that appear on the module user interfaces or the dataflow network graph itself. Hence, we propose to expand SCIRun’s current Persistent system to also include these session state values.

In addition to our state/provenance development plans and our enhanced support for different file types, another major goal for our infrastructure is to support the GPGPU algorithms that will be developed by the TRDs. We are proposing to alter our general infrastructure core by adding an OpenCL core, a provenance/state core, and a new File I/O core. A schematic subdivision of the proposed SCIRun core and its application layers is depicted in figure 13.1. In addition to the new functionality in the core, we will also add an intermediate layer to reside between the application and the core functionality. This layer, which we refer to as the Application Control Layer, will wrap the underlying algorithms into modules that expose specific functionality. However, a module does not necessarily expose just one algorithm; it may actually control a variety of algorithms, for instance solvers that are located on the GPU, as well as versions of the same algorithm that use the CPU if no suitable GPU version is available. The Application Control Layer will also combine actions into a series of steps that are
easier to control from an action recording paradigm and that separate the action definition from the underlying algorithm, making it easier to maintain and debug the program. In the current architecture, both SCIRun and Seg3D combine the module and GUI concepts; however, as we strive to generate a variety of applications on top of the same framework, we will split this functionality into its own Application Control layer. We will move the global Application logic and the GUI components into a third layer that sits atop of the other two layers, as depicted in figure 13.1.

![Figure 13.1: SCIRun and Seg3D integration](image)

The layered architecture will allow us to exchange the top layer of the framework with different Application GUIs and logic; these alternative top layers could then make use of different underlying components. We envision that new applications will share a common framework, as depicted in figure 13.2. One advantage of the layered framework, which we are already exploiting for the BioMesh3D application, is the ability to quickly generate command-line utilities that expose the underlying functionality. We propose to expand this functionality to include not only command line tools, but to also integrate this command-line ability with scripting languages such as Python and Matlab. This integration will allow users to mix and match pieces of our image-to-simulation pipeline with other tools that they are already using, such as the Matlab-based COMSOL system for modeling (www.comsol.com).

13.4.4 Aim 4: Add provenance, scripting, and database support into the common infrastructure tools.

Once we have developed a common application framework with state and provenance and all of the tools have been updated to use the new framework, our next step will be to expose the provenance and playback functionality inside those applications to facilitate easy replay and parameter space exploration. Within the Scientific Computing and Imaging Institute, the VisTrails application has been developed to provide precisely this functionality. We propose to design our state and provenance recording in such a way that we can leverage VisTrails for controlling and editing provenance playback within all of our Center's software applications.

Similarly, the command structure that allows one to play back a series of events for provenance exploration can be turned into a simple command language for each application, enabling the execution of a series of predefined commands. In practice, though, while many parts of a workflow can be automated, there are a few that may require manual user manual feedback. For instance, the process of segmenting a dataset often requires running a couple of automated filters in combination with manual intervention to clean-up specific structures. In order to streamline this process for large collections of datasets, we will reuse the provenance
command play-back options to provide a simple scripting language for the user to run automated or semi-automated processes. In the case of Seg3D, we will add a scripting language that runs several filters, but is also able to relinquish control back to the user along with instructions of what manual process needs to be completed before the script can continue.

Finally, we propose to encode the provenance information inside each of the resulting output data files, such as the finite element mesh or the segmentation label-map. The provenance will be extractable from the files, such that we can store it in a scientific database. As with the provenance information for the entire workflow session, this provenance data will serve as a script that can be replayed to regenerate the data. (This provenance information will likely be a very small file relative to the output data file that it produces when played; in this sense, the provenance “key” is a lossless, very highly compressed representation of that output data file.) For example, a segmentation process in Seg3D may result in different structures that are stored and processed separately afterwards; each of the results will get the provenance information specific to that segmentation encoded in the file. Hence, we are proposing to develop tools and file formats that will facilitate the storage and extraction of all relevant provenance and meta-information.

13.4.5 Aim 5: Add support for large data-sets in common infrastructure tools.

Not only does the patient-specific modeling paradigm generate more datasets than ever before, but new imaging technologies are also producing datasets that are larger than ever before. With the resolution improvements gained in the various imaging modalities, these datasets pose a serious challenge for the image-to-model pipeline; datasets at the high end of the spectrum are still larger than a modern PC can handle at once. To keep pace with the ever growing demand for resolution, we propose to integrate the technology we developed for large-data support in ImageVis3D into the other applications as well. The technology that allows ImageVis3D to use datasets that are far larger than what can fit into memory is a combination of bricking and multi-resolution levels of detail. These strategies for organizing data become especially powerful when used in conjunction with algorithms that are tuned for data streaming. Specifically, wherever possible, the algorithms must be optimized to access just a single brick of data at a time and to propagate results from lower levels of resolution up to higher ones. To most efficiently operate with streaming multi-resolution data, we will optimize the key algorithms within the SCIRun and Seg3D pipelines on an as-needed basis. As a first step, we will integrate UVF format (unified volume format) into SCIRun and Seg3D and add a new interface layer allowing the applications to read and write bricks and process them on-the-fly.

Besides being able to handle large imaging data, we also propose to extend our infrastructure to include support for processing collections of datasets in the SCIRun framework. Examples of such dataset collections include temporal data, statistical atlases, and extracting statistics out of many models. To improve support for handling collections of datasets, we will expand our array concept of objects, where multiple objects can be grouped together and an operation can be either applied to every element in the dataset or common features can be extracted from all sets.

13.4.6 Aim 6: Provide infrastructure for streaming / GPU architectures.

With General Purpose GPU (GPGPU) computing becoming a viable option for improving the speed of many algorithms, we need infrastructure that ensures that the algorithms developed in the TRDs can be distributed across a gamut of platforms, including graphics cards. While the development of standards like OpenCL (www.khronos.org) simplifies the task of uploading programs and data and managing the GPU for general purpose computing, additional infrastructure is needed to ensure that the appropriate drivers are located and loaded in the program. One of the key aspects the infrastructure core will provide is the ability to test these new algorithms on a variety of graphics cards and report success/failure for each configuration, as well as performance data. By using our current regression testing system and expanding it with even more variations in commonly used graphics cards, we will be able to track and modify the performance data as we prepare these GPGPU algorithms for use in our collaborative projects.

OpenCL drivers have not yet been made widely available on our customer platforms; we will have to generate additional documentation to instruct our users on where to obtain the latest drivers and which versions have been verified as acceptable. Only with proper documentation and testing will the transition to more GPU driven computation be smooth.

Besides the availability of drivers and libraries to dynamically bind the required symbols to support upcoming standards like OpenCL, there is also a wide variety of functionality across graphics cards. That is, not every
card is suitable for GPGPU computation. We must make crucial pieces of software available for the CPU as well, even through performance will almost certainly suffer.

The shift towards GPU-driven computation means that traditional ways of dividing up computation may no longer suffice. Our current CPU-based algorithms for finite elements on regular grids require the creation of a stiffness matrix and then a call to the solver. In our case, the solver is meant to solve any system, independent of the structure of the matrix. Although an implicit representation of the stiffness matrix for a structured grid would be advantageous for a CPU program, it is even more crucial for a GPU-based program; this is due to the streaming architecture that was optimized for a high computational density on a small memory imprint, where memory access latencies between different parallel threads can greatly inhibit performance. To support this new type of computing, programs like SCIRun need new representations for abstracting these computations. Depending on the algorithms developed for solving finite element systems on the GPU, the definition of the finite element system should not necessarily result in a sparse matrix system. Rather the pieces of the model should be cached until the model is ready to solve for the construction of an implicitly defined linear system on the GPU with the proper boundary conditions. This paradigm shift implies that the current infrastructure needs to be altered to accommodate this shift in computational strategies; these modifications to the system will be generated as part of the creation of the new framework.

13.4.7 Milestones and timeline

As it is important to continue to support our current collaborators and users of our software, the development of the new framework, as outlined in this section, will be done in parallel with the maintenance of the current system. We envision a gradual transition where new core functionality is slowly phased-in and old pieces are slowly phased-out, all the while maintaining the stability of the system.

The first Infrastructure task we will engage in the first year will be the integration of the remaining BioMesh3D components into the SCIRun architecture. Next, we will integrate a more flexible I/O mechanism into the core that supports SCIRun, BioMesh3D, ImageVis3D and Seg3D. Also in year one, we will start the separation of the GUI layer out of SCIRun and complete a prototype separation, wherein each module has an interface that is independent from the graphical user interface. By the end of year two, we will have completed the implementation and testing of the new state manager and provenance manager with Seg3D and SCIRun. By the end the same year, we will have completed the integration of the UVF large data format. In years three and four, we will complete the new architecture and enable all the tools with provenance, undo/redo stacks, and a separable GUI layer. While we are developing all of these new features, we will be adding new functionality to SCIRun, BioMesh3D, ImageVis3D, and Seg3D to support the collaborations, as well as integrating the new functionality that comes from the TRDs.

13.5 Impact and significance

One of the main goals of the Center is to provide software support in our area of expertise. It is paramount to generate software that is stable, tested, documented, and well-designed for its intended purpose. Although the algorithms developed in the TRD cores are the driving force behind the innovation in the software applications, we would lack substantial impact on the community without a software distribution process to harden the code. Thus, we strongly believe that our Center Infrastructure development projects will remain a key component in driving the success of the Center.