The SeisRes Operating Framework

1. Introduction

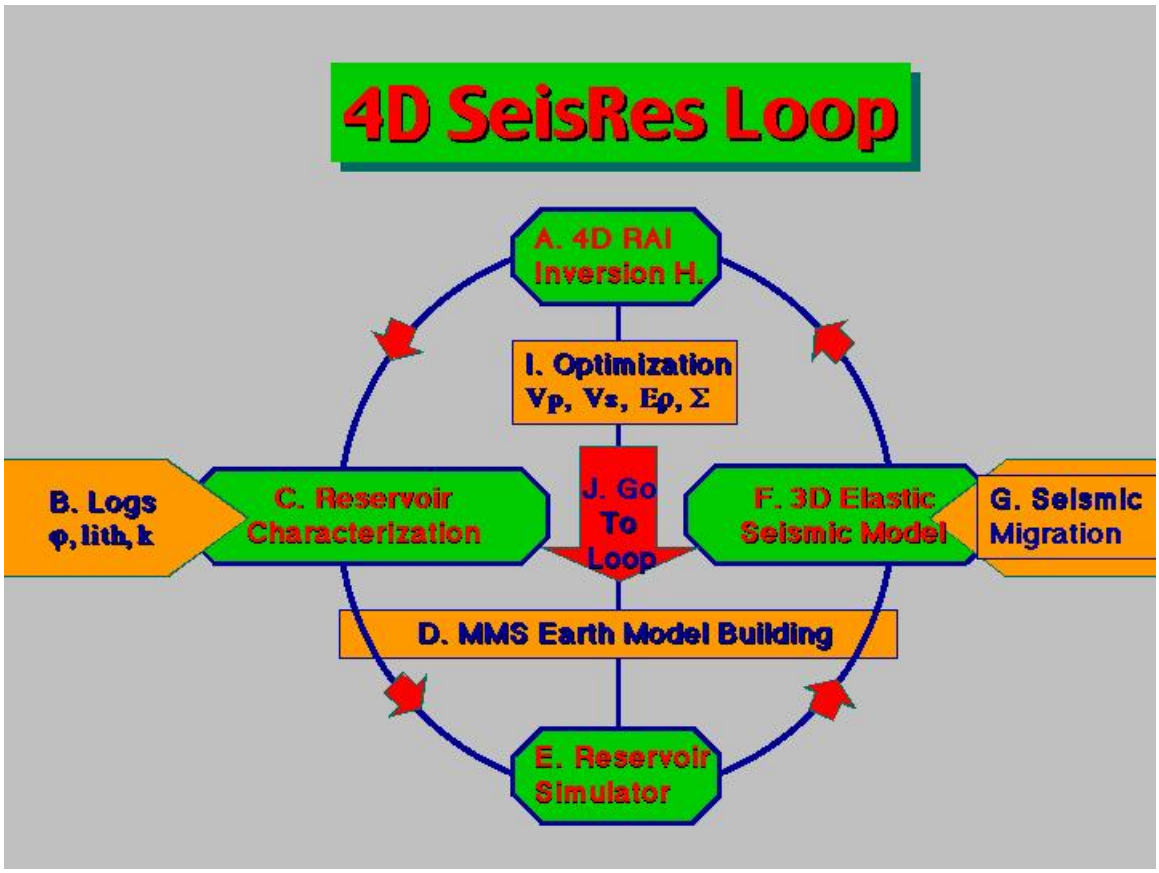
The tracking of fluid drainage over time is a required condition for efficient reservoir monitoring. 4D time-lapse seismic differencing holds great promise as

the keystone to an integrated reservoir management strategy that is able to image changes not only within a reservoir but within the stack of reservoirs that make up most of the oil and gas fields of the world. Yet there are major components of 4D sesimic monitoring that are only just being developed by the industry. Field acquisition is still centered on reacquisition using 3D methodologies; processing and interpretation are focused on normalization and differencing of time lapse data itself; and seismic modeling is 1D and 2D, acoustic, and built around one reservoir at a time.

Three years ago, the need to integrate the observed 4Dseismic differences with reservoir simulator predictions and production histories of fluid withdrawal in fields became apparent. If quantitative reservoir management were to truly come about, the engineering, geological, and geophysical worlds would have to be combined into what is now being called an "earth model" solution. That is, the observed seismic changes over time must be reconciled with the known fluid withdrawal information and the complexity of the permeability pathway information if successful predictions of future drainage were to be optimized.

What was missing was the computational Operating Framework (OF) that would allow for the seamless and rapid communication between and among the varied software applications that is required for reservoir management. We launched a very ambitious software development project code-named "SeisRes". Not only would the reservoir stack be simulated from a fluid flow perspective, but the drainage changes would be fed into a 3D, elastic seismic modeling program that could simulate seismic amplitude changes accurately enough to be realistically compared to real 4D seismic field data differences. An optimizer would then reconcile the differences between the differences. That is, the time-lapse differences between observed and computed seismic and fluid flow models and data would converge to the best view of the real changes occurring in oil and gas fields that the industry has so far been able to produce.

The SeisRes "Loop" consists of the following workflow:



A. 4D RAI workflow for non-linear inversion of two 3D seismic volumes acquired at different times during the production history of a field, and their time-depth conversion, normalization and differencing;
B. Well log preparation and depth-time conversion using SigmaView;
C. 4D Reservoir Characterization of the two seismic volumes using geostatistical co-kriging in EarthGM;
D. Export to the MMS MultiMesher for Earth Model building;
E. Fluid flow simulation in either Eclipse or VIP;
F. A 3D elastic seismic modeling phase to generate 4D synthetic seismic cubes;
G. Export of the modeled seismic to Omega for migration;
H. Differencing of 4D model- versus observed-seismic data and analysis of the difference-of-the-differences;
I. Optimization that identifies changes in physical properties of reservoirs to more closely match fluid withdrawal, pressure changes and seismic differences; and
J. A "Go To" loop back to E.

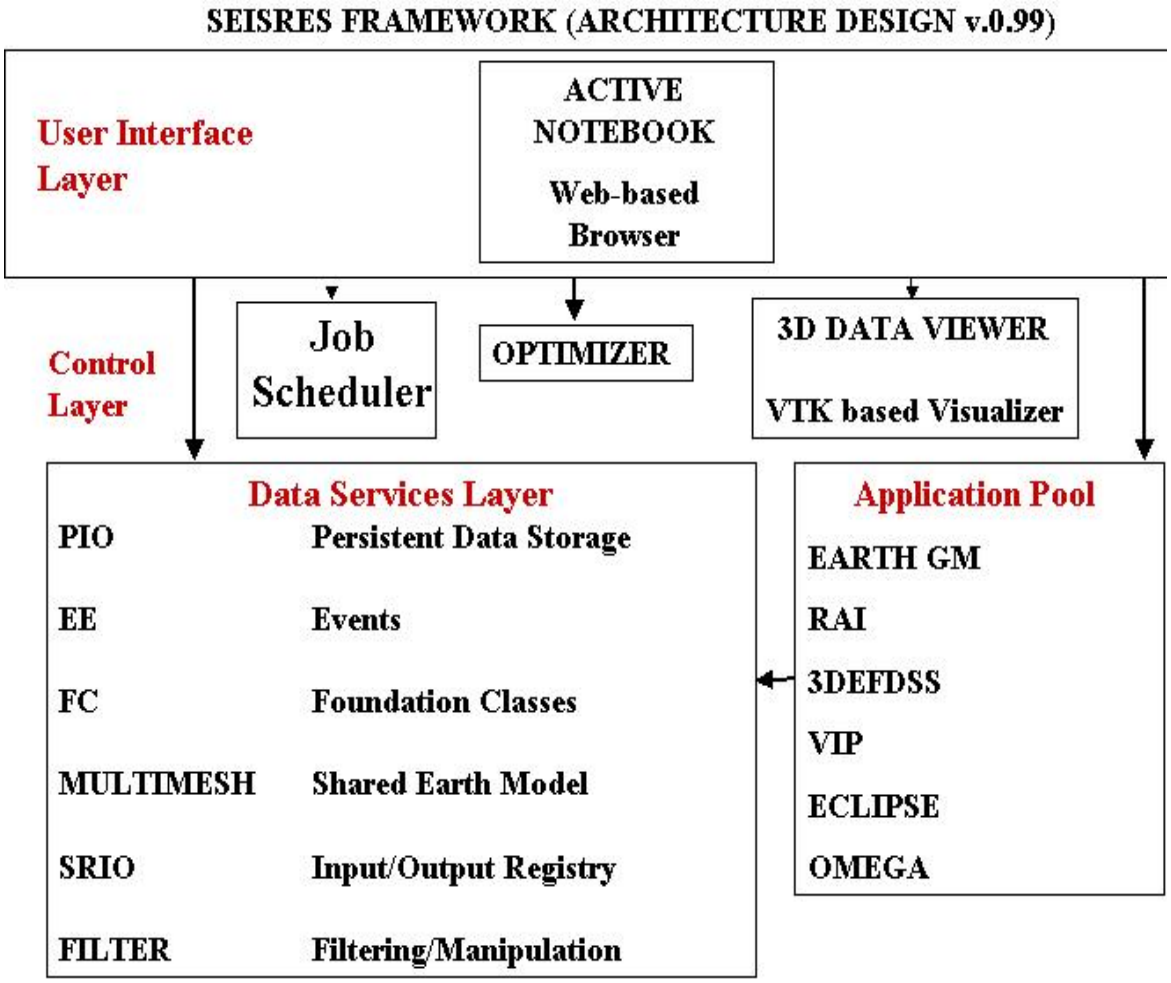
The architecture and design of the interaction of the seismic modeling code with statistical reservoir characterization (EarthGM), observed seismic differences (RAI), reservoir simulators (VIP and Eclipse), and inversion and migration codes (Omega) is indeed a sophisticated computational task. In the process of

development of SeisRes, the Web grew around us so that we were able to build the industry's first extensible, operating framework that enables and versions data and interpretation workflow among the various Vendor applications need to complete the loop from geological and geophysical interpretation to engineering implementation required by the modern oil field asset team.

2. SeisRes OF Architecture

The SeisRes design and implementation efforts to date have produced more than 500,000 lines of state-of-the-art C++ SeisRes system code, and another 500,000 lines of scripts, wrappers, and implementation scripts.

The architecture of the SeisRes Operating Framework contains the following major components:



1. We control and track all activity within the OF using our web-based **Active Notebook**. A hind-cast of previous runs can be quickly and easily reviewed because we version all input and output to each application using the Active

Notebook. All transactions are timed, including computer cycles used throughout the WAN, so billing is quick and easy.

2. We provide a vendor-neutral data model with our object persistent input/output, **PIO Data Repository** for our OF, that sits on top of the users various existing data management systems. We implemented our PIO Data Repository using fast and reliable TCP/IP , distributed objects, and low-level protocols like XDR. CORBA (or next generation systems like SOAP) still do not support native persistency, but we can manage PIO with them.

3. We provide access to vendor applications, not just vendor databases, using automated **Wrappers**. For example, we allow the user to choose between Eclipse (from Schlumberger) and VIP (from Halliburton) within our OF superstructure, just as you click on different programs on the Microsoft Windows OF on your PC desktop. We take care of the connectivity, meshing, data traffic, and versioning. We use “Swig” to create interfaces for scripting Wrappers automatically.

4. We have an **Event Handling Mechanism** to make applications run asynchronously. We parse the workflow among several applications simultaneously and distribute them on the client/server network, then reassemble them as the complete rather than having to wait for one to finish before beginning another.

5. We provide a rich set of reusable, extensible **Container Foundation Classes** for engineering, geological and geophysical data so that new applications and data types can be added to the OF management system easily and quickly.

6. We provide **MultiMesh**, a tried-and-true automatic meshing system from IBM that is topological so that whatever the gridding requirements of an Application are, we can quickly deliver it.

7. We provide an **Optimization Tool Kit** so that key parameters can be modified to converge on a least-error solution during the running of an application. The optimizer can be visualized over the web and changes to parameter settings for one application are propagated to others that use those same settings.

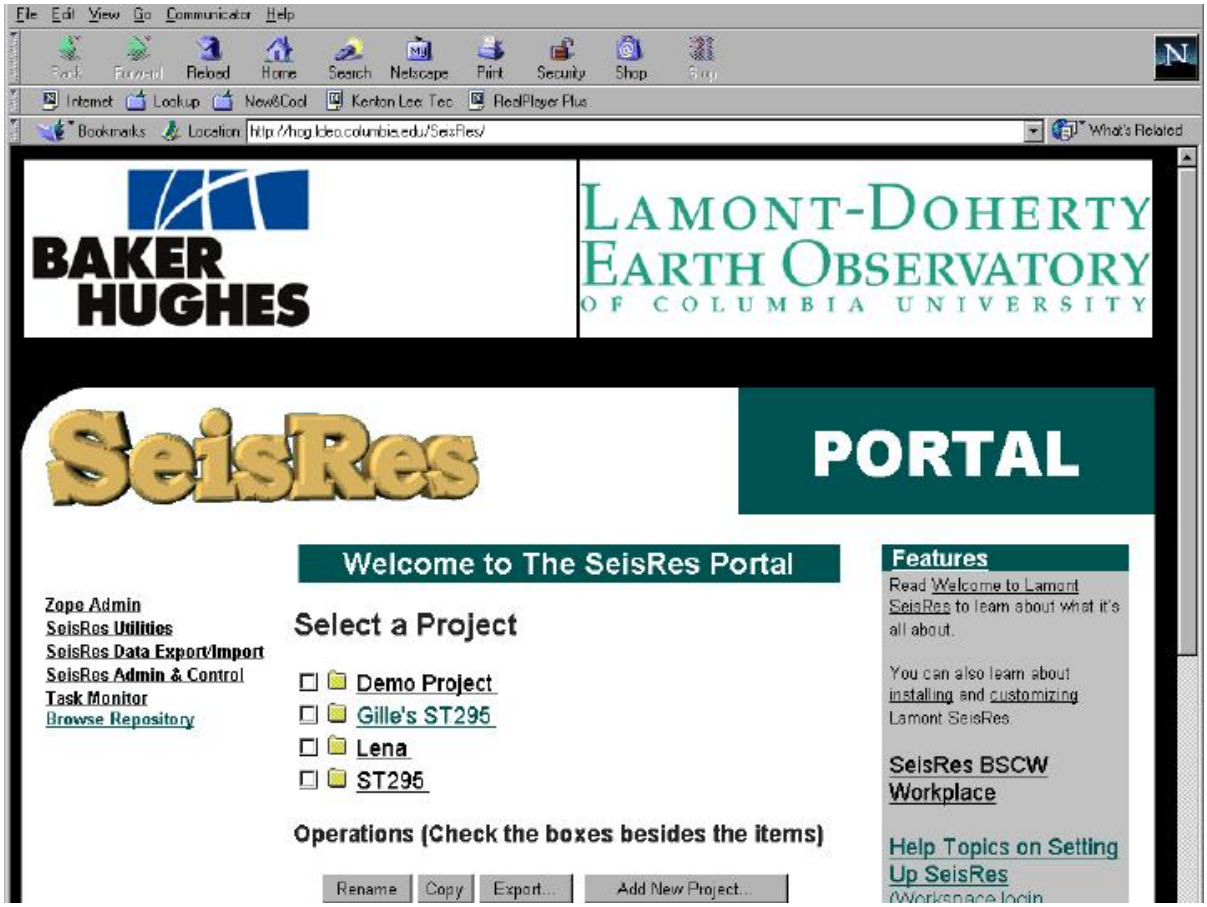
8. We provide the **SeisRes Data Viewer**, a Web-based visualization system developed for us by VTK, a GE spinout company. The user can design the visualization of progress of his computer simulations over the Web and manipulate the images in real time as they are computing.

3. SeisRes System Components

3.1. Active Notebook

Best-practices in the execution of science and engineering computations is to keep a notebook with a record of the experiment and its trail and errors (the modern version of the researcher's notebook). In doing a complicated task like SeisRes, a notebook will necessarily have structure to it -- broken down into sections describing the multiple tasks involved in the workflow and sub-investigations done along the way to complete the overall task. For SeisRes, we implemented modern web technology to do a computer-based notebook. The notebook will capture the tasks (and all their attributes) of SeisRes as they are being performed by the user. It is an active notebook because the initiation and monitoring of SeisRes tasks are done using this web technology. Past work in SeisRes can be reactivated and investigations renewed using the notebook.. The user interface to the broad set of SeisRes tasks is the notebook.

The Active Notebook is a website that monitors and records SeisRes tasks as they are being performed by the user. Past work in SeisRes can be reactivated and investigations can be renewed using the Notebook. Thus, the industry has the capability of hindcasting computational tasks related to reservoir management using the Active Notebook. All keystrokes and data that went into a set of interpretations among the applications controlled by the OF can be



recomputed, even if years have passed in the meantime (assuming the archival data has not been destroyed, of course). Decisions that might have resulted in a giant discovery can be investigated for learning and best-practives improvement

purposes, and the workflow disiminated throughout the company, for example. With the ability to author new web content linked closely to the SeisRes computation that generates it, the Active Notebook has the ability to author content and carry out SeisRes tasks using a document generation metaphor. The Notebook serves as the main user interface to a broad set of tasks, broad because it is infeasible to track detailed internal user activities while using interactive programs such as EarthGM. All SeisRes tasks and the SeisRes SDV can be initiated and are captured in the Notebook

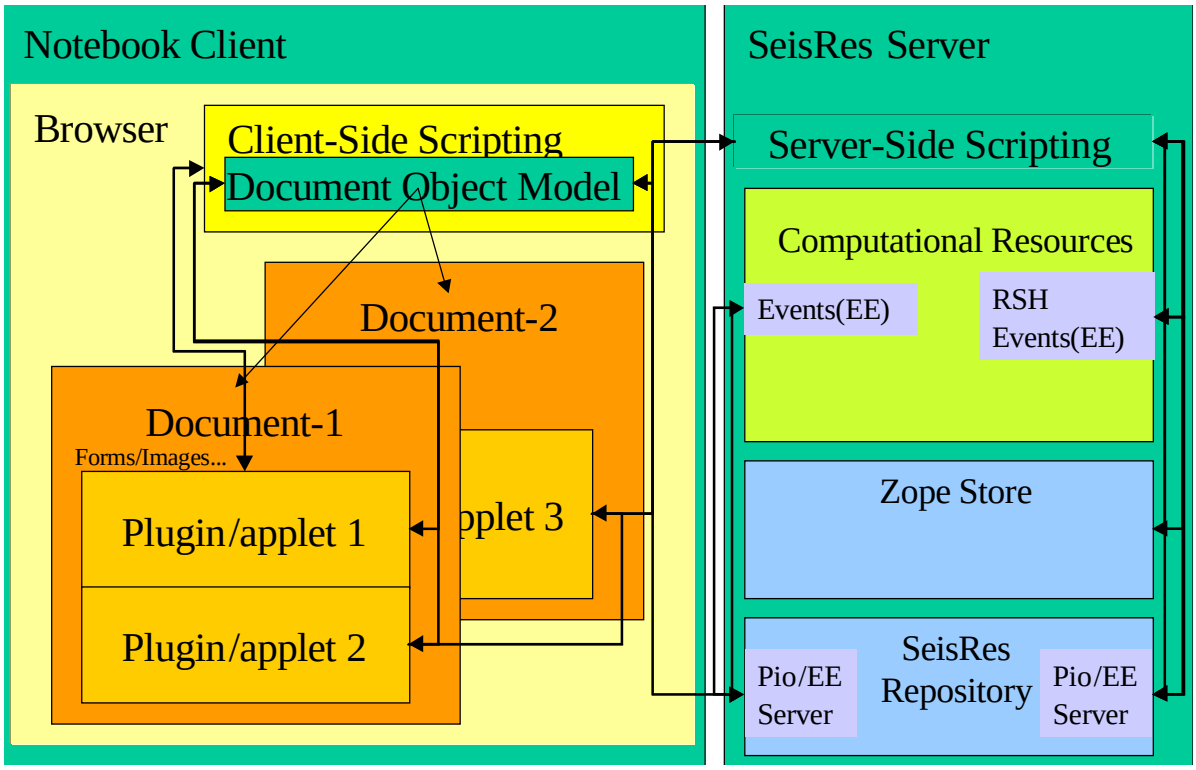
The architecture provides a mechanism for both client-side and server-side scripting that dynamically generates user interfaces to SeisRes tasks within the notebook. These interfaces are archived within the notebook as work steps of SeisRes progress. A thin client supports visualization and is embedded in a standard Netscape web browser.

The spirit of design is to be able to dynamically create from SeisRes metadata in persistent store, new interfaces to SeisRes workflow tasks. Scripting on both client and server are coordinated to achieve this. We prefer scripting languages to a language that is compiled or even byte compiled (i.e. Java) for this task. We are using tcl and the tclet plugin presently. While a SeisRes notebook document may include Java applets, these are placed (authored) on the web document using scripting. Note, however, that there are scripting language interpreters that work within Java (some examples: FESI, a free EcmaScript (Javascript-like standard effort supported by Netscape) Interpreter in Java, and JPYTHON).

3.1.1. Notebook Client

The client supports computation orchestrated though the web browser. The Active Notebook is based on using VTK as a tclet plugin. The browser has an interface to SeisRes repository objects directly within the browser's scripting environments. These include, but are not limited to, tcl, Javascript, or JPython. The scripting environment in the browser has access to the browser's document object model (DOM). One such access is Netscape's LiveWire..

The thin client runs in trusted mode when executing the visualization as well as when interfacing to the repository and event objects, since SeisRes will be used in an intranet situation, initially. XML is used to transport structured data between the browser and the SeisRes server. The browser makes use of embedded viewers that parse XML for viewing or parse the XML directly in their scripting. Embedded viewers can be script based (i.e. tclet), Java based, or pre-built as plugins. The client notebook supports an interface to authoring and status. We use the Zope authoring interface for this.



3.1.2. SeisRes Server

The server tracks the workflow progress of a user and dynamically constructs new web content including client-side scripts based on user initiative, SeisRes objects, and metadata about the state of workflow of the on-going SeisRes experiment. Changes in state in the client are tracked with forms submission (http POST), cookies, and direct plugin communication with the server (for example, the ability for a tclet plugin to do http POST). The ultimate store of persistent data is the SeisRes metadata store (in Zope and the SeisRes data repository) -- cookies are used, however, to stage persistent data to this store.

We use Apache as the http server. We run the python-based Zope under Apache as a way of dynamically publishing objects to the web. The idea is to use Zope objects to represent the workflow tasks and documents of SeisRes. These objects represent task submission, monitoring, results, synopsis, and associated setup documents that contain submission forms for parameter files for example. Preparing SeisRes documents will execute SeisRes work tasks on the set of computational resources for SeisRes work. Zope has a persistent object system so these documents are archived with the state of SeisRes work though its work tasks. Zope uses a server side scripting language called DTML.

The main functionality of the server side scripting is to construct client-side scripts and web content that comprises:

1. an interface to SeisRes tasks dispatched from the SeisRes Server to compute hosts using RSH.
2. the generation of a client side script that can generate visualization (VTK) pipeline network or a user interface on the no-so-thin client for example.

3.1.3. Notebook Communications

The communication needs between notebook components and the SeisRes server can be summarized as follows:

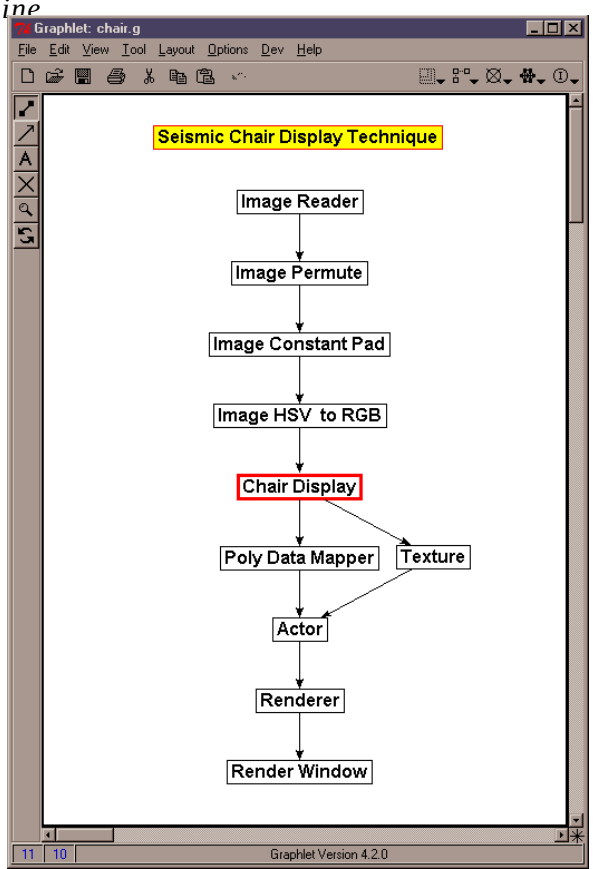
- Applets/plugins see the DOM (Document Object Model)
- Applets/plugins see each other
- Server Side Scripting see the DOM
- Applets/plugins post information to the server
- Client-side Scripting and Applet/Plugins access Events and Repository objects directly using the Event and Repository servers. They communicate with Zope metadata using http POST and indirectly with server side scripting.

3.1.4. Order out of Chaos

Good user interface design should heed issues like organizing the sum of workflows for the various SeisRes experiments, the layout of menus, and user input widgets, plugins, applets, etc. Individuals approach complex tasks in different ways and the overall SeisRes system needs to accommodate user-varying initiatives on work order, etc. as well as display best-practives from workflows used in similar cercumastances elsewhere in the client's company.

Zope tree display used to organize documents

Dataflow Diagram of Chair Display VTK Pipeline



To address these concerns we implemented two features:

1. We introduced a notion of "viewers": plugins, applets, etc. that have a careful but simple interface design that covers the major functions a user would do with the viewer. A viewer could be a seismic viewer with its intuitive interface to visualize seismic data. The parameter widgets and menus with the viewer have a layout that considers the human factors involved in a user visualizing seismic data. The viewer also functions as a control element. We use tclet plugins as the basis for the viewers.

2. We built a web document management and dissemination system in the SeisRes server. Explorer- (as in Windows) like interfaces and/or dataflow graphs are used to give the user a high level view of the notebook's contents with drilldown as well as steps to be executed in the workflow. . Zope is used for the document management system. The two figures above show the use of Zope's DTML tree to organize the notebook contents of a SeisRes investigation and the use of the tcl-based graphlet to show the dataflow for a seismic chair display.

3.2. PIO Data Repository Service

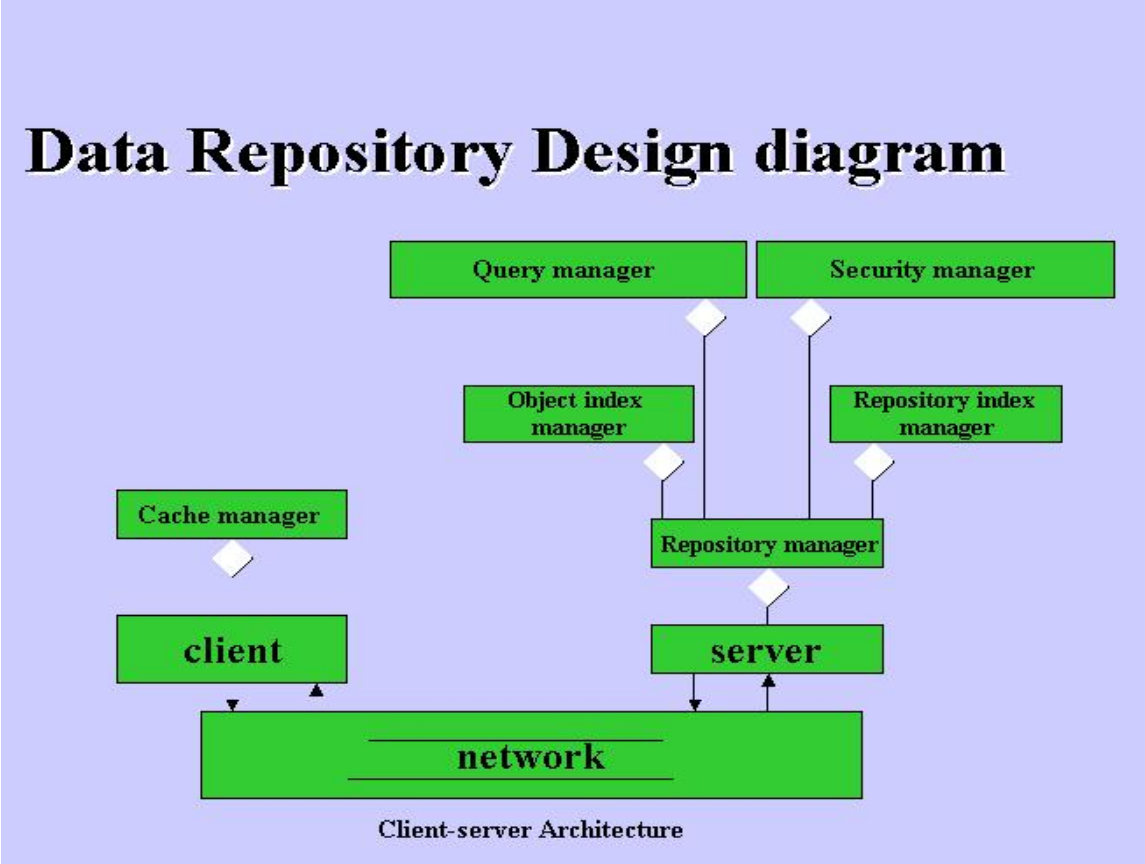
The SeisRes persistent storage package (pio) is an important piece of the SeisRes software because geological, geophysical and other data must be persistent in the SeisRes loop. This persistence requires that data objects can be restored to their original form at any time.

The general idea behind our persistence design for the data repository is to serialize SeisRes objects using XDR and then store the serialized objects in our repository. Our manager-server can be accessed by remote clients through the Web-based Active Notebook controls. This feature allows a distributed workflow to pass object names or references among wrapped software applications.

The SeisRes data object repository functions like an object database which stores and retrieves C++ objects. The storage for the SeisRes data repository is the unix file system. A unix directory is a physical repository. A repository directory has to have two index files: one called the object index file and another called the repository index. In actual implementation of the object repository, the pio object repository manages two another managers: the object index manager the and sub-repository index manager. The object index manager is responsible for adding, removing, renaming, and retrieving object descriptor and ensuring that the index file is consistent and persistent. The repository index manager is responsible for adding, removing, and retrieving a repository index object and making sure the repository index file is consistent and persistent. To guarantee consistency and persistency of the object repository, a centralized server is needed to maintain object index file and repository index file. Other issues such as security and transaction monitoring are handled by the Active Notebook.

The data repository client-server is implemented on top of the system socket layer. TCP/IP protocol is used for communication, that is point-to-point connection is guaranteed for each client. The data repository client-server has two high-level interface classes pioClient and pioServer. The class pioClient is the interface class for all applications and pioServer class is the interface class to pioRepositoryManager, which implements all functionality of the pio server.

Communication between the client and the server is implemented on top of the system socket layer. The communication protocol is TCP/IP. Data transferred is the either fixed-size or variable-size byte stream. On the client side, it creates a socket, binds the socket to the server address, then calls connect to make a point-to-point connection to the server. On the server side, it creates a socket,



Client-server Architecture

binds the socket to the IP address of the host, then listens for the client connection. As a client request comes in, it calls accept to create a temporary socket for that particular client. Data will be received through the socket returned from the accept call.

A client requests a service by sending a message. A request message consists of three parts: the first part is the request code, the second part is the client information which includes user name, machine name, process id, time stamp, and unique client id, and the third part is the parameters related to the request.

The sequence of communication on the client side is:

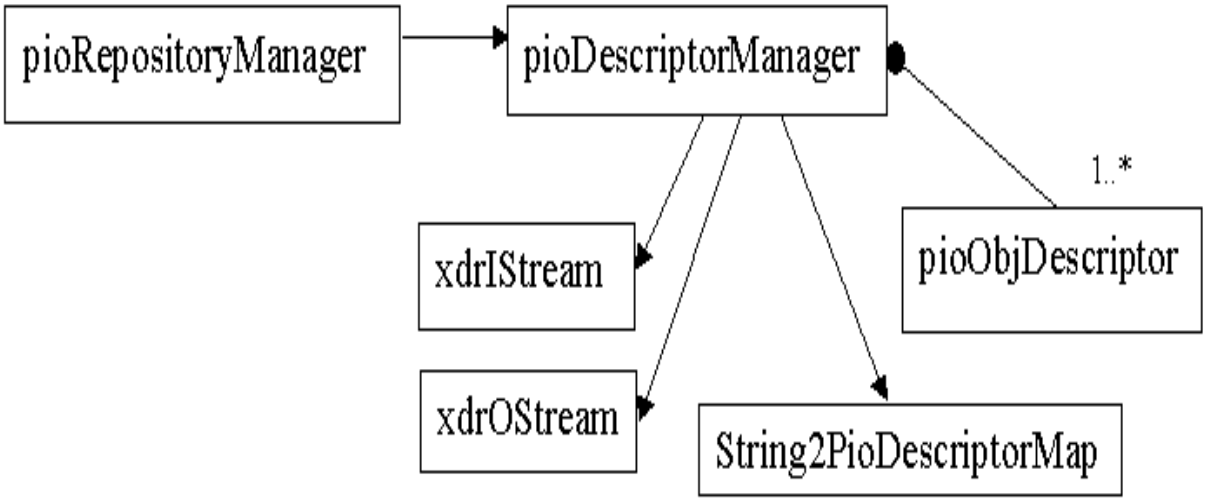
Send a request message;
Receive acknowledge if the requested service will be served by the server;
Send data if necessary (such add object to the repository); and
Receive acknowledge if request fulfilled.

The acknowledge message will be an integer which tells the client if the requested service success or failed. If failed what caused it failed.

The sequence on the server side is:
Receive a request;
Acknowledge the client if requested service available;
Receive data from a client if necessary such as add method;
Send data to the client such as get method; and
Send acknowledge to the client if the requested service fulfilled.

The pioServer uses pioObjRepositoryManager to do all work requested by a client. The pioObjRepositoryManager uses two index managers to manage the directories and files under a unix directory. PioRepositoryIndexManager manages directories, and pioObjIndexManager manages object files.

The SeisRes Data Repository has four high level components: the Repository Manager, the Object Descriptor Manager, the Persistence I/O Handler, and the XDR Streamer. The repository stores objects with the assistance of the Object Descriptor Manager, which maintains a table of indexed objects with their associated descriptions. The Persistence I/O Handler is responsible for the construction and casting of objects. It uses the XDR Streamer to serialize objects to files in XDR format, which are then stored in the repository. Note that the repository is a set of files stored in NFS. This concept is analogous to the repository of a source code versioning system such as CVS or RCS. The difference is that we store machine independent binary files (XDR format) representing serialized versions of objects.



3.2.1. pioRepositoryManager

A project repository is simply a Unix directory. The Repository Manager adds and removes objects into/out of the repository. It uses the Object Descriptor Manager to catalog the objects into the repository. Each project directory has one file, which contains a collection of descriptors (pioObjDescriptor) objects. These object descriptors have information about all objects stored in the repository. The Object Descriptor Manager class (pioObjDescriptorManager) adds, retrieves and deletes any object descriptor from the repository. In this implementation, each object is a single file in ASCII or XDR format. The static function to create an object of a defined type has to be registered in a registration table (String2PtrFuncMap) before the object is stored or retrieved from the repository. A typical session using the repository to add an object is demonstrated in the following code excerpt:

3.2.2. pioObjectDescriptorManager

There is an object descriptor (pioObjDescriptor) associated with each of the objects stored in the repository. The Object Descriptor Manager (pioObjDescriptorManager) contains a singleton map:

typedef map< string, objDescriptor > string2ObjDescriptorMap

That relates the object name to its descriptor. This manager essentially performs operations to add, remove and commit changes to the repository.
The Object Descriptor contains relevant information associated with the objects to be stored such as file format (ASCII or XDR), name, type, project name, id number, owner, time-stamp, object description and xdr version string.

3.2.3. Persistence I/O Handler

This component is the most important in the design. The Persistence I/O Handler is responsible for registration, construction, initialization, and proper casting of the stored objects. The Handler is a placeholder for the types that we want to store in the repository. The Handler uses XDR to serialize and write the objects into the repository. Reading the serialized objects from the repository is more complicated, because the Repository Manager does not know the type of the object it is going to read. The Repository Manager only has a string containing the object name. Therefore, it utilizes the string-to-pointer function map to locate the proper method to construct the object and return it as a pioObjBase pointer. This pointer is then cast (narrowed) by the user using the objCast method that is essentially a dynamic cast checking in addition to an object registry into the repository. It is not possible to retrieve unknown object types from the repository and if the user tries to retrieve an unregistered type then an invalid pointer (nil) is return. If the object is not registered at all an exception is thrown.

3.2.4. XDR Streamer

The xdrStream class wraps the XDR serialization functions for the fundamental built-in types in C++. The XDR was created by Sun Microsystems, Inc and is freely available. It is normally built in the libc of Unix systems for remote procedure calls. XDR provides a conventional way for converting between built-in data types and an external bit-string representation. These XDR routines are used to help implement a type encode/decode routine for each user-defined type. The XDR handle contains an operation field which indicates which of the operations (ENCODE, DECODE or FREE) is to be performed.

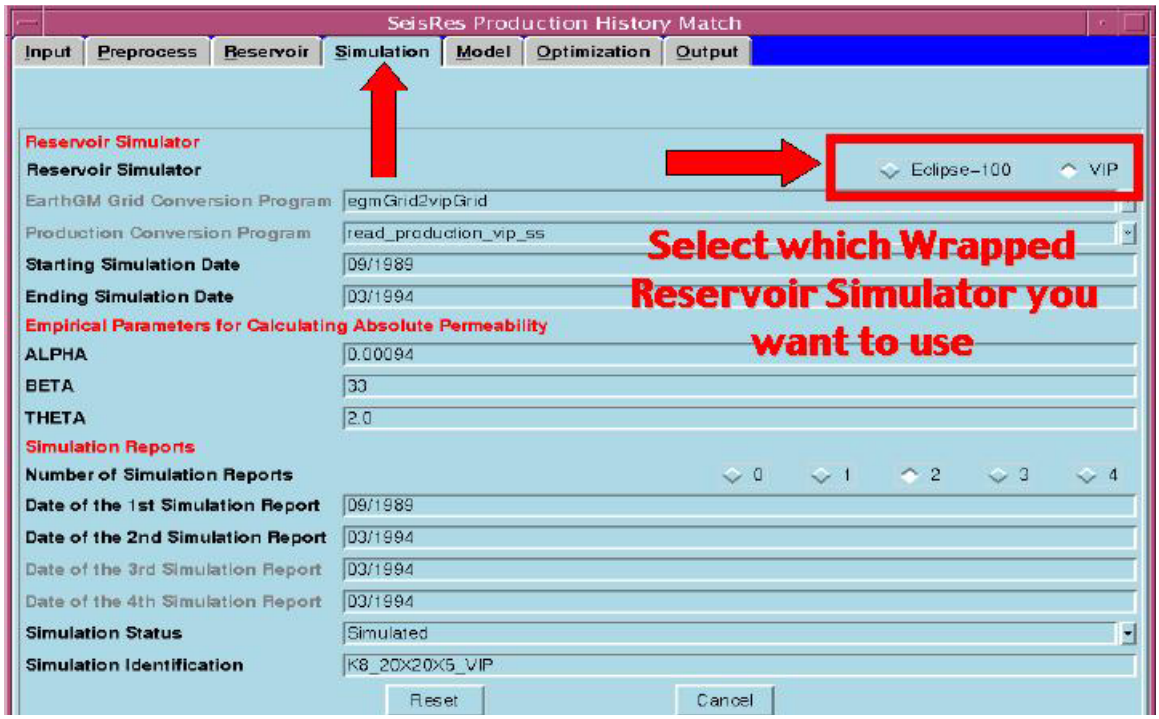
3.3. Wrappers

The SeisRes OF is a very complicated computational system which involving many software applications from vendors as well as proprietary legacy codes from Western Geophysical. The SeisRes workflow may involve many different asset team members working on many different applications which may be distributed on different machines in different countries taht are connected through the network. Making trafficing and versioning among many applications in a workflow efficiently in uniform and synchronized ways is what this component wrapper does. A SeisRes wrapper is like a black box that contains an application within. There are pipes connected to the both sides of box, one side is the input and the other is the output. One box can be connected to another box by connecting outputs of one to the input of another, as long as data types in and out of pipes are same. Each pipe of the box is a port that is identified by name. There is only one type of data that is allowed to flow through the pipe, and that is defined by each application.

Each wrapper box has the following functionality. First it can execute the application as soon as inputs required by the application are all satisfied. Second, it sends events about the status of the execution to the event server. Third, it checks data types to match those coming through the pipe to those needed by the specific application. If the data type coming from another pipe is different from the data type required, then the wrapper invokes the appropriate formatting program to convert the data to the proper type, if there is a formatting program in the registry for the type of conversion required. Each wrapper is implemented in C++, and then compiled and tested for unix systems running on Sun, SGI and Linux operating systems.

3.3.1. Wrapper Registration

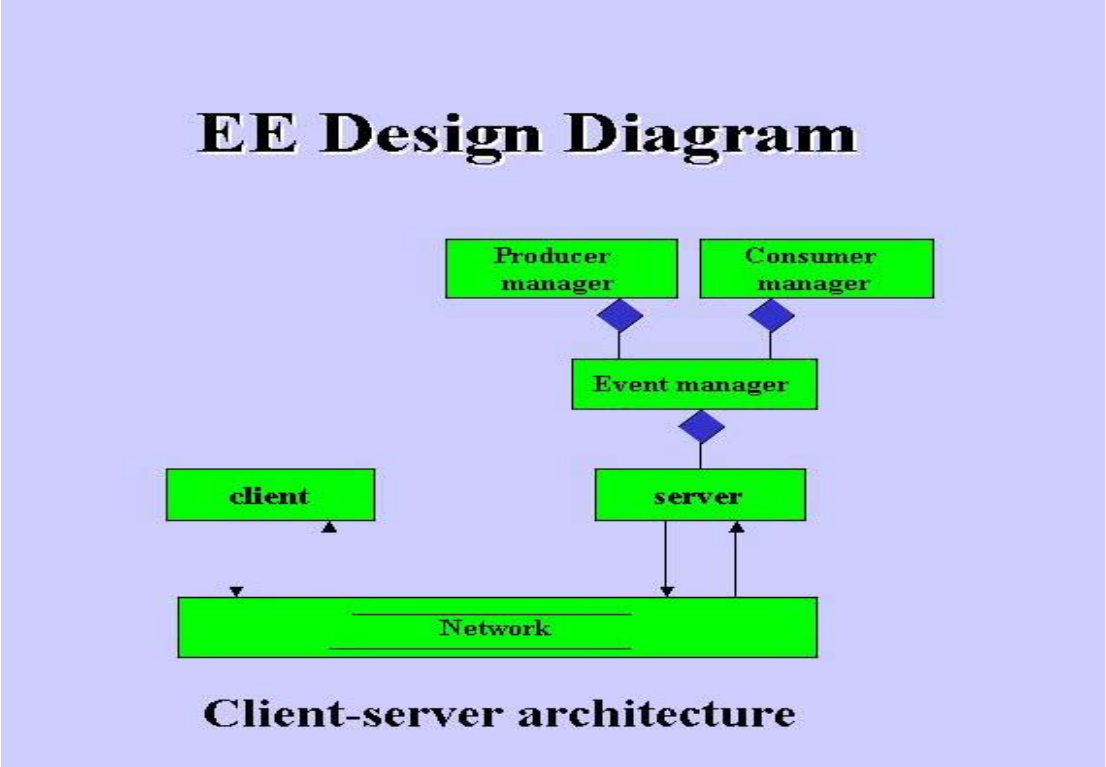
To wrapper a new vendor application, the application must be registered. This registration creates an application specification that is in the form of an appSpec object. This application spec object is stored in the data repository so that in the future, the wrapper can obtain information about this application. If the application reads and writes files, then information about these files must also be registered. This process is called to create a file specification in the form of a fileSpec object. This fileSpec object will also be stored in the data repository for the wrapper to use to obtain information about the kind of data needed in the pipe.



3.3.2. Description of srWrapper

The srWrapper class is designed to be an automatic wrapper box. There are two kinds of I/O ports, the file port and parameter port. The file port indicates a file will be attached to the port. The parameter port means that port holds a parameter value such as a string, integer, float etc.

The srWrapper class provides mechanisms to create a box, add input and output ports, set values for the port, connect ports from one wrapper to another, and execute the application. It also provides the query mechanism to allow the user to ask the box to obtain information about what's goiing on inside the box.



3.4. Event Handling Mechanism

The SeisRes OF software is an integrated, distributed system that seamlessly connects [or includes] many vendor applications and codes. A typical SeisRes job involves many vendor programs running at any particular time. In traditional computer applications, such as sequential batch processing, the user of each is responsible for monitoring the status of his job. There is normally no communication among individual application programs. In our SeisRes, OF, we have built is a sophisticated event handler to monitor the progress and status of each of the processes.

A SeisRes event is a piece of information generated from the client application. This information is delegated to interested parties who are expecting such information. For example, if you want to visualize intermediate results when running a simulation, then the SeisRes simulator can send an event to the visualizer. Also the data object can be delivered to the other application.

The Event Handler keeps books on all information vital to the end-users and synchronizes multiple executions. The synchronization is achieved through the Event Handling Service by utilizing a centralized messaging system that allows all job processes to communicate with each other and report their status and exceptions. The Event Handler is implemented as a centralized server which uses sockets to communicate with clients. The event handler server can register clients as either event producers or event consumers.

Our implementation of the event handler client-server uses a "poll" model. That is, the client has to poll the server to find the data in which it is interested. Produced events are stored on the server for interested parties to fetch. This is the so-called "polling-model". In addition, events can be pushed back to the clients who are listening, this is so-called "push-model".

3.5. Event Exceptions (EE)

There are 5 different parts of the EE component. The EE client is for applications to send and receive events. The EE server serves all EE clients and manages events. The event manager provides APIs to the event server. The producer manager provides APIs for the event manager to manage from producers, and the consumer manager provides APIs for the event manager to manage for consumers.

3.5.1. Event Client

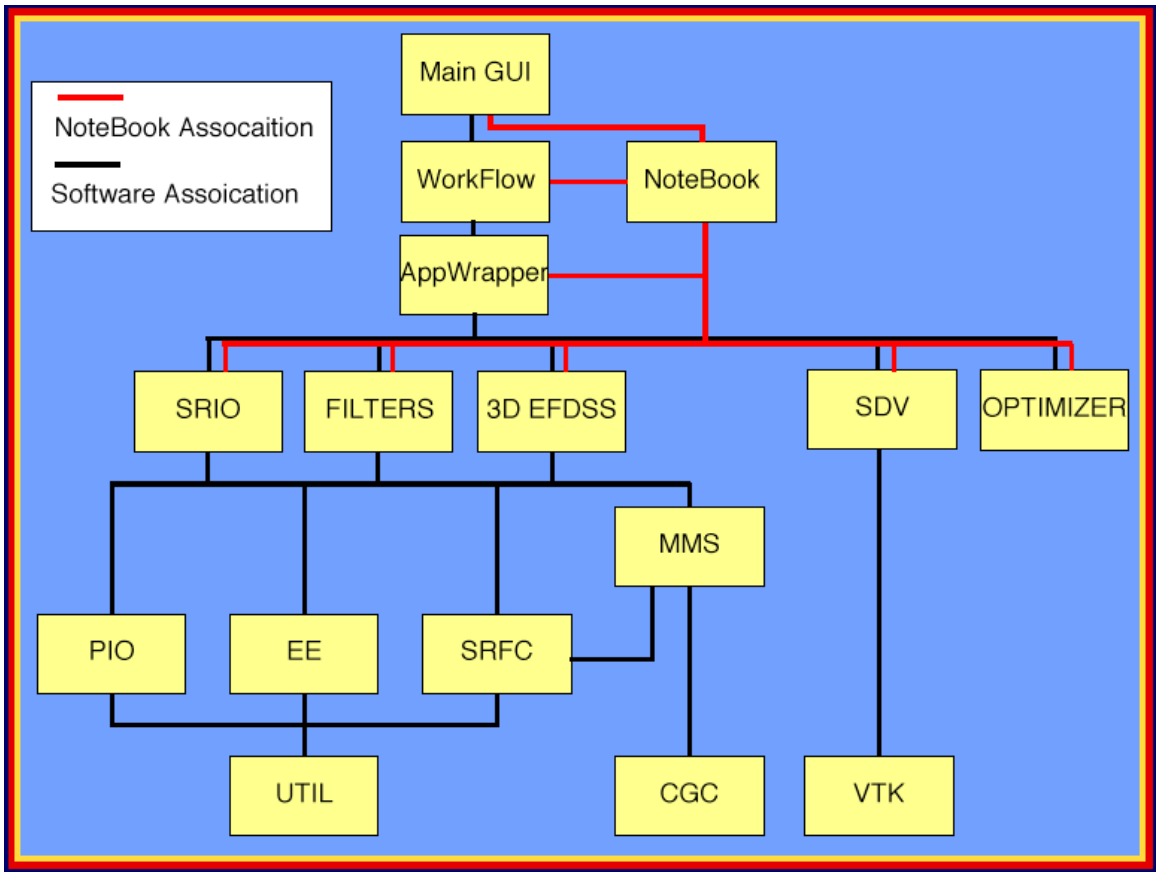
The event client is designed for applications to communicate with the event server through the TCP/IP connection. This client provides all necessary APIs for an application to send and receive events and to do queries. A client first has to register itself as a producer or consumer. To produce events, the client must be registered as producer, then adding events to the server is granted. To consume events, the client must be registered as consumer, then the client can poll the server about events it is interested in. Also the client can tell the event server what events it is interested in or producers it is expecting events from. Then the sever can push events back to the registered consumers.

3.5.2. Event Server

The Event Server is a service provider that serves event clients. It is responsible to register clients, manage events, answer client’s queries, push events back to registered clients, etc. The server depends on the event manager to do all the work. The event manager then further manages two other managers, producer manger and consumer manager. The producer manager is responsible for the addition of events from the client to the event queue, and retrieval of events for consumers. The producer manager manages a list of producer, and each producer will then manage an event queue and a consumer queue. Events produced by this producer are queued to the event queue which has a priority protocol of first-in-first-out (FIFO). All consumers who are interested in this producer are queued on the consumer queue. The consumer manager manages a list of consumers registered on the server. Each registered consumer then manages its own event type queue and producer queue. The event type queue stores all event types this consumer is interested in, and the producer queue stores producers interested by this consumer. There are two version of EE server implemented: single thread and multi-thread . The latter is designed to handle multiple clients at the same time.

3.6. Container Foundation Classes

SeisRes is a plug-and-play environment built upon fundamental "Lego" blocks called foundation classes. The API of each of these classes is exposed to scripting languages such as Tcl, Python, Pearl and Java, allowing fast prototyping of new applications and tools. For example, the active notebook development draws heavily from these packages through the Tcl scripting language. We followed the STL design for most of the packages, which is divided into containers and algorithms/filters.



3.7. Util (SeisRes Utility) package

The util package provides a set of C++ classes categorized into data containers, such as arrays, algorithm classes related to containers, and utility classes for string, system and resource information, unix file and directory manipulation, and pattern matching.

Data containers are arrays of up to 6 dimensions. Matrices and base array classes are generic numerical arrays and are derived from generic arrays.

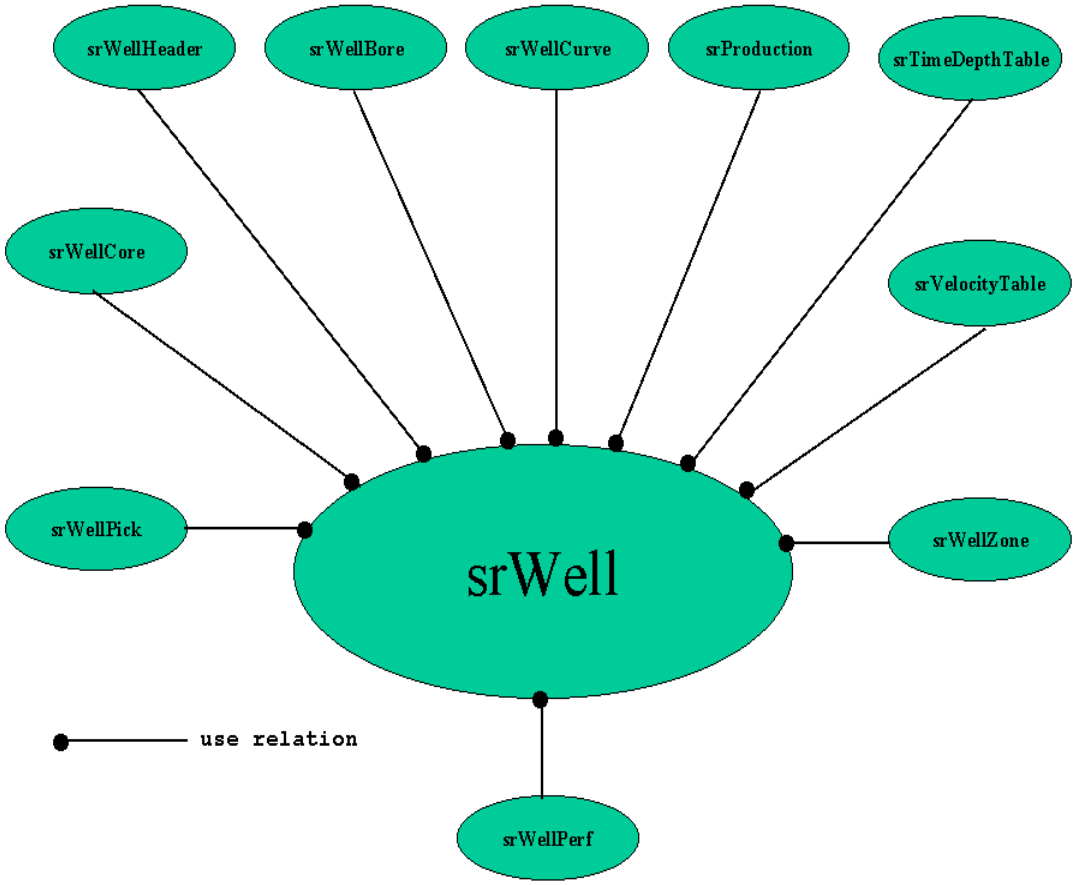
Numerical array classes have overloaded numerical operators such +, -, *, /, += etc. The same design rule applies to matrix in both 2D and 3D.

Our class strings are a subset of the standard string class provided by the C++ language. However, ours have some special string manipulation methods widely used by all SeisRes packages. The pattern matching class does pattern matching just like its name says. The class SystemInfo allows the application to obtain system information such as time, login name, system resource information etc. The class FileInfo allows applications to get information about a unix file. The unixDirUtil class is used to generate file name tree structures of a unix directory. Algorithm classes are related to each data container. One design rule for SeisRes util classes is that we separate containers from algorithms. This design makes container classes more reusable in the future.

3.8. SRFC (SeisRes Foundation Classes) package

The SRFC package contains the foundation classes that implement a set of data containers for specific geological and geophysical datasets. Typical data types used in all geological and geophysical softwares are volumetric data (3D seismic, 3D velocity, etc), well data including well culture data, well bore (well path geometry), well logs, well pick, zones, perfs, time-depth conversion tables, velocity tables, cores. Other data types are horizon, fault, reservoir model and various tables used in the fluid simulation software. Since these data used many different packages with very different formats, an internal format for each data type described is desired in the SeisRes system. Currently srfc package provides containers for all these data types. These containers is only used to hold data with help of access (get methods) and manipulation (set method) to communicate with the object. There is no algorithms implemented for these containers. Algorithms are implemented in the package filter. This design will make these containers extensible and reusable in the future.

To better understand *SRFC* container design, let us take a detailed view of the class *srWell* which uses many other containers to implement collected well data.



The class *srWell* design diagram

3.9. SeisRes Data Input and Output (SRIO) package

Sources of the SeisRes data are from many different legacy software such as traditional interpretation applications (e.g., Landmark, GeoQuest), complex seismic data processing software (e.g., OMEGA), reservoir characterization software (e.g., EarthGM 3D), fluid simulation software (e.g., VIP, ECLIPS), visualization software and many others. These applications take different data formats as input and generate many different data formats as output. The SeisRes loop is a large optimization system that incorporates many datasets to generate the best reservoir models. It is impractical and also impossible to implement OF software that takes into account every possible data format. We have built an OF software that operates on well-designed and often used data objects, but other data types must be convert to the internal data types. The SRIO library serves this purpose. The SRIO package consists of a set of classes that define public APIs to all applications, and derived classes for each different application software package. Every class has two APIs: read and write. The

read method reads client data and converts to *SRFC* or *MMS* objects. The write method converts *SRFC* or *MMS* objects into client data format.

3.10. SeisRes Data Filtering (Filter) package

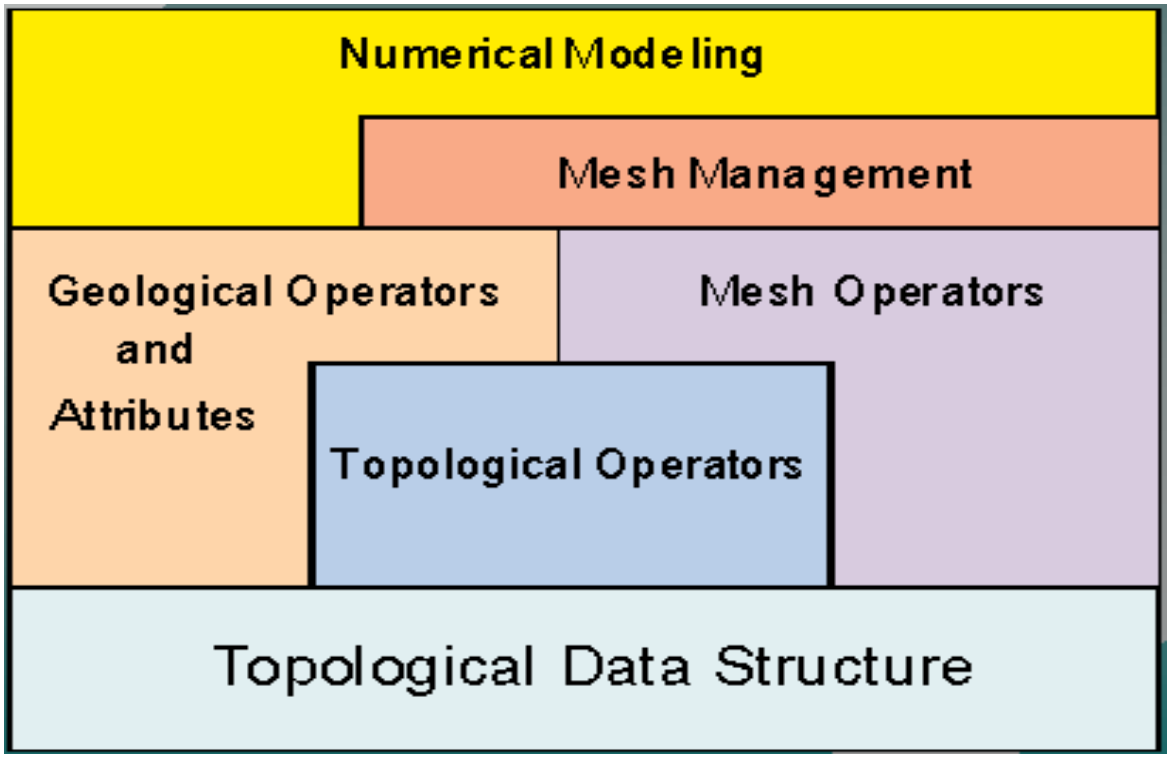
Packages like SRFC do not provide any APIs to manipulate those data objects. For example, well bore data usually comes with (x, y) coordinates plus vertical depth and measured depth, but a well bore with two-way travel time is often used when comparing with seismic data. To get two-way travel time, a time-depth conversion table and algorithm are needed. Another example is mapping data to a horizon, this process involves two different data objects: horizon and volumetric data. An interpolation algorithm has to be implemented to obtain data for each point of the horizon. The filter package provides a set of classes to filter specific data from SeisRes container objects algorithmically to satisfy the above described requirements.

3.11. MultiMesh System (CGC and MMS) packages

Many SeisRes data types consists of two different kinds of data: geometry and associated attributes. Correct registration of attributes with geometry is a critical part of the SeisRes OF system. This registration is implemented with help of the a shared earth model (CGC) and multimesh system *(mms)* developed by Ulisses Mello of IBM. The CGC system creates and maintains a topological representation of an earth model that is be used as the reference model in each SeisRes project. The *mms* provides containers for all different kinds of geometry objects and meshes needed by applications such as point, polyline, surface, polygons, tetrahedra, bounding box and more. To associate the *srfc* container with the geometry, a set of field classes is implemented in this package. These fields are generally designed for 2D and 3D structured and non-structured datasets. The 2D structured fields are mapped horizons and faults, 2D non-structured fields are triangulated horizons and faults. 3D structured fields include regular, rectilinear and curvilinear fields. The 3D non-structured field is an irregular mesh. These field containers are like srfc containers. They are objects to store geometry and attributes. Available APIs are set and get methods only. Each field class has methods to encode and decode for overloaded *xdr* input and output streams.

The high-level architecture of our modeling framework is shown below. This is a layered architectural software pattern in which each layer has a distinct role in the framework. In the base of the framework, we use a topological representation based on the Radial Edge Data Structure- REDS- which is used to represent complex non-manifold topologies. REDS explicitly stores the two uses (sides) of a face by two regions that share the same face. Each face use is bounded by one or more loops uses, which in turn are composed of an alternating sequence of edge uses and vertex uses. The REDS is general and can represent non-manifold topology. We make extensive use of high level topological operators for building earth models because topological data structures are in general too

complex to be manipulated directly. Edges of REDS may represent well paths, a set of faces or a shell may represent the surface of a fault or seismic horizons, and set of regions may represent geological layers and fault zones. The associated meshing and remeshing of these geological objects is based on the connectivity and spatial subdivision information stored in REDS.



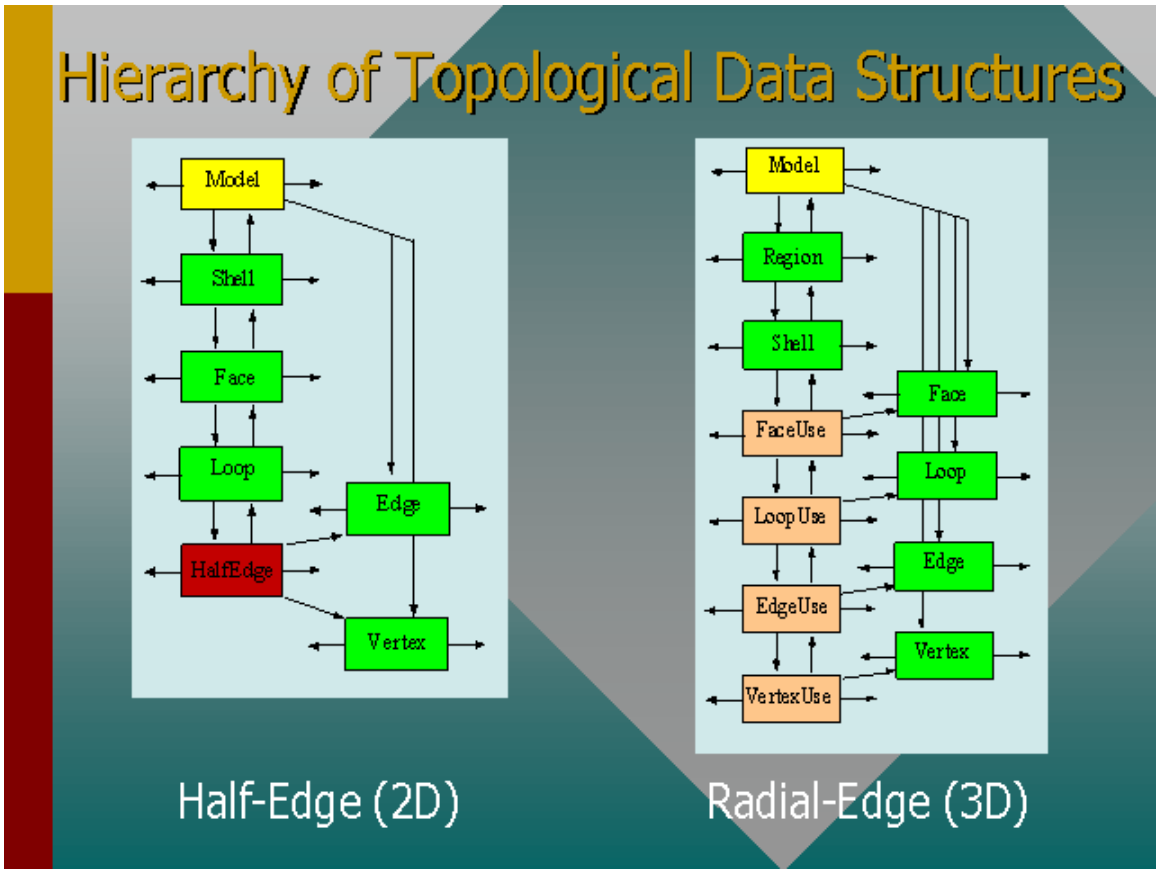
Architecture of the Meshing Services.

We implemented the REDS and its topological operators using C++, and this implementation is very compact, having less than 50 C++ classes. The REDS is the component that stores the topological and geometrical representation of an earth model. MMS is the layer that generates and manages numerical meshes associated with earth model sub-regions. It is important to note that meshes are treated as attributes of geological entities such as blocks, horizons, layers and faults. Hence, a mesh is not the model, but only one possible realization of a model or a sub-region of the model. Using CGC and MMS, the meshing operators can provide multiple mesh representations with multiple resolutions of a given earth model. One particular important application of these operators is in the area of reservoir characterization where it is commonly necessary to upscale geological grids to a resolution that the flow simulation can be executed in available computers. Operations between coarse and fine resolution grids are greatly facilitated in this framework.

3.11.1. Shared Earth Model (CGC)

In order to share geological information among the various applications used in SeisRes, a shared earth model builder was implemented (CGC). An earth model is built from a set of polygonal surfaces defining the boundaries of geological structures. CGC has various geometrical operators built-in to facilitate the creation of proper 3D representations of geological entities such as faulted reservoirs.

The structural seismic interpretation of the reservoir provides the geometrical elements (set of polygonal surfaces) necessary to create a reservoir earth model and its spatial subdivision. The geometrical and topological description of an earth model is obtained incrementally by adding polygonal surfaces sequentially to the model. The resulting earth model contains the space partitions (regions of space) defined by these surfaces. Meshes can be generated for the entire earth model as well as for each individual region of the model. Each region can have multiple meshes with various resolutions associated with it (below). These region meshes are treated as attributes of the model's region similarly to other physical attributes such as lithology, density and velocity. In the current implementation, a region maintains a list of the name (String) of the mesh objects associated with it. These meshes are stored in the SeisRes repository and can be easily queried and retrieved by name.

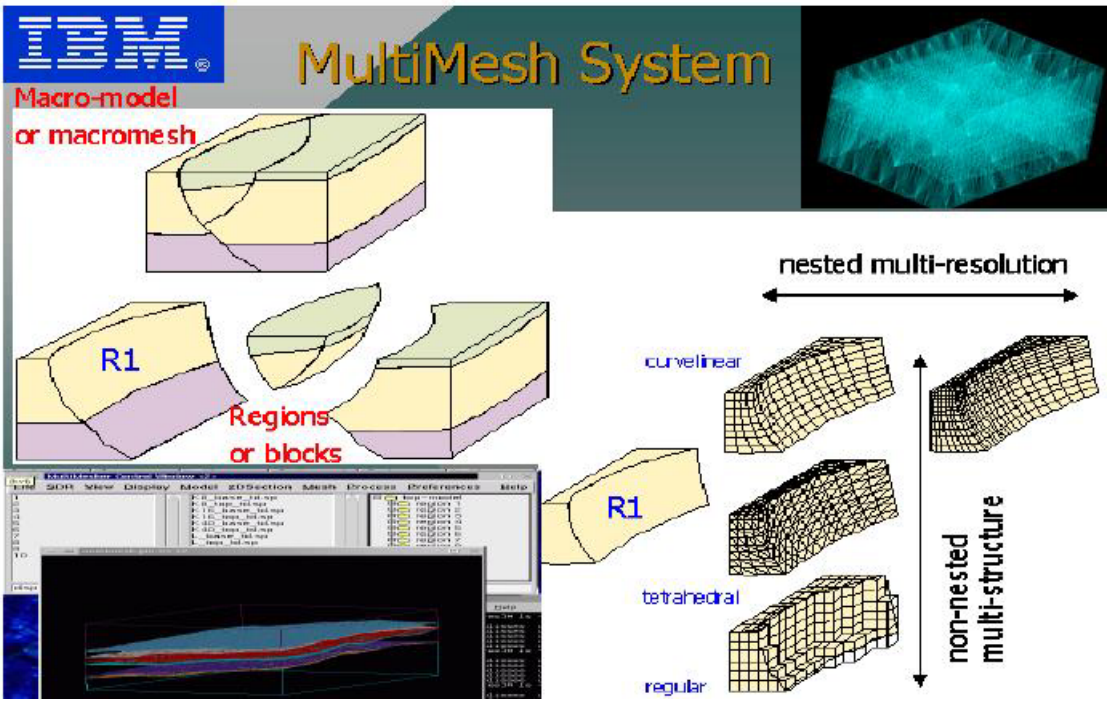


3.11.2. Radial Edge Data Structure

It is important to realize that this framework also allows us to manipulate voxel representations (regular meshes) of the earth model with great flexibility. For example, we treat 3-D seismic volumes as regular grid attributes of the earth model. Because the earth model has explicit information of the geometry of geological objects in the model, we can easily select, for example, only the seismic voxels of a particular reservoir object.

3.11.3. MultiMesh System (MMS)

Meshes necessary as input for some of the SeisRes applications are generated automatically by the MultiMesh System. This system was designed to integrate and transfer information in numerical meshes among applications that require distinct mesh representations. The meshes are discrete realizations of the earth model. This is analogous to the reservoir characterization process in which each reservoir realization is just a possible representation of the reservoir. A particular mesh (regular, curvilinear or tetrahedral is just a possible representation of the earth model). Multimesh is able to generate structured (regular and rectilinear) and non-structured (tetrahedral) meshes. It can manipulate all meshes necessary to integrate applications (Eclipse, FDM, EarthGM). IBM has contributed to building reservoir classes (SRFC) on top of some MultiMesh classes, and final work will focus on the integration of Multimesh with other applications.



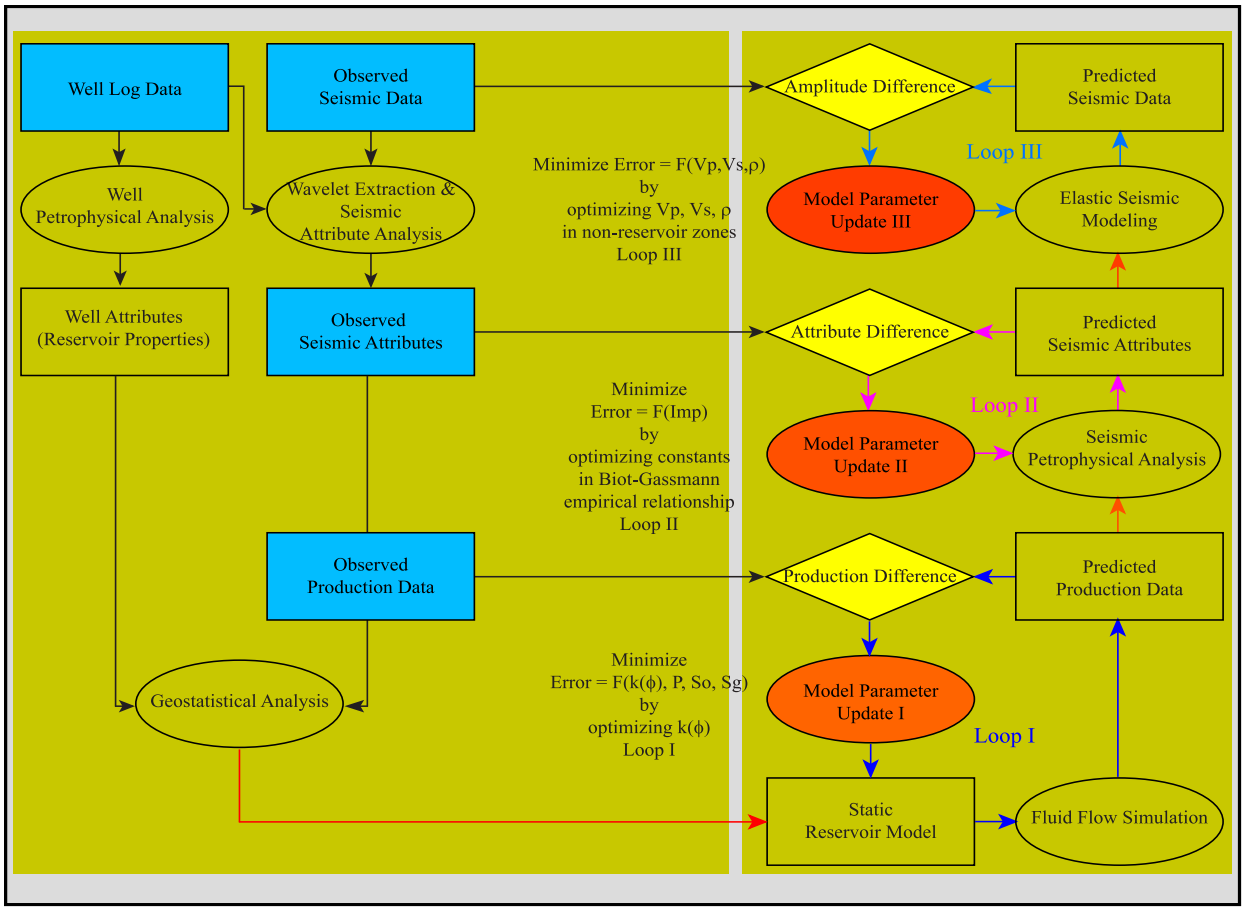
The design of the mesh classes in MMS has been influenced by the design of the VTK mesh classes. However, our field classes are much more flexible. We made the decision to have mesh design close to VTK's because it makes simpler to create VTK mesh objects for visualization.

3.12. Optimization Tool Kit

The Optimization Tool Kit is designed to be a set of tools can be deployed at any time and any place within the SeisRes OF to provide parameter estimation services. It is implemented as a loosely coupled component because the need for parameter estimation varies from app to app. The principal underlying this choice of design is that it allows a selection of options, including hybrid options combining algorithms from different categories, to produce the most appropriate procedure. The technical goal is to quickly implement sub-optimization loops to facilitate the entire optimization process for the seismic reservoir simulation.

The optimizer consists of three components: optimization solvers, forward simulation wrappers, and simulation data converters. The forward simulation solver and simulation data converters are developed separately for reservoir property characterizer, reservoir simulator, petrophysical property characterizer, and 3D finite-difference simulator.

The Optimization Laboratory is implemented according to the workflow illustrated below. The wrappers are developed to aim for the smooth execution of each individual sub-problem. The sub-problems are illustrated as different color in the diagram. It is clear that each sub-problem involves one or more forward simulation processes that generate the predicted data from the optimization model parameters. Each sub-problem also involves solving an optimization problem.



The optimization component consists of several optimization algorithms in the form of executable programs. Each of these optimization wrappers offers the following functionality:

1) Able to obtain a model parameter update,
2) Able to perform interactive update as well as automatic update model parameters, and
3) Able to send and receive requests to and from other application wrappers including other optimization wrappers.

We have built three optimization algorithms in the SeisRes Optimization Laboratory. They are a generalized linear solver (GLS), a generalized nonlinear solver (the modified LevenBerg-Marquardt Solver, LMDIF), and a constrained Genetic Algorithm solver (GENOCOP III). The GENOCOP III itself is often considered a hueristic, hybrid solution to some optimization problems. The GLS wrapper is almost completed; we have defined data I/O format for GLS, and implemented automatic scaling of the columns of Jacobian matrix; the control mechanism is implemented.

3.12.1. Fluid Flow Simulation Wrapper

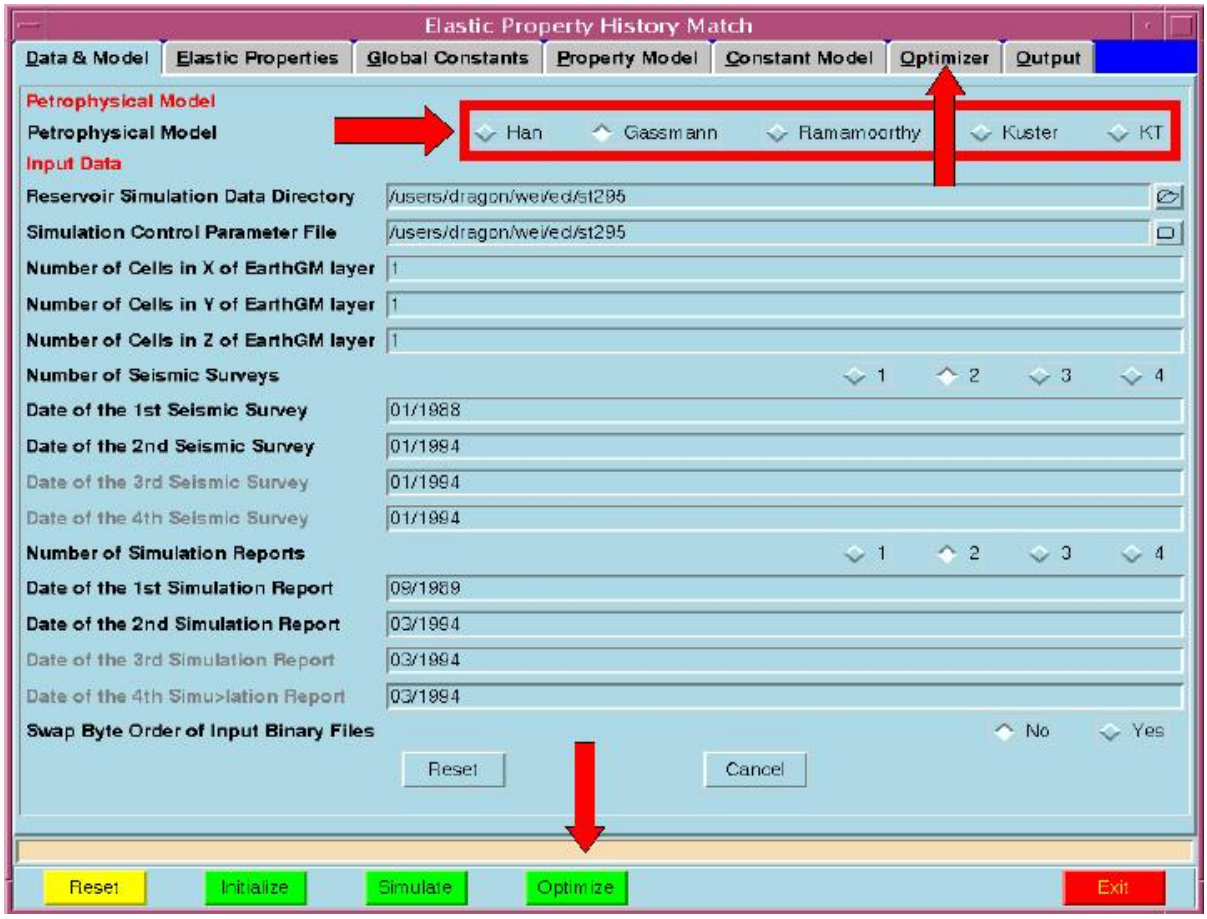
We have created a TCL/TK wrapper that drives the Eclispe and VIP reservoir simulator. We are able to loosely integrate the Eclipse and VIP wrappers with the Optimizer to perform production history matches.

3.12.2. Petrophysical Property Modeling Wrapper

We have written a suite of tools that implement both theoretical and empirical equations published in the literature to match impedances computed using various Biot-Gassman algorithms with observed impedances. We identified 11 key parameters to reproduce the complexity of inverted acoustic impedance. By integrating with the GL solver, we are able to optimize these constants to model the impedance changes from reservoir simulation results.

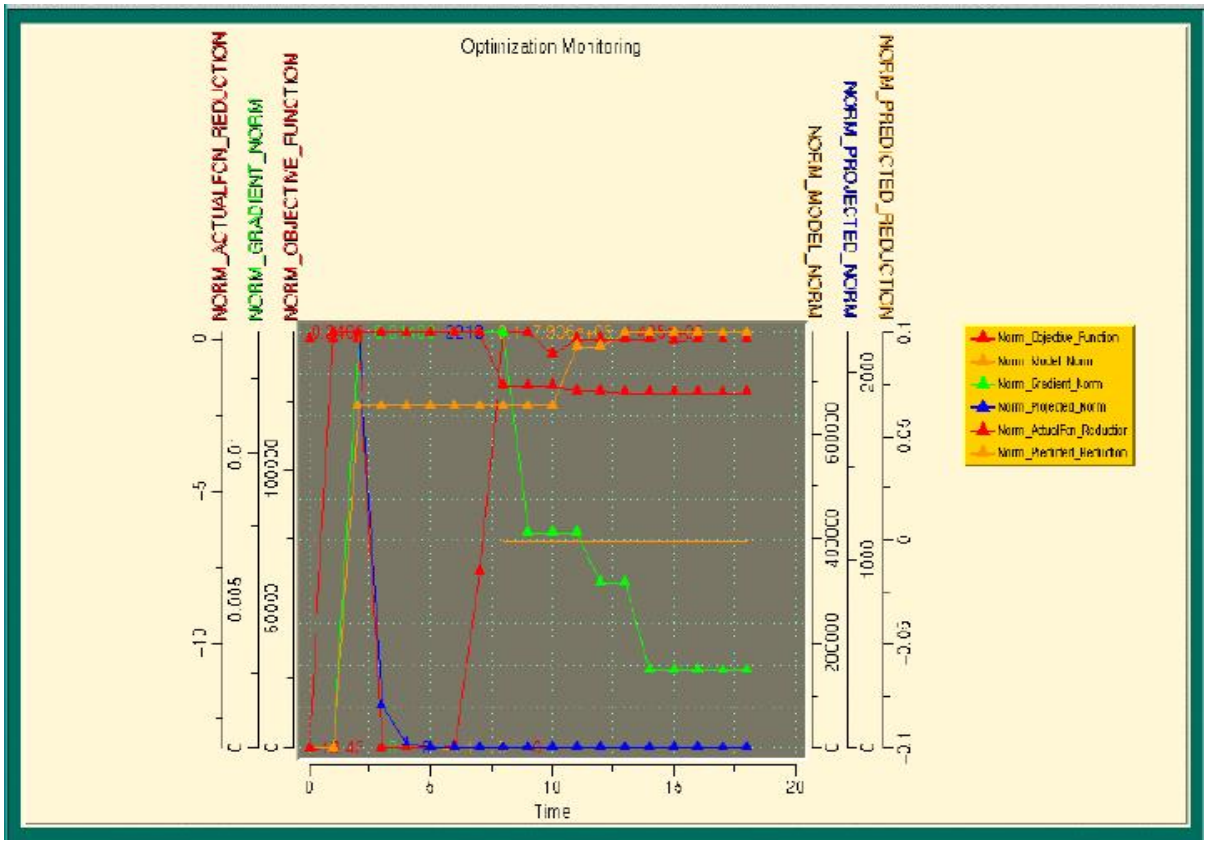
3.12.3. Visualization of Optimizer as Computing

Events are sent during the optimization loop so that a vizualization of the convergences of the gradients and norms computed as the parameters are changed in the reservoir simulation.



3.13. SeisRes Data Viewer

The SeisRes 3D Data Viewer (SDV) has been designed to 1) display a variety of geoscience data types registered in real-world coordinates in a common scene on the Web, 2) use state-of-the-art rendering methods, 3) run on all popular workstations, and 4) be easily extendable by other developers. The prototype



displays seismic binned data (stack, migrated, acoustic impedance volume, etc.), surfaces and well logs. It is integrated into the SeisRes Data Repository.

The reservoir is characterized by multiple sequential seismic surveys; seismic attribute volumes; many well logs of different types and vintages; geostatistically-derived data volumes on regular and stratigraphic grids; fluid saturation volumes; four-dimensional fluid-flow maps; fluid-interfaces, horizon, and fault surfaces and possibly other data types. Being able to view all these data -- spatially registered with respect to one another in the local real-world coordinate system and rendered in a variety of modes so that the interrelationships can be perceived -- is a great help and may be a necessity for understanding spatially complex reservoirs over time.

The OF uses the Visualization Toolkit (vtk). vtk is freeware; its source code is available to anyone via internet download. It provides an interface to 3D graphics that is easy to use relative to OpenGL or other low-level interfaces. It is written in C++ and provides a well-documented C++ API. It also provides an API to Java and the popular scripting languages Tcl/Tk and Python..

The SDV has been developed to the point of visualizing binned seismic data (stacks, migrated volumes, attribute volumes, etc.), well logs and a computer-graphics ASCII file format known as the BYU format. The files are read from specialized formats designed for simplicity in this first demo. Seismic data can be converted from SEGY and logs can be converted from one of the SigmaView formats.

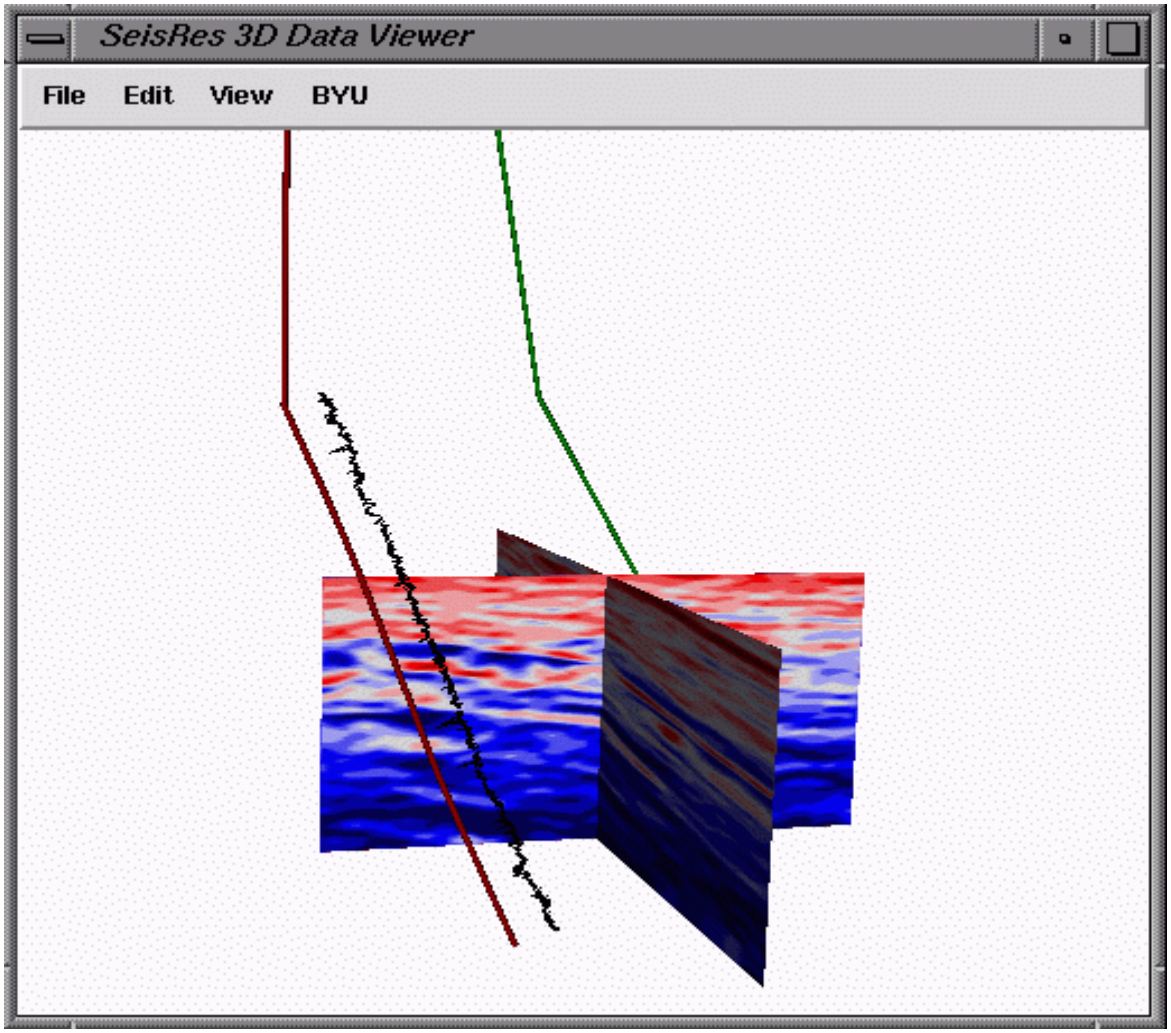
The vtk and all the SDV code is portable to Windows The development has been done on both SGI and Sun workstations with no problems other than some makefile and environment variable peculiarities. Many people use vtk on NT or Linux on PCs. Because the top-level code is written in Tcl/Tk, it can be invoked from a web browser.

The SDV is designed as a central framework and data-specific pipelines. A pipeline is a concept inherent in vtk. All data is processed by a pipeline consisting of the serial connection of a reader or source object to import the data in its native form, various filters to convert it into graphical form, a mapper to generate the graphics primitives, an actor to associate 3D transformations, colors, lights and other graphics properties with the data and a renderer to draw it all. The framework can operate with any one or more of the pipelines, and pipelines can be developed without access to the framework source code.

3.13.1. The Main Viewing Window

The center of attention in the SDV is a single viewing window enclosed in a Tcl/Tk top-level window. All 3D objects are displayed in real-world coordinates here.

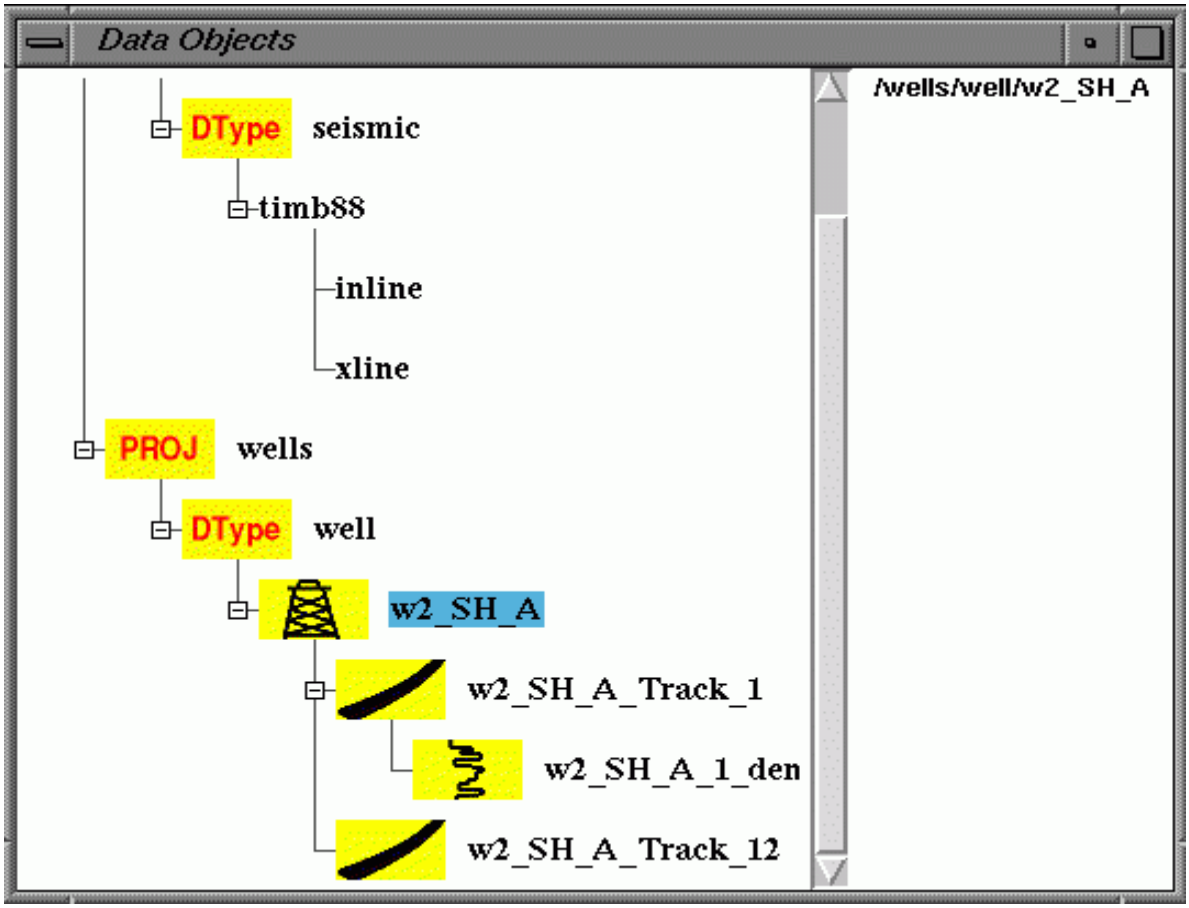
The main window has a typical menu bar across the top. The File, Edit and View buttons were placed on the menu bar by the SDV framework. The BYU button was placed on the menu bar by the BYU pipeline. Since the GUI is written in Tcl script it is easy for pipelines to add objects to it without modifying the framework code.



3.13.2. The Data Object Tree

The second main element of the framework GUI is a graphical data object tree.

This window shows the objects that are loaded into the SDV. The SDV organizes data objects in a tree hierarchy. At the top level is the single instance of the SDV. Second in the hierarchy is a project. Under a project, the data objects are grouped by data type. The tree structure below the data-type level is determined by the data-specific pipeline. Notice that the hierarchy of the well



logs and seismic views are different

The right hand frame of this window is available for displaying information or GUI widgets associated with a single selected component of the tree. In this figure, the project node has been selected as indicated by its blue background. At present, the framework only prints the name of the object. More interesting things can be associated with this selection.

3.13.3. VTK

The framework consists of the viewing window, a graphical data-object tree and the GUI widgets common to all data types. It is not modified by any of the data specific developers. In fact, only the C++ header files , the shared libraries, and the main tcl script is needed for developing new features. A new pipeline is added by adding one or a few lines to the .sdv_resource file, and informing the operating system where to find the tcl scripts and libraries containing the new pipeline code.

Vtk is distributed in source-code form so that it can be built on most common computers: most Unixes including Sun, SGI, HP, and AIX, Linux and Windows NT. It uses a hardware implementation of OpenGL if one is available on the host computer (Unix or NT) or software implementations of OpenGL, or a Windows-specific graphics language. It has some facilities for multiple graphics pipes such as are found in CAVE environments.

Vtk is maintained by Kitware, Inc. and is distributed from an ftp server at Rensselaer Polytechnic Institute. It requires a C++ compiler to "make" an executable version. Many examples are provided to allow the user to see how 3D objects can be visualized and to illustrate how the various classes can be used.

Vtk has APIs for Python and Java. We elected not to use Java since the Java version uses Java3D for its underlying graphics support. Java3D does not perform nearly as well as OpenGL at present. It may be an option in the future. Python is a much better-structured scripting language than is Tcl/Tk but is far less widely used. We have probably avoided many bugs by using Tcl/Tk. The decision to use Tcl/Tk needs to be reviewed periodically. A change from Tcl/Tk to Python would be straightforward. A change to Java would likely entail a complete recoding of the Tcl portion of the SDV framework: less than ten pages of code at present.