

**THE TELE-IMMERSIVE DATA EXPLORER (TIDE): A DISTRIBUTED
ARCHITECTURE FOR TELE-IMMERSIVE SCIENTIFIC VISUALIZATION**

BY

NIKITA SAWANT

B.E., Computer Engineering, University of Bombay, India, 1996

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering and Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2000

Chicago, Illinois

ACKNOWLEDGEMENTS

I would primarily like to thank Dr. Jason Leigh and Dr. Andrew Johnson, who have been my mentors in my past two years in EVL. I owe a huge debt of gratitude to them for **always** having time to answer my queries and doubts.

I would also like to thank each and every member of the CAVERN group for supporting me through all these years, especially Chris Scharver who has been an invaluable source of help in implementing my thesis.

Finally I would like to thank Dr. Thomas Defanti and Maxine Brown, for having given me the opportunity to work in EVL and all the other students and staff members of EVL for having supported me in the past two years.

NS

TABLE OF CONTENTS

<u>CHAPTER</u>		<u>PAGE</u>
1	INTRODUCTION TO VIRTUAL REALITY AND SCIENTIFIC VISUALIZATION	1
	1.1 Limitations of Current Scientific Visualization Systems	2
	1.2 Application of Tele-immersion to Scientific Visualization	3
2	BACKGROUND	7
	2.1 Scientific visualization systems: from two dimensions to virtual reality	8
	2.2 Comparison of Visualization Systems	9
	2.2.1 Earlier Problem Solving Environments	10
	2.2.2 MineSet	12
	2.2.3 Virtual Reality Systems	14
	2.2.3.1 Virtual Wind Tunnel	14
	2.2.3.2 CAVEvis	15
	2.2.3.3 Cosmic Worm & CAVESudy	16
	2.2.3.4 Visualization Systems for Geoscience Data	17
	2.3 Common Traits in Visualization Systems	18
	2.4 Shortcomings of current systems and ways to overcome them	20
	2.5 Characteristics of Data	22
	2.5.1 Data Types	22
	2.5.2 Data Visualization	22
	2.5.3 Data Storage and Retrieval	23
	2.6 Summary	23
3	IMPLEMENTATION OF TIDE	26
	3.1 Features of TIDE	26
	3.2 Object Oriented Design	29
	3.3 Frameworks	29
	3.4 Basic components of tele-immersive applications for visualization of large data	30
	3.4.1 Data Visualization	30
	3.4.2 Handling Large Data	31
	3.4.3 Leveraging Collaboration	31
	3.5 Conceptual Organization of TIDE	32
	3.5.1 Remote Data and Computation Services	34
	3.5.2 Tele-immersion Server	34
	3.5.3 Tele-immersion Client	35
	3.6 The TIDE Architecture	36
	3.6.1 Tele-immersion Client	36
	3.6.2 Tele-immersion Server	39
	3.6.2.1 Co-Servers of the Teleimmersion Server	41
	3.6.2.1.1 The World Server	41
	3.6.2.1.2 The File Server	43

3.7	The TIDE Framework	43
3.7.1	The Tele-immersion Server	44
3.7.1.1	Communication with the Clients	45
3.7.1.2	Handling Client Requests	46
3.7.1.3	Data Representation and Conversion	47
3.7.2	Tele-Immersion Client	48
3.7.2.1	Limbo	48
3.7.2.2	User Interface Protocols	50
3.7.2.2.1	Query Interface	50
3.7.2.2.2	Visualization Interface	51
3.7.2.2.3	Network Mediator	51
3.8	A Simple TIDE Application	53
3.8.1	Extensions required for the TIC	53
3.8.2	Extensions required for the TIS	55
4	APPLICATION OF TIDE	59
4.1	Application Domain	59
4.2	Application specific details for the DSTP TIC	61
4.2.1	2D Query Interface	62
4.2.2	Wand Interface	63
4.2.3	Communication with the TIS	63
4.3	Application specific details for the DSTP TIS	64
4.3.1	Data Representation	64
4.3.2	Data Conversion	65
4.3.3	Application Specific Commands	65
4.4	Demonstration of TIDE at Supercomputing '99	66
5	EVALUATING THE DESIGN AND PERFORMANCE	69
5.1	Good Features of Tide	69
5.1.1	Centralized control of collaboration	69
5.1.2	Centralized Location for Data	69
5.1.3	Multiple Processes	70
5.1.4	Large Data Visualization	70
5.1.5	Extensibility of the TIDE Framework	71
6	FUTURE WORK	73
6.1	Handling Time Dependent Data	73
6.2	Synchronous and Asynchronous Communication	75
6.3	Persistent Environment	76
6.4	3D interfaces for virtual environments	76
6.5	Data model for data representation	77
7	CONCLUSION	78
	APPENDIX	79
	CITED LITERATURE	85
	VITA	89

LIST OF FIGURES

<u>Figure</u>		<u>Page</u>
1	AVS Interface: (a) A Visualization Network (b) Visualization	11
2	MineSet: Splat Visualizer view of a three-dimensional landscape with columns from the adult94 sample dataset mapped to axes, sliders, color, and opacity	13
3	Comparison of Visualization Systems	24
4	A Client collaborating with a remote Client (seen as an avatar)	28
5	TIDE's Visualization Process	34
6	Distributed Architecture of TIDE	36
7	UML Class diagram for the TIS	44
8	UML Class diagram for the TIC	48
9	Extensions made to the TIC framework for the TIDE-DSTP application	61
10	Extensions made to the TIS framework for the TIDE-DSTP application	64
11	TIDE at SC99: A user running the TIDE Client on a PowerWall, at the ASCI booth. Inset: Another user collaborating from an ImmersaDesk	66
12	(a): A 2D graph of Temperature (Y-axis) vs. Ozone(X-axis), color is mapped to latitude generated by DSTP Client. (b): A 3D graph of Longitude(X-axis), Latitude(Y-axis), Ozone(Z-axis) and Temperature (color) generated by TIDE	67
13	Command Pattern	80
14	Observer Pattern	82
15	Template Method Pattern	83

ABBREVIATIONS

3D	Three-Dimensional
2D	Two-dimensional
VR	Virtual Reality
TI	Tele-Immersion
TIDE	Tele-Immersive Data Explorer
AVS	Advanced Visual System
CVE	Collaborative Virtual Environment
CVR	Collaborative Virtual Reality
DSTP	Data Space Transfer Protocol
RDBMS	Relational Database Management System

SUMMARY

Visualization is the key methodology that gives the research scientist /analyst an insight into data that may be generated from various sources such as computational simulations and scientific experiments. A recent trend has been towards the use of Virtual Reality (VR) technology for data visualization, to give the user a realistic insight into the data.

Teleimmersion is the amalgamation of data mining and significant computation with collaborative virtual reality. It allows multiple networked users to participate in a shared virtual environment. The collaborators can talk to each other and can see each other in the environment. Teleimmersion augments the data visualization and analysis process to produce a new genre of applications.

With advances in the fields of computational science and engineering we now have faster computers that generate data, which is in the range of a few hundred megabytes to several terabytes. Large data visualization poses a new challenge to the visualization community, as most of the existing systems are not capable of visualizing vast amounts of data.

Massive data sets and collaborative visualization add a new dimension to ongoing research on visualization in virtual environments. Collaborative extensions have been added to existing non-VR systems. A number of dedicated single user systems allow the user to visualize large datasets in virtual environments. Most of these systems are application specific and cannot be extended. Even though these systems cater to a diverse set of application domains, some trends and patterns in their approach to visualize data are discernable. These design features can be reused in future applications.

SUMMARY (continuation)

The main contribution of this thesis is in proposing the Teleimmersive Data Explorer (TIDE): a general architecture that blends collaboration with the visualization, which can be used by application developers for rapidly building teleimmersive applications for large data visualization. A basis of the TIDE architecture has been implemented.

The following chapters describe in detail existing visualization systems, this forms the basis to identify the characteristics of visualization systems. An analysis of the problem of large data visualization and collaborative visualization is done to identify possible solutions. A framework for TIDE is proposed, implemented and evaluated.

Currently most scientific visualization systems limit the user to visualize data on a two dimensional (2D) desktop. Since Virtual Reality (VR) can make the data visualization process more realistic by providing a third dimension, it seems natural to make use of this technology for visualization instead of the traditional 2D desktops.

VR systems such as the CAVE(CAVE Automatic Virtual Environment)[16] immerse the user in a virtual environment by using surround screen rear projection technology[16], in which the user is fully immersed in the Virtual Environment (VE). Immersive systems were designed primarily for scientific visualization purposes, to provide the user with the ability to walk around and analyze scientific data and to make the data exploration process more realistic. Allowing a user to interact and collaborate with other users in the environment can further enrich the immersive experience.

However very little effort has been made in the development of general-purpose problem solving environments for data exploration in immersive environments. Even less work has been done in the development of collaborative scalable immersive visualization systems. Hence the focus of this thesis is to develop an extensible framework for data exploration that allows multiple users at different geographic locations to collaborate in a data analysis session involving a very large data set, in an environment.

There exist a number of general-purpose scientific visualization systems that aide the user in performing a variety of tasks such as data exploration and visualization, however they confine the user to a 2D environment, the workstation. The users of these systems use a two-dimensional interface to interact with and visualize data containing more than two-dimensions. This can be very limiting in that the spatial relationships between various data points may be unclear. Virtual Reality systems such as the CAVE[16] (CAVE Automatic Virtual Environment) make three-dimensional spatial interaction possible.

Another problem that the Scientific Visualization community faces pertains to the visualization of large data. Simulation results today can surpass 100Gbytes, and these are expected to scale with the ability of supercomputers to generate them. Most general-purpose scientific visualization systems are based on a data flow model that needs to hold all the data to be visualized in core, and hence cannot be applied to visualize large data sets in the range of hundreds of megabytes to a few terabytes as it is impossible to fit this large a data set into the virtual and physical memory of the workstation used for visualization. Neither do these systems allow multiple remotely connected users to collaborate on the data being visualized, though extensions have been added to make some of the systems collaborative, as an afterthought.

Recent research has led to the development of immersive visualization applications designed specifically for VR devices and for accommodating large data sets; these systems focus more on a particular task and are not collaborative. The goal of TIDE is to define the infrastructure

for scientific visualization applications that can handle large data sets and allow users to visualize data immersively and collaboratively.

Consider the visualization of data resulting from a Computational Fluid Dynamics (CFD) simulation. The numerical data collected consists of time varying scalar and vector fields, in three-dimensional space. Traditional two-dimensional visualization systems would generate three dimensional time dependent models from the data and finally project them on a two-dimensional screen, causing the visualization model to lose an important depth cue: binocular disparity (stereoscopic depth). If the display monitor supports stereo then binocular disparity can also be implemented by projecting different images for the left and the right eye and using shutter glasses to provide the correct image to each eye. Two-dimensional graphics systems generally implement perspective projection, occlusion, lighting and shading to give the notion of depth.

VR systems provide the cues of binocular disparity, motion parallax and convergence in addition to those provided by two-dimensional graphics systems. Motion parallax is implemented by tracking the position of the user's eyes and re-rendering the scene based on the user's perspective. Implementing additional depth cues in a system causes the final rendered image to be more realistic and convincing. One of the benefits gained from using virtual environments for visualization systems, is the near real-time three-dimensional interaction that these systems provide to the users. However, non-immersive VR devices render the images on screens which are perpendicular to the users, many a times causing the user to lose a sense of immersion if he/she looks beyond the screen.

In immersive VR systems such as the CAVE, four rear-projection based screens surround the user. Three screens form the three walls: front, left and right and one forms the floor. Projectors render stereoscopic images onto these screens. This gives the user a sense of immersion into the data, as they get a computer generated realistic view of the data. Contemporary tracking devices that accommodate six degrees-of- freedom i.e. a in three dimensional space (x, y and z) and (i.e. yaw, pitch and roll), can be used as input devices to give the user the ability to interact with the data spatially in three-dimensions. The Visualization of data collected from other sources such as geographic information systems and medical scans can also reap the benefits that these systems offer due to their employment of perspective projection, binocular disparity, motion parallax, and six degree-of-freedom.

Immersive virtual environments are evolving from being mere display environments for rendering the final output of a visualization process[10], to interactive environments. Many of today's applications focus not only on the display of a stereoscopic image on VR display devices, but also on interaction with the data or models being displayed. This adds new capabilities to the field of scientific visualization. Many computational steering[14] and data visualization applications [13] now make use of immersive virtual environments.

Networked Virtual Environments (net-VEs) or Collaborative Virtual Environments (CVEs) form the base for a new generation of applications, which allow users, located at different locations to interact with each other in real time in a shared space. Immersive scientific visualization systems can be made collaborative by utilizing existing high bandwidth Local Area Networks (LANs), Wide Area Networks (WANs) and the Internet as communication channels. Currently most visualization systems are stand-alone systems, where a single user

analyzes data on a conventional two-dimensional screen or in some cases in an immersive environment. If more than one person wishes to study a given data set, then they either have to share a single interface or work individually and then confer on their views. This can prove to be quite cumbersome, especially if the users are situated at different locations. If both the users could share their virtual environment and data with each other and interact with each other in real-time then they will be able to confer on their analysis more interactively, and thus avoid any unnecessary procedures.

Tele-Immersion (TI) is defined as the integration of audio and video conferencing, via image based modeling, with collaborative virtual reality (CVR) in the context of data mining and significant computation. TI differs from CVEs as it encompasses a wider range on scientific computation and data mining technology. Tele-immersive visualization environments allow users to collaboratively analyze data; digital audio/video may be streamed between the sites to allow the participants in the environment to talk to each other. A wide range of visualization applications can be based on tele-immersive environments, to give their end users a better interface for collaboration and interaction both with remote users as well as the data.

Developing tele-immersive data visualization applications for large data involves merging the fields of networking, visualization and virtual reality. This poses a new challenge to the application developer, for the application needs to accommodate a wide range of technologies. The application should allow the user to collaborate with remote participants when performing the visualization task concurrently. Any changes that a collaborator makes in the virtual environment should be visible to the user. Besides leveraging collaboration the application should allow the user to visualize the contents of a dataset that is larger than the

memory (virtual, physical and disk) capacity of the users workstation. The focus of my dissertation is to bridge the gap between these technologies by proposing and implementing a framework that resolves most of the design issues for building such applications.

Many of the existing visualization systems have been extended to incorporate some of the features mentioned above, however these extensions are often specific to the particular application domain. The following chapter delves deeper into such systems, to identify common traits and shortcomings. Some good features of existing systems are incorporated in the TIDE architecture along with additional features. A distributed Client/Server architecture is proposed and its framework and implementation is described.

Data visualization is a technology that has been evolving for quite some time now. This chapter is a survey of the current data visualization systems. There exist several commercially available scientific visualization systems for two-dimensional desktop environments. Currently there does not exist a single commercial visualization application, which has been tailored for Tele-immersive environments. There is ongoing research in the development of visualization applications that use VR systems for display and interaction. Some systems focus on visualizing the end results of computational simulations, like visualization of the results of computational fluid dynamics (CFD) simulations or tornado simulations. Another class of applications not only allows the user to see the end results of simulations but also enable the user to steer and control a simulation in progress on remote supercomputers. Regardless of their goal and approach each of these systems contribute something to the field of Scientific Visualization.

In the sections that follow a number of these systems are described in detail, starting with the commercially available non-VR systems followed by VR systems being researched at various universities. Their approach to solving the problem of visualization is studied and their advantages and shortcomings are enumerated. A brief introduction to the approach taken by TIDE is given and a comparison is made with the existing systems to bring out their differences, similarities and capabilities.

Commercial problem solving environments for data visualization have existed since the late 1980s. These are systems like IRIS Explorer[5], Khoros[6], IBM Data Explorer[8], and AVS[9]. SCIRun[22], which is a scientific programming environment for Computational Steering is one recent application that allows the construction, debugging and steering of large-scale scientific computations. MineSet[23], an integrated suite of software tools for data mining and data visualization developed by Silicon Graphics is yet another data visualization system.

Research on utilizing virtual environments for data visualization has gained momentum since early 1990s. One of the early systems is the Virtual Wind Tunnel[12], which was created at the NASA Ames Research Center in 1992. Some of the recently developed systems are CAVEvis, a visualization tool for interactively exploring large time-varying three-dimensional scalar and vector fields developed at the National Center for Supercomputing Applications[13], and CAVEstudy a system that enables scientists to interactively steer a simulation from a virtual reality environment developed by Vrije University, Amsterdam. Of the above-mentioned systems only CAVEvis is immersive and can handle large data. None of these systems are collaborative. Immersive VR technology is being widely used to aide geoscientists in interpreting and analyzing three-dimensional geophysical and geological data. Such an application that visualizes geoscience data in an immersive environment is described in [25].

Very little research has been done in the development of tele-immersive applications that can handle large data. This could primarily be because totally immersive systems such as the

CAVE were developed only in 1993, whereas most of the visualization systems were developed in the late 1980s and early 1990s. Additionally VR requires three-dimensional interfaces, and these interfaces are currently being researched. Menu systems for two-dimensional interfaces have been well defined over the years, whereas three-dimensional widgets are not well defined.

As immersive systems gain popularity and well-defined 3D widgets for VR interfaces emerge along with the availability of high bandwidth good Quality of Service (QoS) networks and the prevalence of computational simulations that generate tera-bytes of data; the trend will be towards the development of tele-immersive applications that can visualize large massive data sets.

Different approaches have been taken to visualize data collected from various sources such as medical scans, simulations and satellites, etc. Some visualization systems, are based on a data flow model that allow the user to define a pipeline of filters/transformations that the data goes through before being rendered on a two-dimensional device. AVS, IBM Data Explorer, Khoros, IRIS Explorer and SCIRun fall into this category. These are non-VR systems. Mineset is another non-VR system that provides visual tools for data mining algorithms, applicable to large multidimensional data sets.

Visualization systems that use the VR technology are more dedicated towards solving a particular problem domain. The Virtual Wind Tunnel focuses on visualization of the results of CFD simulations. CAVEstudy is an immersive computational steering system, and CAVEvis is an immersive system for visualization of tornado simulations.

In the following sections each of the aforementioned systems is described in detail. The approach taken by each system to solve the problem of data visualization, method of interaction with the system, their benefits and shortcomings are discussed. Internal architectures of the systems are also described where necessary.

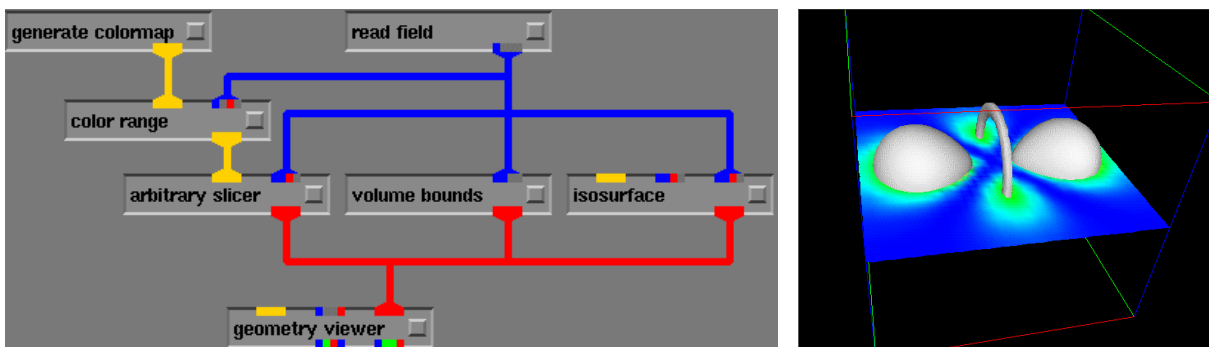
2.2.1 Earlier Problem Solving Environments (PSEs)

This section will describe PSEs like IRIS Explorer[5], Khoros[6], IBM Data Explorer[8], AVS[9] and SCIRun[21]. These systems are collectively discussed here, as they have a similar approach towards data visualization. They provide the user with a collection of modules, the user then uses visual programming techniques to connect these modules together into a visualization network; data is read in by the initial module in the network, each subsequent module in the pipeline acts on the data transforming it, till the final module, which is generally a rendering module, renders it on a display device.

Modules are represented on a two-dimensional screen as objects that can be manipulated via a familiar point-and-click interface. The modules are connected such that the output of one module is connected into the input of another. These connections define the path through which the data flows through the system. Each module is a routine that operates on its input data to produce some output. The user can set parameters for these routines through standard widgets, to define the behavior of the module; examples of typical parameters are threshold value for which an isosurface has to be calculated or the percentage decimation value for a decimation module. The first module can either read data streamed from a computational simulation on a socket or can import data from a file. The final module can either be a rendering module, which converts the data into an image to be generated on a two-

dimensional display device or a module, which exports the data to a file. Figure 1(a) below shows the visualization network for generating an isosurface and a slice from input data, Figure 1(b) shows the resulting visualization.

IBM Data Explorer[8] has a client/server architecture; the client is the GUI and the server process operates as a computational engine. The server accepts a well-defined protocol (a scripting language), which is generated by the interface. The server is controlled by a data flow executive, which determines what tasks need to be executed based on user requests and schedules their execution. The executive can be operated independently of the user interface via the scripting language.



An advantage of these systems is their extensibility; users can extend the system by supplementing the set of data types that can be processed by the modules. New types can be defined (by deriving from existing types) for handling data structures that could not be fitted into existing types, new modules can then be written to process this data. Hence these systems can easily be adapted to new applications and data. These systems allow the user to interactively create applications using visual programming techniques, thus alleviating them

of the task of writing complex programs. Since these modular systems need to operate on the entire data, they are not well suited for visualization of large amounts of data.

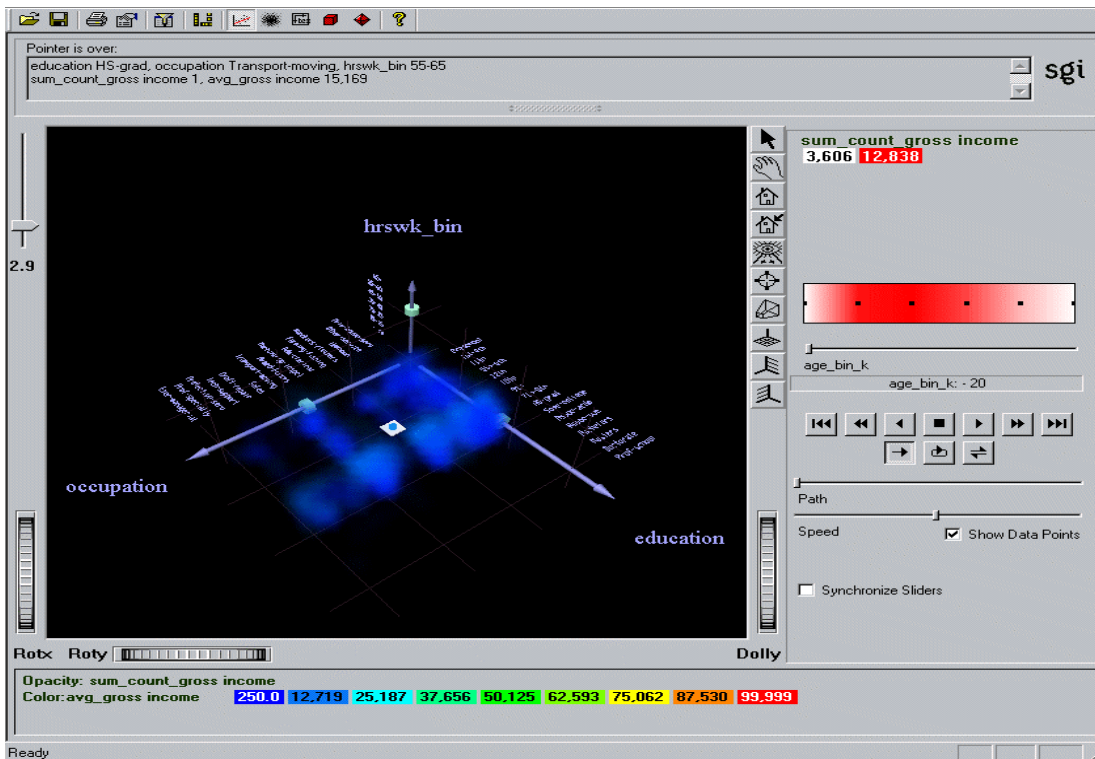
In addition to the functionalities provided by other systems SCIRun implements several methods to avoid the excessive memory use inherent in standard dataflow implementations. This inefficiency in the use of memory is due to the implementation of the data flow paradigm and not in the data flow model. Most dataflow systems maintain a copy of the dataset at each stage in the dataflow network, this will cause excessive thrashing when data sets are large and hence will result in poor response time. SCIRun overcomes this by allowing multiple modules to have shared pointers to common data sets or by letting the user decide which intermediate data sets are to be retained. SCIRun also allows interactive steering of the design and computation phases of a simulation.

2.2.2 MineSet

Operational data generated by business transactions is consolidated in a data warehouse, which often is a Relational Database Management System (RDBMS). The analysis of this data to find relevant information i.e. patterns in the data that will support future business decisions, is called data mining[19]. To gain a deeper understanding of the data, Silicon Graphics developed MineSet, which merges data visualization technology with data mining, by providing an integrated suite of software tools for data mining and data visualization. MineSet provides users with a rich set of visual tools for faster discovery of meaningful trends and relationships.

Underlying the interfaces to the visual tools, are analytic data mining algorithms that build comprehensive data models for analysis. Data mining results and visualizations can be easily deployed across corporate networks, and are available to users through point and click access. Analytical models developed in MineSet can be applied to any data set with the touch of a button. MineSet has a distributed, client/server architecture where in a MineSet client can connect to several MineSet servers thus allowing for explosive data growth.

Though MineSet has been developed as an integrated tool for data mining for large corporate data warehouses, it has the same goals as any other data visualization application for massive data sets: to gain a deeper insight into large data sets. Data mining algorithms find trends and



relationships in the data, which is complemented by the data visualization that utilizes the visual bandwidth to help the user gain a deeper, intuitive understanding of the data. Even

though MineSet produces information rich visualizations, these are still visualized on two-dimensional screens. Figure 2 shows the interface provided by MineSet and a splat visualizer view of a three-dimensional landscape from a sample dataset mapped to axes, sliders, color and opacity.

2.2.3 Virtual Reality Systems

Recent research in data visualization has focused on the use of virtual environments as an environment for data exploration and visualization. This has led to the development of a new class of applications/frameworks for data visualization. These systems have a more focused application domain for the purpose of high performance. The Virtual Wind Tunnel[12] and CAVEvis[13] are virtual reality based scientific visualization applications. The Cosmic Worm[11] and CAVEStudy[14] are systems that enable scientists to steer and control a computational simulation from a virtual environment such as the CAVE.

2.2.3.1 Virtual Wind Tunnel

The Virtual Wind Tunnel created at the NASA Ames Research Center in 1992, is an application designed to study Computational Fluid Dynamics (CFD) Simulations. The VR system consists of the BOOM (Binocular Omni-Oriented Monitor) as the display device and the VPL Data Glove for input. The wind tunnel generates visualizations, from scalar and vector data, such as streamlines, streaklines, particle streams, isosurfaces and contours. Visualizations in the virtual wind tunnel are associated with points in space. This allows a direct manipulation paradigm to be applied to the control of the visualizations. These points in space are controlled with visualization control tools, such as emitters. The user can move

these tools around in the wind tunnel. The use of these tools allows the user to move groups of visualizations about at the same time.

The Virtual wind tunnel is designed to support both high rendering rates as well as large amounts of computations. The client-server distributed architecture has a graphics process group executing the draw function of environment objects and the computation process group executing the compute functions, both operating asynchronously from each other. The graphics process groups are on a client system and the computational process group is on a server. The virtual wind tunnel can be applied to visualize scalar and vector fields, for example to visualize velocity vector fields and density vector fields around aircrafts.

In a BOOM, screens and optical system are housed in a box that is attached to a multi-link arm. The user looks into the box through two holes, sees the virtual world, and can guide the box to any position within the operational volume of the device. This might encumber the visualization process for some users. The Virtual Wind Tunnel is also limited in its application domain to the visualization of scalar and vector fields and is not easily extensible.

2.2.3.2 CAVEvis

CAVEvis is very similar to the Virtual Wind Tunnel, in principle. It also is a visualization tool for interactively exploring large time-varying three-dimensional scalar and vector fields and utilizes a virtual environment. CAVEvis utilizes the CAVE as the VR system and unlike the BOOM, the stereoscopic images are projected onto screens that surround one or multiple simultaneous users. The users wear lightweight LCD shutter glasses to see in 3D. The user uses a wand, as an input device. Trackers are attached to the glasses and the wand to track their position and orientation, in the virtual space. The object space is divided into a user

space and domain space. 3D-menus and other widgets are drawn in user space, where as the actual visualizations and associated direct manipulation tools are drawn in domain space. Objects in the user space are always relative to the user, and are always in sight.

Like Virtual Wind Tunnel, CAVEvis also has a distributed architecture, however CAVEvis focuses only on rendering tasks and user interface functionality. It interacts with a Functional Module (FM), which is a server to handle CAVEvis requests. Both CAVEvis and the FM operate independently of each other. CAVEvis is only aware of the domain time and space and the visualization objects that need to be rendered in the current time frame. It makes requests to the FM based on user input and the FM merely fulfils one request after another.

Though CAVEvis uses immersive VR systems, it does not make use of networking capabilities to link remote users for collaborative analysis sessions, it does support large data sets though.

2.2.3.3 Cosmic Worm & CAVestudy

High-speed networks and powerful graphics workstations make it possible to directly interact with scientific simulations running on massively parallel computers. Virtual reality systems can be used to not only see the end results of a computational simulation, but also to control such a simulation running on a remote computer. Some desktop scientific visualization applications already offer the capability to steer a remote simulation(SCIRun[22]). Research has been going on to provide the same functionality from within a virtual environment. The Cosmic Worm[11] is one such application developed to allow research scientists in NCSA's

astrophysics group to study cosmic behavior. CAVEstudy[14] is yet another system that allows immersive and interactive analysis of a simulation running on a remote computer.

In the Cosmic Worm interaction with the remote simulation is accomplished via a menu system invoked with the wand. The simulation can be stopped/paused at anytime, the user can see a visual representation of the latest time step, and can examine accumulated time steps. The user can modify the input parameters of a stopped simulation and restart it.

CAVEstudy is a more extensible system, in that it allows the user to describe the simulation by a description file, from which a server component for the simulation is generated and a server-proxy is generated for integration with the VR framework. The interface for the server allows one to start, stop, pause and resume the simulation. The server-proxy forwards the commands and input values to the server through the network and manages incoming data from the simulation. CAVEstudy has been used in the development of several applications such as Interactive Soccer, Diode laser simulation and molecular dynamics.

These systems are affected by the latency of the simulation program and the delays caused by the underlying network. Simulations are generally run on massively parallel computers such as CM-5, IBM's SP1 and SP2, etc. Computational steering systems such as these broaden the scope of applications of immersive environments.

2.2.3.4 Visualization Systems for Geoscience Data

The number of visualization centers in the oil and gas industry that use VR technology have increased from 2 in 1997, to more than 20 in the recent years. This increase in the use of VR systems is due to the immense utility of these systems. VR systems reduce the amount of

time required in analysis of the data and help in the rapid detection of significant gas and oil reserves.

A number of VR applications for geoscience visualization have been developed, some of which are commercially available. These systems include a multitude of data exploration and analysis tasks such as engineering reservoirs, plant walk through, viewing seismic slices etc. One such application is described and evaluated in [25]. The application pre-processes the data and converts it into a format (Open Inventor) that can be understood by the application. The VR system consists of a CAVE and three sensors are used to track the head and both the hands of the user. A six-degree-of-freedom input device such as the wand is used for user interaction. Three-dimensional seismic data sets comprise of a regularly spaced orthogonal volumes of data samples. This application provides a set of VR tools to select and manipulate an object. Constraint based virtual tools are used for selection and defining regions of interest and input parameters.

These systems provide an effective visualization tool for geoscience data, however they are not collaborative. A standard paradigm for the rapid development of three-dimensional widgets like the two-dimensional Windows, Icons, Mouse and Pointer (WIMP) user interface for conventional computers does not exist, and is still a field of research. Analysis and management of large data is still a challenge to geoscience visualization.

The systems described here have different application domains and they also differ in their approach to data visualization. The systems can be classified based on their application domain, the environment used for visualization and their approach to visualization. Based on

application domain we have systems like IRIS Explorer, Khoros, IBM Data Explorer, AVS, CAVEvis, Virtual Wind Tunnel, Cosmic Worm and SCIRun, which focus on visualization of data generated as a result of scientific simulations and digital instrumentation systems. MineSet on the other hand focuses on visualization of corporate data using data mining algorithms. Of the scientific visualization systems, IRIS Explorer, Khoros, IBM Data Explorer, AVS and SCIRun allow the user to use visual programming techniques to build visualization applications, these are termed as modular systems. These systems are highly extensible, and can be applied to visualize a wide variety of scientific engineering and graphics data. CAVEvis and Virtual Wind Tunnel focus more on visualization of time varying scalar and vector data resulting from CFD simulations and tornado experiments. SCIRun, Cosmic Worm and CAVEstudy allow the user to not only visualize scientific data but also control simulations running on remote supercomputers.

IRIS Explorer, Khoros, IBM Data Explorer, AVS, MineSet and SCIRun are primarily desktop systems. They use traditional two-dimensional screens for interaction with the user and rendering of the final visualization. Virtual Wind Tunnel, Cosmic Worm, CAVEstudy and CAVEvis are designed to utilize VR display devices. Of these only Virtual Wind Tunnel uses a non-immersive virtual environment, the rest use immersive environments for user interaction and display.

The modular systems follow an approach wherein each of the modules can be assigned to different processes, which may be assigned to different processors for better performance. The virtual reality based systems use a common approach where, a distributed Client/Server architecture is implemented. A visualization client handles the rendering, and the server handles the computational tasks. The server handles client requests; both the client and the

server can operate asynchronously. This takes the onus of performing intense computations from client, which can now focus on rendering at an immersive frame rate. The TIDE framework implements a similar distributed architecture.

Though the systems described here, have different architectures and interaction methods, they all have a common goal: gain insight into data, which is collected from various sources such as medical scans, computational simulations, scientific experiments and even statistical data. Analysis of data can provide valuable information regarding the correctness of the data and the trends and relationships in the data. This information can even decide the future course of action that needs to be taken.

One problem that the data visualization community faces is the visualization of large data. Simulation results today can surpass 100Gbytes easily. Not all systems are designed to handle large data, systems which require to read all the data into main memory (in core) cannot be utilized to visualize 100GB of data. Large data sets can easily overwhelm the physical and virtual memory of the visualization system. A new approach to data visualization is needed.

In some cases it is possible to segment the data based on its spatial, temporal or logical characteristics, and load only required segments into main memory. However the segments themselves could be too large to fit into the operating systems virtual memory or they may lead to excessive thrashing. Hence such applications need to define their own memory management techniques instead of wholly relying on the operating system[20].

Hence if the data is very large, in the range of several hundred Megabytes to a few Terabytes, it is only feasible to visualize a small subset of the data that can fit into main memory. Of the systems described here only CAVEvis and SCIRun, mention being able to visualize large data. MineSet handles large data, by having data distributed over a number of servers and clients can query these servers for data, which they are interested in. Another approach to visualize large data is to have multi resolution versions of the data. A user is presented with a coarse low-resolution version of the data and can then focus on particular regions of the data to see finer resolution versions. If the size of the region that the user is interested in is still large, the higher resolution version can still exceed the memory limits. Large data visualization is still an active research topic, and no perfect solution is available to the problem.

Most of the current visualization systems (immersive or otherwise) allow a single user to interact with the data at a given point in time. Research is generally a group activity wherein a number of scientists bring different skill sets to the table. If a group of research scientists wish to collaborate during a data visualization session they would have to share one interface to interact with the data; this can prove to be quite cumbersome and inconvenient especially if the people involved are geographically at different co-ordinates.

Availability of high bandwidth low latency networks makes collaborative visualization a possibility. Collaborative visualization applications tend to be complex, because of the presence of multiple users. Since the users interact in a collaborative context, they need to operate synchronously with each other and for this it is necessary that they operate in a shared space and are aware of the activities of other users in the environment. Collaborative Visualization applications have great potential in that expertise of a particular user can be

shared without having to relocate the user from their geographic position and users can collaborate on the visualization. Having realized this, collaborative extensions[21] were added to AVS, and IRIS Explorer[3]. In Collaborative AVS data may move from one module to another in the same AVS network or to a module in an AVS network on a different computer by means of a collaboration module. This allows users to access a central data set or multiple local copies, and for multiple users to interactively share any visualization parameters. In IRIS Explorer, too, each user can build their own network, but has the opportunity to make a connection to a collaboratively aware module that can pass data to and from other collaborators' network.

2.5.1 Data types

Visualization techniques are dependent on the dimensions of the domain of the quantity being visualized and the type of the quantity i.e. scalar or vector. Time dependency adds another dimension to the domain. Scientific data generally consists of 3D scalar and vector fields. The type and the dimensionality of the data being visualized largely determines the requirements of the scientific visualization method. The complexity of geometric modeling algorithms increases if there are fewer restrictions on the data and/or if the dimensionality of the data increases.

2.5.2 Data Visualization

To gain better insight into data, it is important that the visualization is an accurate representation of the data. “How” the data is visually represented determines the effectiveness of the visualization experience, hence the geometric models derived from the data should be comprehensible and should utilize appropriate visual cues to aide the user in

finding trends and relationships in the data. For the visualization of 3D scalar data generally geometric modeling algorithms generate scalar glyphs, isosurfaces, slices and orthogonal slices. 3D vector data can be visualized using 3D vector glyphs, streamlines, streaklines (if the vector quantity is time varying) and particle advection. Most modular visualization systems described, support visualization of all types of data. These systems also provide imaging algorithms for the visualization of 2D images. CAVEvis and the Virtual Wind Tunnel visualize 3D scalar and vector fields. TIDE currently supports 3D scalar data and generates scalar glyphs from the data, the color, opacity, dimension and position of the glyphs in space are mapped to different attributes. The TIDE framework is extendible to support other visualization techniques and data types.

2.5.3 Data Storage and Retrieval

Different formats exist for the archiving data on high capacity disks. In some cases the data may be pre-processed and converted into a format understood by the visualization system for example octree-decomposition techniques are used to decompose the data into a spatial hierarchy, and the data is then stored to disk. Hierarchical data formats allow users to retrieve a subset of the data based on spatial or temporal proximity, most of these data archiving systems have their own API for data retrieval[26]. The Relational Database Management System is also used for storing scientific data. Data can be retrieved from the archive using a Structured Query Language (SQL).

Applications that merge immersive environments (which inherently allow participants to collaborate with each other) with data visualization, can give users a ‘true’ sense of presence in an environment with remote users and visualization tools and objects. This is the research

goal of TIDE. The TIDE architecture implements some of the good features, which exist in the current systems, and provides additional capabilities, which overcome the shortcomings of the existing systems. It retains the client/server architecture that is found in many of the immersive systems. Unlike existing collaborative systems[3] [21] that have added extensions for collaboration, TIDE focuses on making interaction and collaboration an inherent part of the architecture. Applications of TIDE have basic inbuilt collaborative capabilities, which can be further enhanced as per application needs.

Y	Y	N	Y	Y	Wide
Y	Y	Y	Y	Y	Wide
Y	Y	N	N	Y	Wide
Y	N	N	N	Y	Wide
Y	N	N	Y	Y	Wide
Y	N	Y	N	Y	Computational Steering
Y	N	Y	Y	Y	Time varying scalar and vector fields
Y	N	Y	N	N	Scalar and Vector CFD Data
Y	N	Y	Y	N	Computational Steering
N	N	N	N	Y	Data Mining and Visualization
N	Y	Y	N	N	Limited
N	N	N	N	Y	Wide
N	Y	Y	N	N	Time varying volumetric data
Y	Y	Y	Y	Y	Wide

TIDE has a command driven architecture, wherein a client communicates with the server based on an application specific protocol. This implementation is similar in concept to a Remote Procedure Call (RPC) mechanism. The client sends command requests to the server and the server implements handlers that execute these requests. To handle large data sets, the

client can query for a subset of the data and can further refine its queries to get a higher resolution version of subset.

Figure 3. gives a better picture of the capabilities of TIDE in comparison to other existing systems. TIDE merges immersive technology with data mining and computation, for a new genre of tele-immersive applications for data exploration. Details of the TIDE architecture and one of its applications, is discussed in the following chapters.

The research focus of TIDE is to provide a skeleton for developing applications for data exploration and analysis from tele-immersive environments (and study its effectiveness??). Merging immersive VR technology with massive data visualization will pave the way for a new class of applications that can effectively use VR devices for visualization of large data sets, (and give more insight to the user). The TIDE framework defines the architecture of such applications.

The TIDE framework allows groups of scientists each at a geographically disparate location to collectively participate in a data analysis session, in a virtual environment. The data being analyzed can be stored on data servers, which are at a different location from the clients'

Imagine a scenario where there are three scientists who wish to confer on the effects of ocean currents on the earth's climate. However all the three scientists are located at geographically disparate locations and so is the data. One of the scientists is an oceanographer, all the oceanographic data is situated in his laboratory. Another scientist is a climatic expert, and has the climate data in his laboratory and the third is a geoscientist. The climate data is defined by attributes such as temperature, precipitation and pressure. Whereas the oceanographic data has attributes such as salinity, direction and strength of the ocean current and sea surface

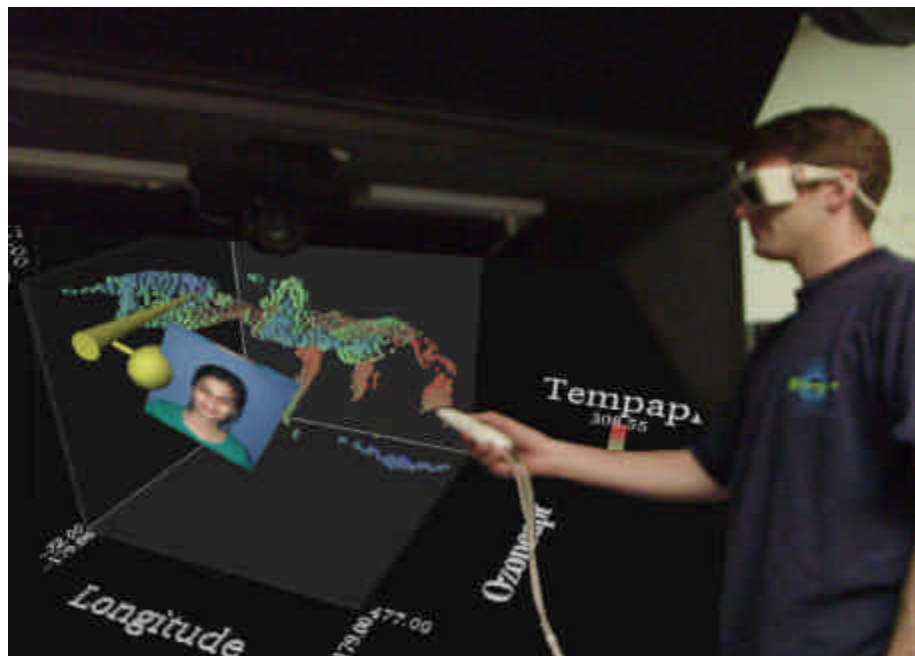
temperature. The scientists decide to participate in a collaborative data exploration and analysis session. Each scientist can access both data sets.

The scientists visualize and interact with the data using a device such as the CAVE or an ImmersaDesk. A virtual environment is created wherein each scientist can see the representation of the remote scientists in his own environment. This representation is called an avatar, which is a three-dimensional model that may constitute of various parts of the human body. The position and orientation of each user in the virtual environment is tracked and his/her remote representation is updated accordingly. So if the oceanographer points to a certain location in the data-set, the climatic expert and the geoscientist will see his avatar pointing to that location in their own world. Digital audio is set up between the sites to allow the participants to speak to each other. The scientists can work synchronously, in collaboration with each other or asynchronously, in a standalone mode. Consider a synchronous session where all the participants in the session share the visual representation of the data. The scientists can interact with the data using three-dimensional tools.

The oceanographer decides to play the lead role and loads the atmospheric data and chooses to visualize the temperature over the earth's surface. The climatic expert notices an anomaly in the surface temperature over Europe. However the oceanographer knows the reason behind this are the strong currents that flow from the hot tropical regions towards the poles and loads the ocean current information from the database. Now the scientists can see the ocean currents that are displayed as vectors, the magnitude and direction of which depend on the strength and direction of the current. The oceanographer correlates this with the temperature information and gives a detailed explanation of the anomaly in the temperature of the European region. During his discourse, the oceanographer can rotate or scale the data

set to focus on certain aspects of the visualization. He/she can also navigate to a particular point in the virtual space. For example say if he/she wants to talk by keeping both the climatic expert and the geoscientist in view then he/she can navigate to a suitable position where both the remote participants and the data are within his/her field of view.

Now that the geoscientist, has gained enough knowledge of the effects of the ocean currents, he/she decides to work asynchronously and leaves the session, the other collaborators see the geoscientist's avatar exit the virtual space. After this point, no information is shared between



the geoscientist and the rest of the participants. He/she then explores some other region specific data and finds similar effects of the ocean currents on the climate. The climatic expert and the geoscientist, could access Dr. A's expertise even though they were not in the same physical space. Neither of the participants needed to have the data being visualized stored locally. Figure 4. shows two clients collaborating with each other.

The scalability and extensibility of an application largely depends on the design. Designing Object Oriented Software is a difficult task, as one needs to identify relevant objects in the design and define their class interface and the relationships between different classes, such that they function together to solve a particular problem. Designing reusable Object Oriented Software is even more difficult as the designer may not be aware of the components of future applications or their behavior. The task then is to identify reusable design components from applications already developed to solve related problem domains and define the behavior of those for future use. This is what TIDE tries to achieve.

Over the years trends and patterns have evolved in the structure and relationships between objects (classes). These Design Patterns[24] can be reapplied to solve similar problems in other application domains. Knowing that a design pattern exists to solve a particular problem, allows the designer to reuse them in the context of the application domain. TIDE uses design patterns wherever applicable this makes the framework more reusable.

A Framework constitutes of a set of interrelated abstract classes that define the skeleton for a particular class of applications. These set of classes characterize the design of applications developed to solve a particular problem domain. Frameworks are developed to facilitate design reuse rather than code reuse. An application developer provides application specific subclasses for the abstract classes. A good framework is one that can be understood easily and provides enough features that are useful to the application developer and is generic enough to be used in a variety of applications. Such a framework will effectively reduce the

cost involved in designing the application, and more attention can be given to application specific details.

Frameworks describe the basic approach that needs to be taken to solve a particular problem and provides hooks where application specific code can be added. Frameworks evolve as more applications are developed based on them. Some components of the framework can be discarded or modified if they do not contribute towards an effective reusable design. New components can be identified and added with time. Writing a good framework can prove to be a daunting task as it is not possible to know the requirements and structure of all possible applications.

There are some constituent parts, which any tele-immersive application needs to be composed of, irrespective of its application domain. Some of the more prominent ones are identified here.

3.4.1 Data Visualization

Data that needs to be analyzed does not have one uniform format; the type of the data depends largely on its source. For example CFD simulations may generate data that is comprised of an unstructured grid of scalar and vector fields. Whereas data collected from remote sensing satellites may be in the form of a regular grid of scalar fields. A geometrical representation of this data needs to be generated for visualization purposes, the data may be processed before this transformation i.e. it may be passed through filters for decimation and isosurface extraction. These operations can be time consuming and may even take several hours to compute.

Three-dimensional input tools need to be provided to allow the users to directly manipulate the data for example to slice through the data or specify the seed points for streamlines. Users should be allowed to indirectly manipulate the data i.e. to save the visualization to a file, to specify input parameters to transformation modules, etc. Any user interaction should be done in real time and the display should be refreshed at a frame rate higher than the minimum required to give the user a sense of immersion.

3.4.2. Handling large data

As it is feasible to visualize only a subset of the data at any given time, users should be allowed to specify accurately what portions of the data they are interested in. The data can be segmented spatially, temporally or logically. Multiple resolution versions of the data can be provided, and the user can start off by visualizing a low-resolution version and higher resolution versions are generated as the user narrows down on particular region of interest within the data.

3.4.3 Leveraging Collaboration

For a collaborative session to be effective the following issues need to be managed:

- All collaborating participants should have a sense of shared space i.e. they are immersed in the same virtual environment with similar characteristics and objects.
- Each participant should be aware of the presence of the other remote participants and should be able to interact and communicate with them in real-time, audio may be streamed between them to allow them to talk to each other. An avatar, a geometric representation of the remote user is used to represent a remote user in local space.
- If a user modifies any object in a shared space, be it the visualization, an object in the virtual environment or moves in the virtual space, then these state changes need to be

transmitted to all the participants in the environment, at the same time any state information received from remote participants should be applied to respective shared objects.

- Large delay in the transmission and reception of the information, caused by low bandwidth high latency networks or computational delays, degrades the effectiveness of the collaboration.

Data visualization involves the conversion of raw data into a geometric representation (polygons, cubes etc.) that can be rendered onto a display device as shown in

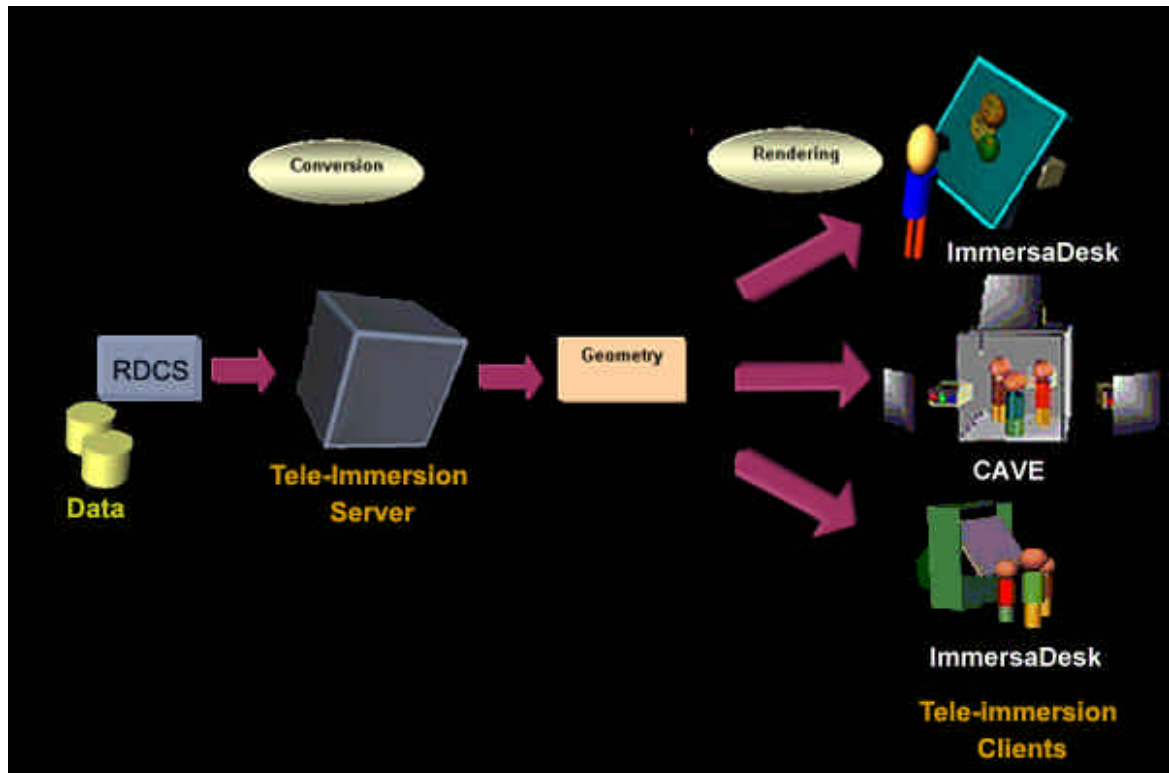
The is responsible for fetching the data from the data source, representing it in the computer's physical/virtual memory using a domain specific data model, performing operations such as feature extraction and decimation (if required) and finally generating a geometrical representation of the model. The is responsible for drawing the geometrical representation on a display device. Both these tasks when performed by a single application can prove to be computation intensive and would generally use up most of the resources such as memory, disk space and CPU cycles, especially for large complex data. This results in very low frame rates, unacceptable by immersive systems such as the CAVE.

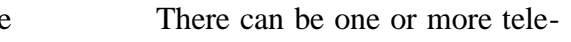
TIDE separates the conversion process from the rendering process by allocating the tasks to different computing environments. The conversion of the data, and the rendering of the visualization are done by separate applications. As shown in the visualization process is split up by implementing a Client/Server architecture. The server application does

the conversion and generates the geometry that is sent to the client, which renders it on a display device to be visualized by the user. In this way the conversion of the data into a three dimensional visualization can be done on a high end supercomputer (the tele-immersion server), that has the resources to carry out intensive computations and the geometrical model can be rendered on tele-immersive display systems such as the CAVE (tele-immersion client). In addition to this the server application also handles collaborative visualization sessions, where in you have a group of clients collectively analyzing/visualizing the data.

Splitting up the visualization process this way allows the server process to focus on the data conversion and manipulation and the client process on rendering and user interaction.

Thus the conceptual organization of TIDE consists of three primary components: the Remote data and computation services (RDCS); the Tele-Immersion Server (TIS); and one or more



Tele-Immersion Client (TIC). This is shown in the  There can be one or more tele-immersion clients, which connect to a central server called the Tele-immersion server. The Tele-immersion clients are the visualization end points that allow the user to participate in the collaborative virtual environment. The Tele-immersion server is in turn connected to one or more remote data and computation services, and mediates the interaction between the client and these services.

3.5.1 Remote Data and Computation Services

RDCS refer to external databases (or data mining servers) and/or simulations/compute-intensive tasks running on supercomputers or compute clusters. The databases hold data generated by computational simulations and digital instrumentation systems. Since the size of the data may vary from a few Megabytes to several Terabytes, the data may be distributed over several such nodes. Also it is impossible to visualize all the data, as it will not fit in core memory, only a subset on the data can be visualized at a time. Hence this data itself may be processed in such a way that from the entire data set, a smaller data set is extracted, which is a more coarse version of the original large data set, which can be rendered by the TIC at a desirable frame rate. Hence a trade-off is made between the resolution of the data set and better interactivity. The coarse version can be obtained by averaging, i.e. by replacing a set of values by their average and/or by decimating. In addition to this several attributes can be mapped to distinct visually perceptible cues such as stereoscopic depth, hue, size and opacity.

3.5.2 Tele-Immersion Server

The TIS mediates interaction between the TIC and the RDCS and also serves as a persistent entry point for the clients so that they may initiate long-running data-intensive queries and

come back at a later time to view their progress. The TIS handles multiple clients and synchronizes their interaction with the RDCS. The TIS allows each TIC connected to it to operate on its own local sub-set of the data. It is the TIS that actually retrieves the raw data from the RDCS and converts it into a three dimensional geometrical representation that can then be visualized by the TIC.

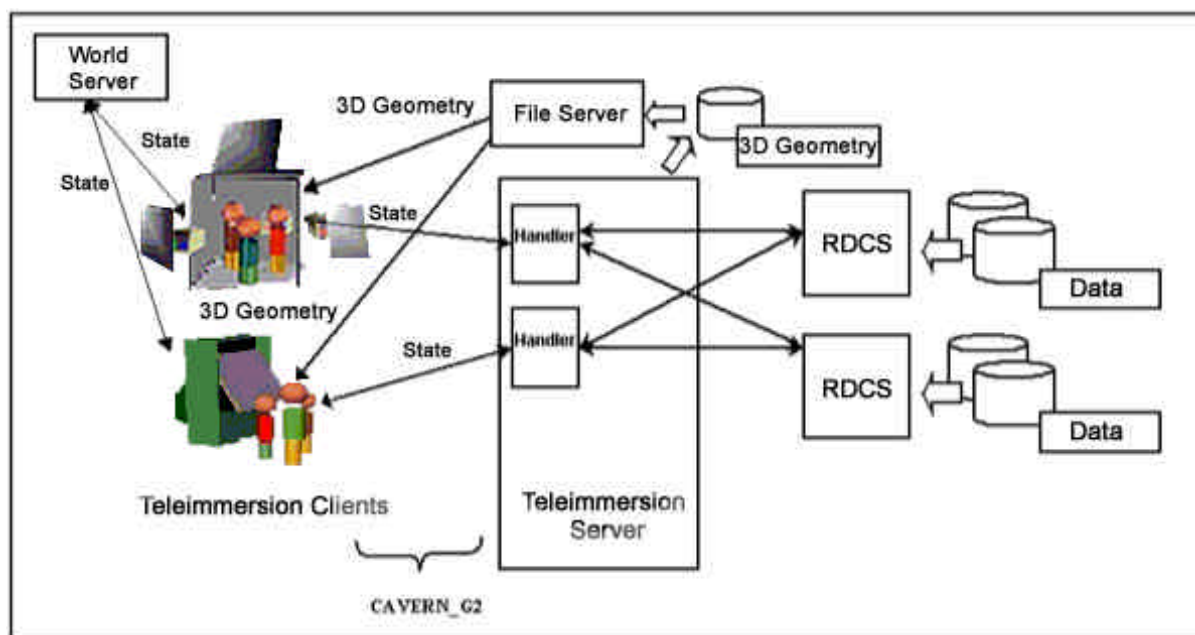
The TIS is designed such that each client has control over its own data. If the client is working synchronously with other participants in a session then the final geometric model can be broadcast to all the clients or if the client is working asynchronously then the results are sent only to that particular client. Inventor is used as the graphical description language for the geometrical model. Since Inventor's geometry information is consistent with that specified in VRML 1.0, the visualization can also be loaded into a VRML browser. The TIS uses an Inventor writer to create the file, another writer, which uses a graphics file format compatible with the client's, can easily replace this.

3.5.3 Tele-immersion Client

The Tele-immersion client (TIC) consists of the virtual reality display device (such as the CAVE, ImmersaDesk, PowerWall etc.) and the interface to allow collaborative retrieval and visualization of data. The TIC handles the rendering of the three dimensional visualization, which it receives from the TIS, on the VR display device. It allows the user to interact with the data and to specify any input parameters required for generating the next visualization step. The TIC also provides the user with the tools to directly manipulate the visual representation of the data. If the TIC is a participant in a collaborative session, then any changes made to the visualization such as rotation or scaling, are propagated to all the clients

and any such information received from the remote participants is applied to the its local model. In addition to this the TIC also handles the rendering of avatars to allow participants to collaborate with each other effectively.

The distributed client/server architecture of TIDE is as diagrammed in



3.6.1 The Tele-Immersion Client

The process of data visualization is one that involves several steps to reach the end rendered result. A user must first query a data archive for specific information. Filtering operations provide more specific details about the particular aspects of the data to visualize. These operations may include partitioning data, specifying correlations between different types of data, or other filtering methods. All of these specifications are consolidated to produce a rendered visual image. It is this rendered image that represents the data visually. A user

may, depending upon the visualization application, perform additional operations based on the visualization itself. These include manipulations like translations and rotations as well as data-related functions like zooming in on specific parts of the visualization.

TIDE has been designed to maintain a conceptual separation between these two modes of interaction: *query* pertains to communicating with data archives to specify and obtain information about the data; and *tele-immersive* is any interaction performed in the context of the tele-immersive virtual environment. These latter actions by the user could also communicate with the data servers. Information about actions in the tele-immersive environment need to be communicated to other users to indicate the user is performing an action. The query interface address communications relating to the data itself, while the visualization interface addresses interactions related to the shared environment and the visualization geometry.

The visualization environment utilized by TIDE is a shared virtual environment inhabited by multiple users. When collaborating in a virtual environment, the user should be able to interact with the remote participants. This can be done by streaming audio between the participants to allow them to talk to each other and by providing a representation of the remote user, an *avatar*, that is exactly reflects the users position in space. Depending on the tracking abilities of the VR system, the users position and/or orientation can be tracked by electromagnetic sensors. Any changes to the client's position and orientation also need to be broadcast to everyone in the environment. This is information that needs to be updated regularly at high frequencies. An environment server leverages the flow of this information

between various clients. A separate environment server is dedicated to this task, as long delays degrade the effectiveness of the collaborative experience.

After a query has been completed by the data servers, visualization geometry is sent to each of the connected clients. That geometry is loaded into the environment for viewing. The user can scale and rotate the visualization to get the correct view of the data, or to bring a particular region of interest into focus. There is a time interval between the moment that the user submits a query and the visual image is rendered on the screen. This delay depends on the amount of information queried, the available bandwidth of the network, the time taken to convert the data into a visual model, file download time and time taken for rendering the image. This time interval varies from a couple of milliseconds to a few seconds. For improving the performance of the system, the client starts the process of loading the new visualization in a new thread, which runs in the background. The old visualization is removed only after the new one is ready. This way the user does not have to stay idle for long intervals. Whenever any client changes the state of the visualization, this change is propagated to all the other clients participating in the collaboration.

Although the visualization has been reduced in complexity from the fidelity of the original data, the geometry can still be quite intricate. Complex geometry can hinder the performance of the interaction due to increased time between frame updates. To reduce the polygon count of the geometry during manipulation, level of detail features have been introduced. This switch to a significantly lower number of polygons makes the manipulation updates much more fluid in response to the user's actions. As soon as a manipulation begins, the geometry view switches to a lower-resolution bounding box. This switch takes place for all users in

the environment. It also serves to provide an indication that a user has initiated a manipulation operation on the geometry.

In order to provide additional status information about the current state of operations, the client uses audio cues. These cues indicate when a visualization download has begun. Additionally, a 3D watch icon indicates that the client is performing processing--in this case, background loading the geometry into the scene graph.

3.6.2 The Teleimmersion Server

The TIS abstracts the TIC from the actual data, the TIC need not be aware of how and where the data is stored. It only needs to specify to the server what data it is interested in and how it wants to visualize the data i.e. as a three-dimensional plot, extract an isosurface from the data or correlate several attributes by generating a histogram etc.

In this way the TIS server acts as a mediator decoupling the TIC from the source of data, ideally, allowing any client to visualize any data. The TIC can then concentrate on making the rendering process more efficient i.e. focus on tasks that improves the interactivity with the visualization and increase the frame rate.

The TIS is multiprocessed i.e. for every new client a separate server handler process is created. When a client connects it can specify what view of the data set it wants to visualize and the corresponding server process retrieves that data from the data server. Every handler process stores locally, the subset of the data being visualized by the client. The client can specify operations on its own subset of the data set like feature extraction, decimation, mapping attributes of the data set to various visual dimensions etc. The handler process

performs these operations on the data, converts the data into a three dimensional representation and notifies all the clients that new data is ready. The client then downloads the visualization from the server ().

The clients share only the geometric representation in a collaborative session, not the data. The novelty of the TIDE architecture lies here, where it decouples the data from its visualization, thus allowing them to be treated as separate entities. If a new client connects to an already existing session then the TIS automatically sends commands to the new client enabling it to visualize the current data being explored. To visualize another data set a client needs to submit a fresh query, the handler process again retrieves the necessary data from the data source.

Since every handler process has the data for its client, the client can perform multiple operations on this data, without having to query the data servers repeatedly. Huge amounts of data transfers lead to an increased delay in responding to the client. If the dataset that the client is interested in fits in to the physical memory of the TIS then the data can be retrieved once and different views can be generated off the same data, this improves the response time as the TIS does not have to execute expensive data retrieval operations always.

For example consider a scenario where there are two clients (A and B) visualizing an atmospheric data set, which is characterized by attributes such as wind velocity, precipitation, temperature and vegetation. Client A is analyzing the effect of precipitation on the vegetation and Client B is correlating the temperature to the wind velocity. Client B discovers an anomalous trend in the correlation and needs Client A's opinion to support his finding, hence he/she asks client A to participate in the analysis session. Client B then shares

his visualization with Client A. After discussing with client B, A can still go back to his own visualization by asking the TIS handler process to generate a visualization from his copy of the data, without having to re-query for the data.

A shared memory arena is used for communication between the various handler processes and the parent process. Communication between the TIC and TIS is established using CAVERNsoft G2, a C++ toolkit for building collaborative networked applications [1].

3.6.2.1 Co-Servers of the Teleimmersion Server

To efficiently handle the communication between various participants in the collaborative session, separate dedicated servers are allocated the tasks of handling exchange of shared information and file operations, namely the file server and the world server.

3.6.2.1.1 The World Server

Information about the immersive environment is communicated among multiple users through the TIDE world server. This server allows for collaboration between researchers examining a particular dataset. Each user has hand and head tracking data available from the CAVE library. This information is shared among users to provide the location information for each avatar within the environment. Whenever a manipulation is initiated from within the visualization, the other users need to be aware of that operation as well.

The TIDE world server is comprised of three primary components. First, a UDP reflector transmits UDP packets from one client to another. Second, when information being transmitted is essential, a TCP reflector transmits reliable TCP packets. Finally, a database component manages state information such as the transformations of the visualization

geometry. An identifier string identifies each of these manipulations, this allows for multiple manipulations within the environment.

The CAVE library handles the tracking of the user's position and orientation. This information is transmitted during each frame to the TIDE world server. The high frequency of the transmission does not require a very reliable service. Therefore, a UDP connection is made between each client and the server. During each frame update, the client packages the position and orientation of the user's hand and head, and sends them via a UDP connection to the server. The server reflects this information to each of the other connected users in the environment. Each client in turn updates the avatar for the particular user.


Client commands involving direct manipulation of the interface are signaled with reliable TCP transmissions, as any loss of this information could lead to misunderstandings in the communication between participants. For example, when a user begins to rotate an object, the client sends a rotation initialization command to the server. When this command is reflected to the other remote clients, they each switch their views to the bounding box representation of the geometry. After manipulation is completed, the client sends a rotation complete command. Each remote client receives the reflected command from the server and switches the visualization back to the full-resolution geometry. In the time duration between the beginning and the end of a rotation the client sends state information that reflects the amount/degree of rotation to the remote clients, which use this information to rotate their view.

The TIDE world server's primary function is to serve as a central connection point for each of the clients. The server reflects commands and operations to each of the connected clients to enforce the shared states of the environment.

3.6.2.1.2 The File Server

A dedicated file server allows multiple clients to download the geometry files generated by the TIS. This improves the performance of the TIS as it does not have to deal with uploads to all the clients, additionally the architecture can be further modified to allow the clients to download the files only if they wish to.

The file server is based on CAVERNSofts remote file I/O networking class.

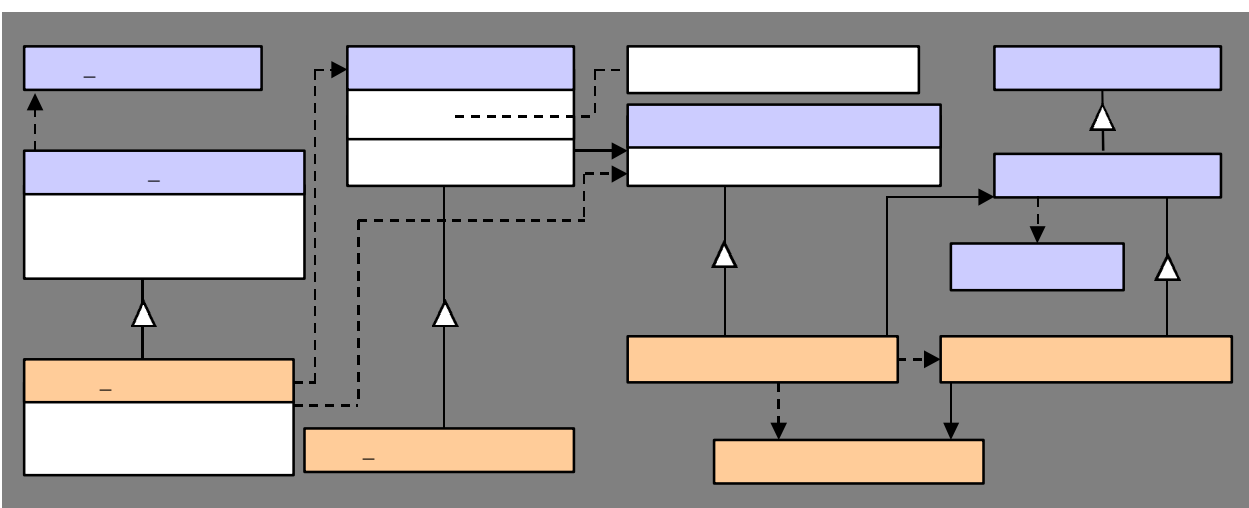
TIDE provides an application developer with the infrastructure to build collaborative tele-immersive applications to explore large multi-dimensional data sets. TIDE has a distributed command driven architecture as shown in . A message-passing interface is used for exchanging information between the TIS and the TIC over the network. A typical application of TIDE would consist of a number of TICs on various VR display devices requesting services from the central TIS. The TIDE framework defines how the client handles collaboration, user interaction, rendering of the visualization and querying the TIS for information. It specifies how the TIS handles: multiple requests from multiple clients, data representation and conversion. Communication with the TIC is established using a reliable TCP channel.

The implementation of the framework is done in C++ and effectively uses design patterns[24] such as Template method, Command, Observer and Mediator.

3.7.1 The Tele-Immersion Server

It is not possible to define every operation that the TIS needs to carry out, as some tasks are application specific. However, some key tasks remain common to all applications and these are: handling multiple requests from multiple clients, data representation and conversion. “What” the client requests are and “how” they are to be handled may be application specific. Similarly “what” the data is and “how” the geometric modeling is done may depend on the application domain. The TIS provides a network of interrelated abstract classes that define the interface for handling client requests etc., the task of providing the application specific functionality is deferred to specialized sub-classes, which need to be implemented by the application developer.

shows the interrelationships between various classes using the UML notation. The classes belonging to the framework are blue in color. The Tide_Client_Manager maintains a client database for the Tide_Server. Every client request is packaged into a command object and a handler is implemented to execute one or more commands. For example if a client wishes to visualize a subset of the data then the TIS gets a message from the client that



specifies all the parameters required for data segmentation. The TIS implements a segmentation command to wrap the client request into an object and then identifies a handler which is aware of the data source and knows how to get the subset of the data the client is interested in. So every time a client asks for a data subset the segmentation command-handler pair is sufficient to handle the request. Command-Handler pairs can be created based on the abstract `Tide_BaseCommand` and `Tide_BaseCommandMediator` classes.

The framework also has an abstract converter class that specifies the algorithm for converting the data into a geometric model and sending the reply back to the client. Hooks are provided wherever application specific code needs to be added, this code is provided by concrete converters. The handlers that do geometric modeling use these converters. For example: if the client wishes to extract an isosurface based on a threshold value it sends a message to the TIS, the TIS sets the `isosurface_command` and `isosurface_handler` objects to execute the request, The `isosurface_handler` uses an `isosurface_converter` to generate the isosurface, the geometry information for which is then sent back to the requesting client. Currently the framework has a `Writer` object that generates the geometry as an Open Inventor file. Writers for other formats can easily replace this one.

3.7.1.1 Communication with the Clients

For every client that connects, the TIS forks a new handler process, which is responsible for carrying out any requests made by a client. However the type of requests made by a client, are application specific, hence all the tasks performed by the handler process cannot be defined a priori. A template method, `forkHandlerProcess`, is used to define the function associated with the handler process. This method defines the steps to: read incoming requests from the

TIC, handle those commands (a hook method `handleCommand()` is provided for this purpose) and send the reply to the main parent TIS process to be sent back to the client(s). A hook method `interpretReply()` is provided to allow concrete server classes to interpret the reply, and package it and send it to one or more clients.

The concrete sub classes can define the behavior that can vary by implementing these `handleCommand()` and the `interpretReply()` functions, which are called when a new client request arrives and before sending data back to the client (i.e. unmarshal/marshal the client request/reply). The `Tide_Server` class therefore provides the basic framework to handle multiple client connections and to send commands, state information and data to these clients. The TIC uses a reliable TCP connection to send commands to the server using a message-passing interface.

3.7.1.2 Handling Client Requests

The task of interpreting the client requests and performing the operations needed to carry them out is deferred to application specific concrete subclasses of `Tide_BaseCommand` and `Tide_BaseCommandMediator`, these classes are based on the Command pattern. The concrete subclasses understand the client requests and are aware of the operations to be performed to fulfill those requests. The TIDE framework provides a generic way in which requests can be coupled to handler objects that execute the request. It uses the Command pattern for this[24].

The `Tide_BaseCommand` provides the interface to wrap the client request into an object and stores the receiver of the request as an instance variable. Once this is done a client of this class invokes the command by calling the `execute()` function, which internally calls the

function of the handler (instances of Tide_BaseCommandMediator). Concrete subclasses of Tide_BaseCommandMediator override the operation and perform application specific tasks to handle the requests.

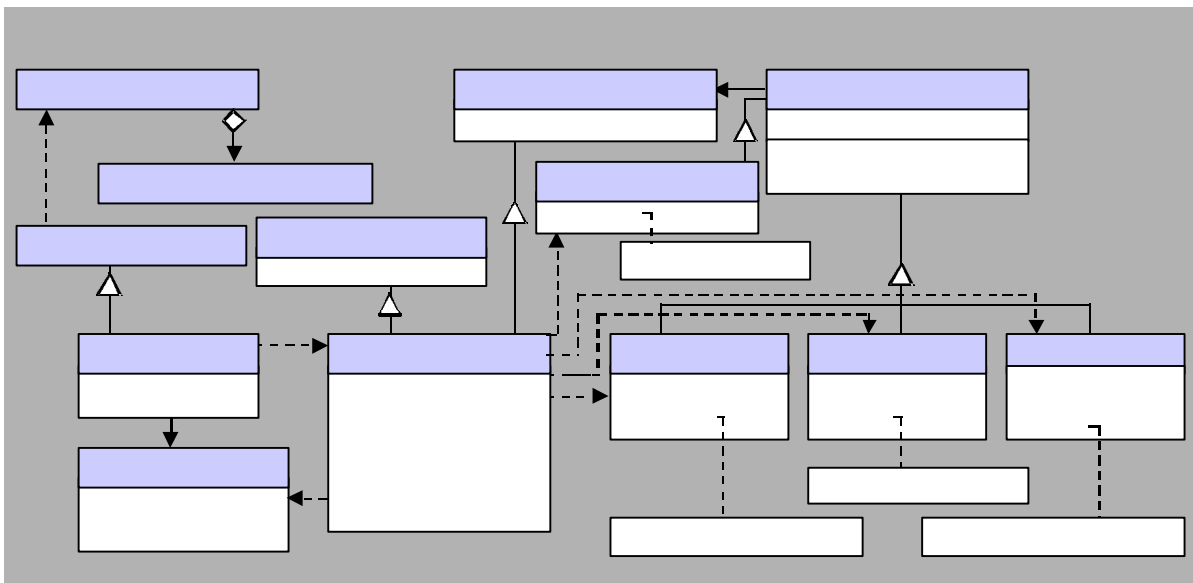
In this way an application can handle any user request by implementing

- A concrete subclass of Tide_BaseCommand that stores the state of the user request and implements (e.g. Tide_Concrete_Command)
- A concrete subclass of Tide_Base-CommandMediator that implements and has the knowledge to carry out the request (e.g. Tide_Concrete_CommandMediator)

The Tide_ConcreteServer class creates both the command and the corresponding mediator object and links them together. Thus it is easy for an application to add new commands, as it does not have to modify existing classes.

3.7.1.3 Data Representation and Conversion

As mentioned in section 3.7.1, the Tide_Converter class is an abstract class that defines the interface for data conversion. The Tide_Converter class provides hooks for concrete subclasses to override for application specific code. Concrete instances of this class will have a reference to the user defined data model, the result is a geometrical representation of the data, which is currently represented in Open Inventor format and is written to a file. This file is then downloaded by the TIC. Handler objects use a converter to generate a specific type of model, for example if the TIC needs streamlines to be generated from a data set, then the handler would use a streamlineConverter defined for the purpose and a streaklineConverter if streaklines were required.



3.7.2 Tele-Immersion Client

The TIC provides an interface to the data, hence its prime responsibility is to handle user interaction and creation of the shared virtual environment for the participants. Currently two types of interfaces are provided for user interaction: a query interface to interact with the TIS and a 3D interface to directly interact with the visual model. The virtual environment is created based on Limbo, which provides a set of basis classes for creating teleimmersive applications.

3.7.2.1 Limbo

CAVERN_perfLimbo0 is a part of the CAVERNsoft G2 toolkit. CAVERN_perfLimbo0 provides the basic shell to build teleimmersive applications. Several other modules included within the toolkit are used for basic functionality within the TIDE client.

An avatar module, namely CAVERN_perfAvatar_c, manages connections between multiple users in a virtual environment. The information pertaining to each remote user is wrapped into a CAVERN_perfBaseAvatar object. The avatar module manages this collection and

makes sure that information of the local user is transmitted to remote users and vice versa. This module creates two network connections with the world server (which is the central collaboration server for the avatars): a TCP connection is used as a hailing channel to signal the entry and exit of users to and from the environment; and a UDP connection is used as a tracking channel to continuously transmit information about the local users' tracking of body positions. The `CAVERN_perfLimbo0` class provides the functionality to load a set of default scene objects into the environment, TIDE uses these objects as elements for providing feedback about the visualization environment. These can be further refined as needed in the future.

The `TIDEclient_c` class inherits from the `CAVERN` classes, the basic functionalities to manage avatars and extends this capability to support user interaction and collaborative visualization. This class implements a `render_callback` function which is executed once every time after the scene is rendered. It is here that most of the user related events are handled. The `TIDEclient_c` class also references a background loader namely the `TcGfxBgLoader`, which is responsible for loading the visualization in a background process. The `TIDEclient_c` class also references an abstract `TIDENet_mediator` class that handles the communication with the TIS and the world server for visualization data and collaboration respectively. So in its implementation of the `render_callback` function, a `TIDEclient_c` object calls the `load_callback` function of the background loader to handle new load requests since the last call to it and the `handle_request` function of the `TIDENet_mediator` to handle communication of user requests etc.

3.7.2.2 User Interface Protocols

Limbo only provides the capability to navigate through a static virtual environment. TIDE has the additional requirements of the ability to query a remote data computation service as well as manipulating the resulting visualization geometry. The framework breaks these interfaces into two classes, TIDEui_query and TIDEui_vis respectively as shown in

. Both these classes inherit from the abstract CAVERNmisc_subject class, and hence are subjects, which are observed by the CAVERNmisc_observer class (see Observer Pattern).

These classes implement a basic function, whenever this function is called all the observers interested in this event are updated. Concrete implementations of these classes need to provide application specific details to build a user interface for querying and visualization and to notify observers of all events. These abstract classes also specify a function that needs to be called to check for any user interaction.

3.7.2.2.1 Query Interface

The role of the query class is to communicate with the TIDE server. Its implementation will provide a user interface for the user to make queries with the data server. The abstract class leaves the application specific details to be implemented by the application programmer. The query can be done through interaction within the virtual environment, communicating with a handheld device, or using a 2D/3D widget interface. The query interface may or may not have any coupling with the virtual environment itself, and implementing it as a separate class allows for a wider range of extensibility. A concrete TIDEui_query class needs to provide application specific and functions.

3.7.2.2 Visualization Interface

Interaction within the visualization geometry is contained within a concrete implementation of the TIDEui_vis class. Like the query interface class, the abstract base class provides no concrete functionality. It serves only to provide a base class for references with other parts of the framework. The visualization interface can extract information about the user's actions within the virtual environment in order to apply changes to the visualization. These extractions could involve obtaining information such as CAVE Wand button presses or any other input device such as the data glove.

After a query process has been completed, it is typically the visualization interface, which adds the geometry to the virtual environment. Any interactions or manipulations of that geometry are the responsibility of the concrete visualization interface, though some basic manipulations such as rotation and scaling are provided. A concrete TIDEui_vis class needs to provide application specific and functions.

3.7.2.3 Network Mediator

The Network mediator (TIDEnet_mediator) handles network communication for the client. It inherits from the CAVERNmisc_observer class and hence is based on the Observer software design pattern[24], the mediator observes instances of each of the TIDE user interface clients. When an operation is performed on any of the interfaces, a notification is sent to the mediator. The responsibility of the mediator is to then transmit appropriate data to the server and other connected users (through the world server), if required.

The network mediator implements a function which is called once between two consecutive frame updates, it is this function that concrete subclasses need to override to

handle application specific functionalities. For example: in a simple case this function would call the `updateVisualization` functions of the query interface object and the visualization objects to see if any user interaction has occurred, any user interaction would in turn trigger other events i.e. as observers are notified of events. The network mediator is also an observer of a `CAVERNdb_client_c`, which allows a seamless collaboration infrastructure for visualization states. Whenever a remote collaborator modifies the state of his/her visualization, the `updateVisualization` function of the network mediator is called, as it is a registered observer for this event. Similarly if the local user modifies (i.e. rotates or scales) the visualization, this information is sent to other remote users. The `TIDEmpi_TIS` provides a TCP communication channel to the TIS. The network mediator implements the `updateVisualization` function to be updated when new data arrives from the TIS, this is called when the `notify` function of the `TIDEmpi_TIS` object is called.. The network mediator checks if any new data has arrived by calling the `isDataAvailable` function. Concrete classes of `TIDEmpi_TIS` need to specialize the `updateVisualization` and `isDataAvailable` member functions.

The TIDE client class has a single network mediator to broker all the network communications for the querying and visualization services of the client application. The mediator contains references to concrete instances of the query and visualization interfaces. When the mediator receives network information, it propagates the data to the appropriate interface for updating. For example if an application programmer needed to capture a particular user input say the threshold value for an isosurface extraction module on the TIS the query interface would provide a widget to get the user value when its `getThresholdValue` function is called by the `updateVisualization` function of the base network

mediator. A concrete CAVERNnet_mediator would implement a specialized function that will be invoked when a user enters a threshold value. This function would in turn send the information to the TIS through the message-passing interface (i.e. by using the `send` function). The `send` function send a packet of data to the TIS, each packet has an integer ID and a data buffer.

The previous sections gave a description of the TIDE framework, how a simple application can be written using this framework is dealt with in this section. A very simple application is considered wherein a user interface button click causes the TIC to request the TIS to reverse a character string. I shall call this application `reverse_string`.

The TIS and TIC exchange packets of information over a TCP channel. The basic packet is defined by a `CMD_PACKET` structure as follows:

```
typedef struct
{
    int cmd;
    int size;
    double timestamp;
    char data[DATA_SIZE];
}CMD_PACKET;
```

Where:

cmd:	Command/Request/Reply identifier
size:	Size of the packet
data:	Data being sent
timestamp:	This field is unused but may be used to specify the timestamp for the packet

Implementation details for the application are given here below:

3.8.1 Extensions required for the TIC

This section is based on the assumption that a user interface is provided that generates a button click event whenever the user clicks a wand button. This interface is encapsulated in a

concrete implementation of TIDEui_query class, namely `TIDEui_query` class. The application programmer needs to do the following:

1. Since only a single button click event exists, `myApplicationQuery` class does not override the `notify` function of the `TIDEui_query` (section 3.7.2.2.1). A button click would cause the `notify()` function to be called which would in turn call the `queryUpdate()` function of the abstract `TIDEnet_mediator` interface.
2. The application programmer needs to provide a concrete `TIDEui_query` class that inherits from the base mediator class and specializes the `queryUpdate()` function for the application as follows:

```
void myApplication_mediator :: queryUpdate(char * data)
{
    int size = sizeof(data);
    tideServerClient->sendCmd(CMD_USER, data, &size);
}
```

3. `tideServerClient` is an instance of `TIDEmpi_TIS` class which is written by the application programmer, and is a concrete `TIDEmpi_TIS` class. It overrides the `checkForNewData()` function of the base class and provides an application specific function that notifies the observer, the `myApplication_mediator` by calling its `newDataArrived()` function, which prints the reversed character string that is gets as follows:

```
void myApplication_mediator::newDataArrived( char * data)
{
    cout << "This is the reversed string returned by the Server :'" << data << endl;
}
```

3.8.2 Extensions required for the TIS

The tasks that an application programmer needs to perform are enumerated in this section. The programmer needs to implement concrete application specific classes that define the interface provided by the abstract classes. The following steps show how a simple TIS will cater to a user request

1. Firstly, the TIS needs to interpret the client request. To do this the application programmer needs to write a concrete Tide_myApplicationServer class that inherits from the Tide_Server class. This class implements the handleCommand() function as follows:

```

CMD_PACKET * Tide_myApplicationServer::handleCommand(CMD_PACKET * pkt)
{
    Tide_BaseCommand * cmd;
    switch(pkt->cmd)
    {
        case CMD_USER:
            {
                myApplicationHandler * handler = new myApplicationHandler;
                myApplicationCommand * appCmd = new myApplicationHandler(handler);
                appCmd->setData(pkt->data);
                cmd = appCmd;
                return cmd->execute();
            }
        default:
            //handle default operation;
            break;
    }
}

```

handleCommand() is called from within the handleClient() function associated with each handler process

2. Concrete `myApplicationHandler` and `myApplicationCommand` classes will replace the `Tide_BaseCommand` and `Tide_ApplicationCommand` classes shown in the UML class diagram for the TIS. `myApplicationCommand` class interprets the client request and

stores the client command `CMD_USER` as a member variable (`myString`) in its specialized implementation of the `setData` function as follows:

```
void setData(char * data)
{
    strcpy(myString, data);
};
```

`myApplicationCommand` class also provides an accessor for this string value: `getString()`, implemented as follows:

```
char * getString(){ return myString;};
```

The `execute()` member function implementation is inherited from the `Tide_BaseCommand` class and calls the `executeCmd` for the mediator, in this case the instance: `handler`, of `myApplicationHandler`

3. `executeCmd()` is implemented by `myApplicationHandler` as follows:

```
CMD_PACKET * myApplicationHandler::executeCmd(Tide_BaseCommand * cmd)
{
    CMD_PACKET * replyPacket;
    char * newString = reverseString(cmd->getString());
    strcpy(replyPacket->data, newString);
    replyPacket->cmd = CMD_USER;
    replyPacket->size = strlen(newString);
    return replyPacket;
}
```

`reverseString()` is a simple function that reverses the contents of an input string and returns the reversed string. The reply packet returned by this function is finally returned by the `handleCommand()` function of `myApplicationServer` class to the handler process, which asks the main parent process to forward it to one or more clients. To intercept this in the main process, `myApplicationServer` class should implement the `processHandleRequests()` function.

4. For example if the reply needs to be sent to everyone then the processHandleRequest() function will be implemented as follows:

```

void Tide_myApplicationServer::processHandlerRequest(MSG* preqMsg)
{
    // Do what the process has requested
    switch(preqMsg->msgno)
    {
        case CMD_USER:
        {
            CAVERNnet_tcpClient_c * procsClient;
            int size = strlen(preqMsg->data);
            //send data back to tide Client
            procsClient = clientDB->getClient(preqMsg->pid);
            if(procsClient)
            {
                if(allClients)
                    sendToAll(preqMsg->msgno, preqMsg->data, &size);
                else
                    sendToClient(procsClient,preqMsg->msgno, preqMsg->data, &size);
            }
            break;
        }
    }
}

```

sendToAll() and sendToClient() are inherited from the base Tide_Server class, these functions send the data to a single client or to multiple clients. The getClient() function provided by clientDB (instance of Tide_Client_Manager) retrieves client information based on the process id of the handler process allotted to the client.

Shared memory is used for Inter Process Communication between the handler processes and the main parent process. The MSG structure defines the messages exchanged between the

main parent process and the handler processes. The MSG structure is defined in the as follows:

```
typedef struct msgstruct
{
    int pid;
    int msgno;
    char data[MSGMAX - HEADERSIZE];
}    ;
```

where:

pid: process id of the sending process

msgno: TIC command being serviced

data: data being sent, the TIC knows the format of the data.

MSGMAX defines the maximum message size, the user can set this and the HEADERSIZE is set to the size of the 'pid' and the 'msgno' fields. Semaphores are used for concurrency control. The application programmer does not have to implement any IPC related code, at most the application programmer needs to set the MSGMAX value etc.

A very simple application of the TIDE framework was described here. The TIDE framework integrates collaboration with visualization and applications developed based on this framework will have collaboration inbuilt. The visualization algorithms may vary and need to be implemented by the application programmer. In the next chapter an application of TIDE to collectively visualize data retrieved using data-mining algorithms is described in detail.

This chapter describes an application of the TIDE architecture. A brief introduction of the application is followed by a description of the application specific functionalities that had to be implemented.

Data servers store large multidimensional data sets such as results of scientific simulations and digital outputs of Geographical Information Systems. These data sets contain information, which is as yet undiscovered and could be potentially useful to the analyst. The TIDE framework can be applied to visualize these “hidden gems” of information.

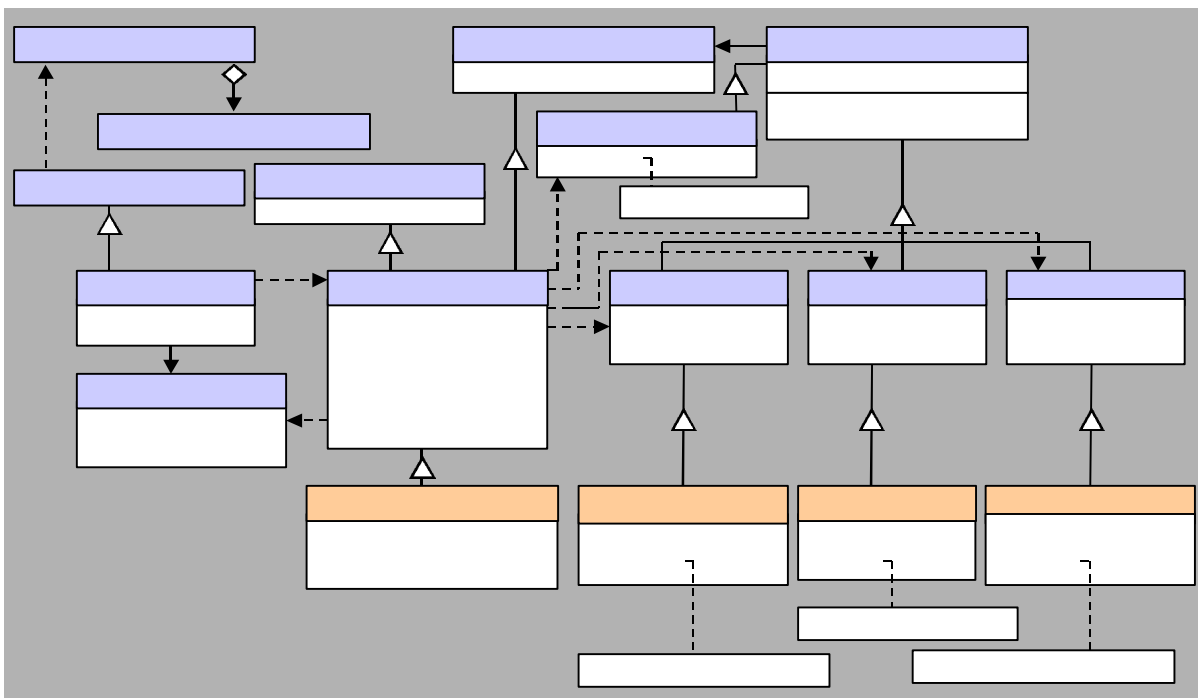
The Laboratory of Advanced Computing at the University of Illinois at Chicago focuses on providing an infrastructure for mining and exploring remote distributed data[16]. They use a protocol for retrieving data from remote nodes on the Internet namely the Data Space Transfer Protocol (DSTP). Servers on some of the nodes called DSTP Servers perform the task of data retrieval. The data on the nodes is organized in tables, with each row in the table corresponding to a data point and the columns specify an attribute of the data point. Attributes of two data points can be correlated if they have more than one attribute in common; this common attribute is the Universal Correlation Key (UCK).

In the implementation of TIDE that interacts with the DSTP servers, the TIC can specify the attributes of the DSTP data that its interested in visualizing. The TIS then retrieves the data from the DSTP Servers. Once the TIS gets the data the client can specify how the attributes are to be correlated and visualized. One can correlate various attributes of the data set by plotting them against one another on a three dimensional graph or by generating a three dimensional histogram.

Since virtual reality devices such as the CAVE provide one more dimension for visualization, the client can select an attribute for each of the three axes (X, Y and Z) as well as the visual representation of each attribute of the data set. When generating visualization from a multivariate data set, the multiple dimensions need to be mapped to different visually perceptible attributes, so that the user can rapidly and accurately identify trends/anomalies in the data sets. In the current TIDE architecture a user can map three attributes to the X, Y and Z-axes and can specify the color, transparency and the size of the points in the 3D graph to be controlled by three other attributes. In this way six dimensions of the data can be visualized at a time.

DSTP Servers have been populated by gridded weather data provided by the National Oceanic and Atmospheric Administration (NOAA). The data consists of monthly satellite measurements of global surface temperatures, precipitation, ozone levels and vegetation index. Complete data sets are available for every month of the years 1985 - 95. The UCKs for this application are latitude and longitude, gridded in one-degree intervals. Each data file consists of about 64000 rows (roughly 360 times 180) and the columns of the file are: latitude, longitude followed by a particular attribute. The combined size of all files is approximately 450 MB.

As implemented for the Data-Space Transfer Protocol (DSTP), the TIDE client contains two different aspects for its interface. A two-dimensional query interface is used to communicate with the DSTP servers during the process of specifying data parameters for visualization. Within the virtual environment, users may navigate through the world and perform



manipulations of the visualization geometry using a six-degree-of-freedom wand interface. The extensions made to the TIC framework for the TIDE-DSTP applications is as shown in

Additional classes, which were added are orange in color. These are the `TIDenet_mediatorDSTP`, `TIDEui_queryDSTP`, `TIDEui_visDSTP` and the `TIDEmpi_TIS_DSTP` classes.

4.2.1 2D Query Interface

Virtual environments still lack the tools for easily creating visual interaction common for actions similar to those taking place in standard widget-based user interfaces such as Motif, Microsoft Windows, or the MacOS. Ongoing research in this area faces several challenges, in particular: low resolution of output displays, pointing and clicking trouble, extensive development time.

To visualize DSTP data the client first needs to specify which datasets it is interested in. One DSTP server may have ozone information and another might have temperature information. If a client wishes to correlate the ozone and temperature then they need to have a common UCK. The user can select the servers from a list, and then a list of UCKs for the data on the servers is presented. Once the user selects a particular UCK, a list of data files containing the attributes to be visualized is shown. On selecting the data files the user can then select the attributes of interest. A two-dimensional Query Interface allows the user to make all the selections. The user can also select the percentage decimation to be applied.

Once the TIS server retrieves the information, the user can map various attributes to visually perceptual cues. Another 2D Mapping Interface is popped up for this purpose. The user can plot each of the attributes to different dimensions as shown or a histogram can be generated to correlate the attributes. Once the user is done with the analysis and wishes to look at other information, a re-query can be initiated by pressing the Reset button, which pops up the Query interface.

Due to a need for rapid development of the interface for querying the data servers, the decision was made to use a two-dimensional interface. The XForms library is used to present

a Motif-based widget interface for performing data queries. The Xforms interface is managed by the TIDEui_queryDSTP class. The specialized handleInteraction() function checks if any user input has been made via the Xform interface, if it is then the observer, TIDEnet_mediatorDSTP's queryUpdate function is called (section 3.7.2.3). The windowed interface of XForms is notably different from the immersive environment of running in a CAVE or on an ImmersaDesk. The window appears separate from the environment of the shared world, something that can be an advantage over immersive widget libraries with which visual components can be clipped by geometry in the scene.

4.2.2 Wand Interface

The Wand is used to directly interact with the visualization. The user can rotate the object by pressing the first wand button and changing the wand orientation, the visual model is rotated accordingly. Keeping the middle button pressed and moving the wand can scale the object. To scale up the model, the wand is moved towards the user and vice versa. The user can navigate using the joy pad. The TIDEui_visDSTP concrete class manages this internally. The specialized handleInteraction() function checks the status of the Wand and notifies the TIDEnet_mediatorDSTP class of any user interactions, by calling its visUpdate() function.

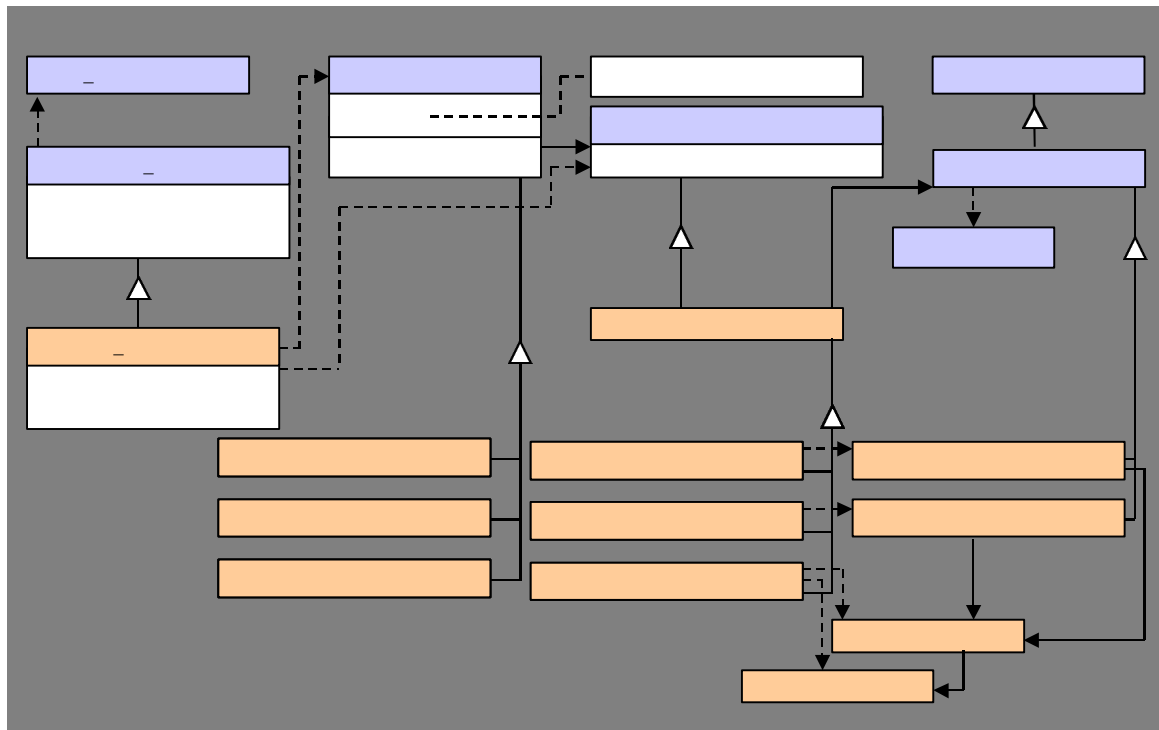
The query interface is brought up and hidden using one of the buttons on the CAVE Wand. This allows the user to perform queries while still viewing the visualization, yet hide the 2D interface when navigating through the virtual environment.

4.2.3 Communication with the TIS

As explained in section 3.7.2.3. the net mediator uses a message-passing interface provided by the TIDEmpi_TIS class to communicate with the TIS. However, the unmarshaling of TIS

replies is done by concrete implementations of `newDataArrived()` and `TIDEmpi_TIS_DSTP`'s `checkForNewData()` function.

Application extensions made to the TIS framework are shown in `Tide_DSTP`. `Tide_DSTP` is a concrete server that implements specialized `handleCommand()` and



`processHandleRequests()` for the TIDE-DSTP application as explained in section 3.7.1.1.

4.3.1 Data Representation

The DSTP data is represented using a relational model. The data is retrieved in a tabular format. Each column in the data corresponds to a particular attribute, and a record defines a data point described by its attributes. To allow the attributes to be mapped to any dimension, `Tide_Parameter` and `Tide_Dimension` objects are created. A `Tide_Parameter` object stores all

the information for a particular attribute and provides the interface to the data. Tide_Dimension objects reference a parameter object based on the users mapping. All the parameters are normalized, and are then mapped to the dimensions. A three-dimensional histogram to co-relate three parameters can also be generated from the data easily.

4.3.2 Data Conversion

On having mapped the attributes to the correct dimensions a geometrical model has to be generated to reflect the mappings. The TIDE framework provides an abstract converter class that defines the steps needed to convert the data into a geometrical model and the reply to be sent back to the client(s). Hooks are provided where the subclasses can do the actual conversion. Specialized converters are written that generate a 3D plot or a histogram from the data, namely Tide_3D_Graph_converter and Tide_Histogram_Converter. The converters use Writer objects to generate an output file. Currently the framework supports a writer for Open Inventor files (i.e. Tide_iv_Writer). Additional writers can be provided for different graphical formats.

4.3.3 Application Specific Commands

The TICs send requests to the TIS for different services. In this particular application of TIDE, the different types of requests made by the TIC are: to generate a 3D graph from the DSTP data, to correlate three attributes using a 3D Histogram, to query the DSTP servers for the data. The query interface on the client allows the user to specify the DSTP servers, UCKs, datafiles and attributes that interest the client. The TIS talks to the DSTP servers using the DSTP protocol and completes the client request. The Tide_CommandQuery and the Tide_QueryHandler objects do this. The Tide_SwitchingHandler manages the switching of

the view from a 3D graph to a 3D Histogram and Tide_MappingHandler maps different attributes to the different dimensions and generates a 3D graph accordingly. Tide_SwitchingHandler uses Tide_Histogram_Converter to generate a 3D histogram and Tide_MappingHandler uses Tide_3D_Graph_Converter to generate a 3D graph ()

A demonstration of TIDE/DSTP was given at Supercomputing '99, Portland, Oregon. TIDE users running on ImmersaDesks at various exhibit booths (notably the National Center for

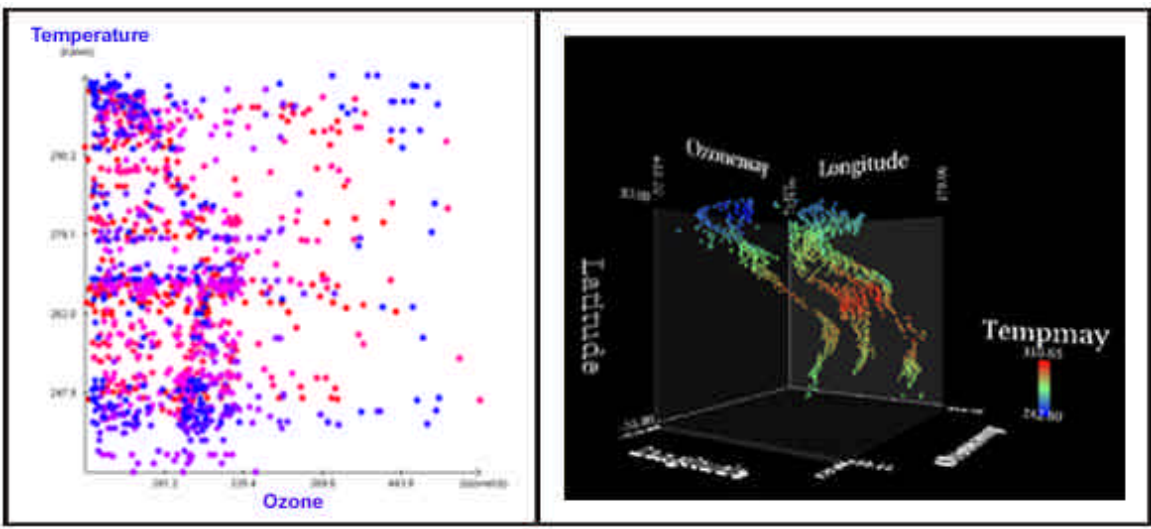


Data Mining, Argonne National Laboratory and Alliance exhibit booths) and a PowerWall at the Advanced Strategic Computing Initiative (ASCI) booth, collaboratively queried and correlated 500 MB of atmospheric data (provided by NOAA) distributed amongst several servers situated in Chicago. A snapshot of TIDE at SC99 is shown in . Two TICs

(in Portland) are visualizing atmospheric data from DSTP servers situated in Chicago. One user is on the PowerWall, the other (see inset) is on an ImmersaDesk.

The current DSTP client interface correlates the attributes and UCK's by plotting a two dimensional graph. A UCK/attribute can be on each of the X and Y-axis and the color of the data-points can be mapped to a UCK. The benefits of using a three-dimensional data representation as opposed to a two-dimensional representation can be seen in

, below. shows a 2D plot of Temperature vs. Ozone generated by the DSTP client and shows a 3D graph generated by TIDE. Longitude is mapped to the X-axis, Latitude is mapped to the Y-axis and the Z-axis shows the Ozone concentration and the color is mapped to temperature. It is easy to discern the hot equatorial belt and the depletion of Ozone at the south pole.



At SC99 the distributed nature of TIDE allowed TIC to connect to the TIS for data, it was not necessary for them to have data stored locally. This can be of great advantage specially if the TIC is to run on a low end work station.

Although we experimented with an atmospheric data set, TIDE and DSTP can be used to visualize various other multidimensional data sets stored on DSTP servers. Ongoing research on TIDE is now focused on the visualization of ASCII data.

The previous chapter described an application of the TIDE architecture for visualizing DSTP data. This chapter analyses the design in greater depth, to identify its advantages and disadvantages.

TIDE lays the foundation for building teleimmersive data visualization applications. The application programmer does not have to bother about handling issues such as leveraging collaboration amongst the clients. Implementing a distributed multiprocessed Client/Server architecture for TIDE resulted in a number of advantages; some of them are listed below:

5.1.1 Centralized control of collaboration

The central server mediates all the information exchanged between the clients. All the updates from a client are first sent to the world server which forwards the data to all other participants, similarly the TIS generates the visualization based on a single clients request and forwards it to the rest, this way a central server has control over events. The central server can arbitrate any conflicts and maintain a uniformity to ensure consistency in the shared environment.

5.1.2 Centralized Location for Data

The TIS handles request for data from all the clients, it stores a local copy of the data for all the clients. The clients do not have to expend memory and CPU resources performing

operations like data retrieval, representation and conversion. The server alleviates the client of performing these tasks, at the expense of increased latency which is introduced due to the delays in the network in fetching the data from the server and talking to the server. This does not affect the rendering frame rate as the client is not blocked in waiting for the data, the user can still interact with the other collaborators, as the server that handles the avatar information is different from the one that is doing the computation.

Since the client only has an interface to the data, it can focus more on improving the rendering task. There is no need to have a monolithic client that handles all the complexities as the tasks are distributed over several processes.

5.1.3 Multiple Processes

Having a separate process to handle a particular client, allows the TIS to pay individual attention to every client. Also since every process runs in its own address space, there is very little possibility that a faulty process will affect the other processes, this increases the robustness of the application.

Since each process has a local copy of the data for the client, it may lead to replication of data and may affect the overall performance of the applications that have a large number of clients connecting to the server. To improve the performance of the processes themselves may be distributed over several processors or a global copy of the data can be maintained in shared memory if all the clients are to visualize the same data.

5.1.4 Large Data Visualization

The centralized Client/Server architecture also allows the client to specify what subset of the data it is interested in and the server then retrieves only that information for the client. In this

way all the data need not be loaded into memory. However for this design to be implemented efficiently the database that hold the data should have a hierarchical structure or should allow for selective retrieval of data based on some input parameters. There is ongoing research on using hierarchical formats for data storage[26].

5.1.5 Extensibility of the TIDE Framework

The TIDE framework has been designed keeping in mind the requirements of collaborative applications for data visualization. It provides a message-passing interface between the TIC and the TIS. The Command-Handler abstraction allows future applications to support different types of client requests. Hooks have been provided to add application specific code. For example: if an application programmer were to develop an application that allows multiple users to visualize a volumetric data set, the TICs query interface needs to be implemented according to the applications requirements, command-handler pairs will have to be implemented for each request from the TIC to the TIS, a suitable data model that captures all the properties of the volumetric data needs to be implemented. Depending on the kind of visualizations to be generated from the data i.e. isosurfaces, streamlines, isocontours etc. specialized converters need to be implemented. No code needs to be written for maintaining the shared VR environment or for network communications between the TIS and the TIC. However, the interface between the TIS and the TIC needs to be defined as well as that between the TIS and the RDCS. Implementation of the interface between the TIS and RDCS is required, CAVERNsoft G2 can be used for this purpose if TCP/IP is the networking protocol.

For the TIDE-DSTP application it was very easy to extend the client queries by just implementing the appropriate command and handler classes. Also it was very easy to plug in the 3D Graph converter, which generated a 3D graph, based on the users selections of mappings for the various dimensions and the Histogram converter, which generated a 3D histogram to correlate three attributes. Only the interface for the TIC needed programming, as most of the VR framework for collaboration was in place and the client did not have to do any computations pertaining to the data. Since the TIC is a thin client and does not need to have the data to be stored locally, we did not have to use very high-end machines to run the client at SC99, and it worked well on an SGI O2. The data was stored on DSTP servers in Chicago, the client could query for this data from Portland seamlessly. On the show floor there were two clients collaborating with each other and simultaneously querying for data. One problem does exist, as currently the server does not arbitrate which client controls the visualization, both the clients can change each other's visualization. This led to situations where a client would be analyzing the data and suddenly saw a new image. A method to rectify is suggested in the future works section and can be easily implemented.

This chapter discusses the various issues in extending the TIDE architecture to handle time varying datasets and to provide a persistent environment for synchronous/asynchronous collaboration. The current architecture only supports synchronous collaboration where each collaborator has local control over his/her data.

The extensions described in this section decide the future course of TIDE, some of these ideas may be rudimentary and are subject to further research.

A time dependent data set consists of different data files, each file corresponding to a particular time value, all the other data values in the file are instantaneous values sampled/calculated at the point in time for which the file was generated. Studying the visualization as a function of time shows the gradual evolution of the other attributes with time. Loading the time steps in the order of increasing time can provide this information. This is an important feature desirable in applications, which wish to see/determine the correctness the progress of an experiment or simulation. An application may provide functionalities to the user to start, stop or pause the animation of the time steps or to progress n-steps forwards or backwards.

Visualization of time-varying data sets poses a new challenge to collaborative visualization.

In a CV application when one participant is sequencing through the different timesteps it may be necessary that all the participants look at the same time step at any instant of time. Different approaches can be taken to solve these issues.

One approach would be to let the TIS arbitrate which frame is being visualized. Only one participant, as discussed in the previous section, will control the state of the animation. When the leading participant goes into an animation mode, the TIS is made aware of all the selections made by the user, this same information is broadcasted to all the participants immediately. So if the leading participant is visualizing time-step 1 then all the other participants will load time-step 1. This approach is feasible with our approach of having one participant decide on what everyone is seeing.

Several issues arise when each time step is very large, it is not possible to load all the time steps into memory, one way of solving this problem is that when a client is visualizing time step x , it already has time step $x+1$ (and maybe $x-1$) already cached, and the server is working on generating time-step $x+2$ (or $x-2$). To further reduce the size of the files the client can specify which attributes it is interested in, and only those are reflected in the time steps.

This is an interested topic for further research. Adding the above mentioned functionality to the framework would involve defining the protocol between the server and the clients i.e. define additional messages to be passed and how they will be handled on either side, the messaging mechanism remains the same only the commands which are passed and their handlers will have to be defined.

Currently TIDE allows for synchronous collaborative sessions, where all the participants have control over their data locally but the client who made the last query defines the visualization. The TIS can mediate on which client controls the visualization. The TIS can generate a “Visualization Token”. The client that wants to play the lead role, can ask for the token, and gets it if it is available. Only the client who has the token decides what every one sees. This gives rise to several issues like, what happens if the client crashes or does not relinquish the token. In such cases the TIS should arbitrate and reclaim/reset the token.

Collaboration is controlled both by the world server as well as the TIS, to handle synchronous sessions both should have a shared client database, that also includes information regarding which session (if at all) the client belongs to. So that avatar and model state information is local to a particular session.

The architecture can be easily extended to allow global control over the data i.e. instead of each client having a local copy of the data they all share a single copy. In such a case the data would be in shared memory accessible to all the handler processes. The TIS can control the modifications to the data by allowing only one client to modify the data at a time (by implementing read/write locks for concurrency control).

Different approaches can be taken to manage asynchronous collaboration. One approach to allow multiple collaborative sessions is to modify the collaboration servers namely the TIS and the world server to handle groups of clients, instead of a simple client database. A new TIC will be notified of existing sessions and can then join the session that it is interested in. Replies to query and data will then be multicast to clients belonging to a specific group.

Another alternative is to allow for a single session, to join another session the TIC will have to connect to the TIS handling that session.

One desirable characteristic in Collaborative virtual Environments is to have a persistent environment. A user, who is in the middle of a data analysis session and has to leave the virtual environment, but wishes to resume later from the same point where he/she left the session, should be able to do so. In such a case the TIS should save the information pertaining to that particular client, so that the TIC can resume his analysis from where he/she left it. If a client crashes in the middle of a session, he/she should be able to resume from a consistent point.

This can be achieved by saving the state of the client process at regular intervals. Since there is a separate handler process for every client, only the context of the process needs to be saved. Object serialization techniques can be used to save the clients environment.

A two-dimensional X form interface is used for certain user inputs. A mouse is required to make any selections. In some cases if there is no workstation close to the VR system a user has to keep going to the workstation to make any selections. This is very unwieldy; the user should be able to work from within the virtual environment. In order to provide an integrated visualization environment, three-dimensional interfaces such as 3D menus and widgets should be used, however these are still a topic of research and no standard exists for easily defining three-dimensional environments for virtual environments.

Future applications of TIDE may need to support a wide variety of data types, and different methods for filtering and visualizing the data. One possibility of providing this is to use an existing library such as the Visualization Toolkit (vtk)[27], that provides a rich set of algorithms for iso-surface extraction, decimation etc and has an object oriented open source structure that can be further extended. In the future it may be possible to integrate into the tide client a visual programming tool that allows users to use visual programming techniques to build a visualization network (in the virtual environment) and the TIS can then interpret the script generated from the visualization network to build a corresponding visualization network of extended vtk modules. The input module may read from the data source the segment of data that the client is interested in.

A suitable Tele-Immersive application for scientific visualization should seamlessly integrate collaboration with data visualization, giving the user a more in-depth insight into the data. This requires integrating the fields of virtual reality, visualization and networking. The focus of this thesis was to study the issues involved in the development of such applications and lay the foundation for the rapid development of tele-immersive applications for data visualization.

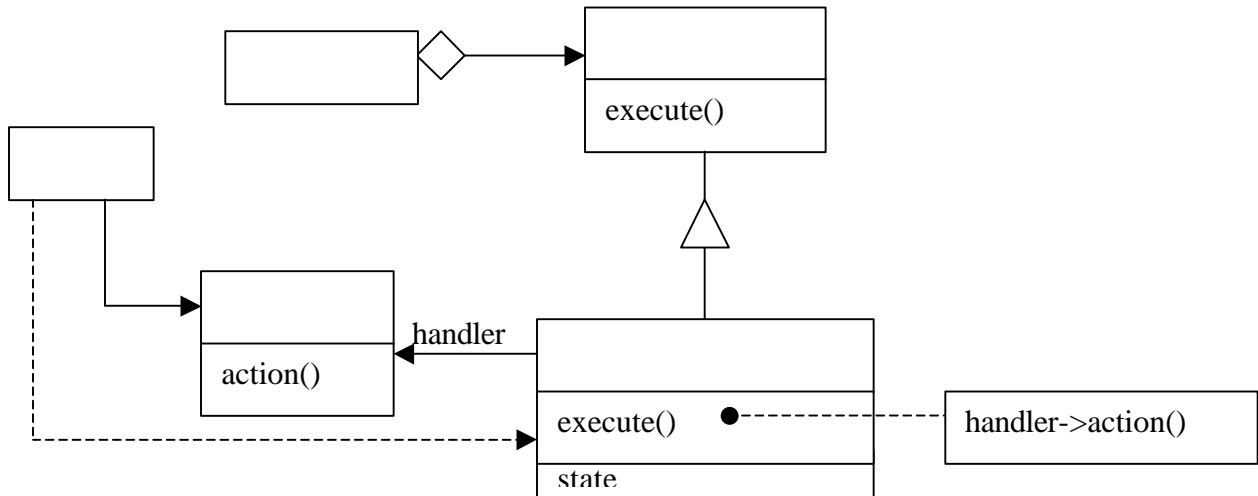
TIDE has a distributed multiprocessed architecture that allows remote clients to collectively participate in a collaborative session. An application of this framework was used to collectively visualize over 500 megabytes of data stored on remote data servers. The application supported the visualization of regular scalar data managed by a distributed RDBMS. The TIDE framework can be extended to support other types of data and visualization techniques.

A standard paradigm for the development of 3D widgets for user interaction in immersive environments is still a topic of research. Availability of such a standard will greatly reduce the application development time considerably. This thesis serves as a starting point for further research in the field of Tele-immersive Scientific Visualization.

	<u>PAGE</u>
1 Command	80
2 Observer	82
3 Template Method	83

1. Command

The command pattern is a behavioral pattern that allows a request to be encapsulated as an object. The client is abstracted from how the request is executed, the execution of the request is done by a handler object.



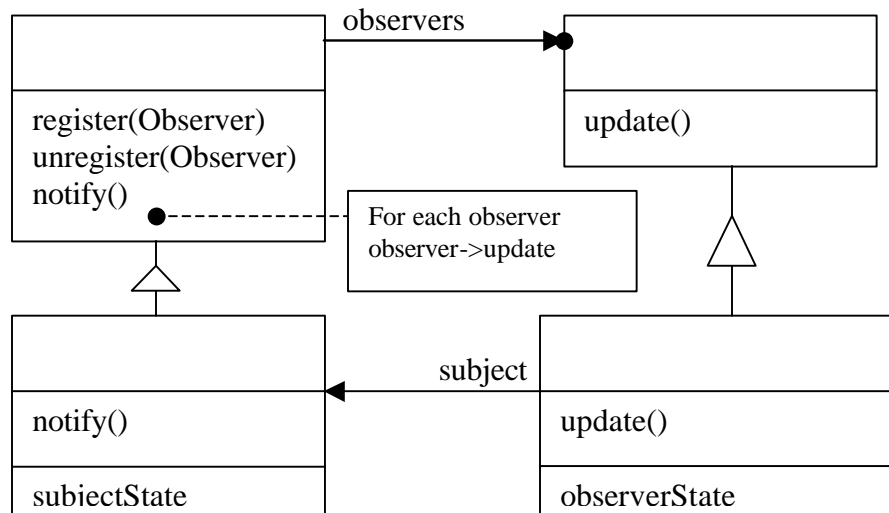
The `Invoker` class provides an interface for executing an operation. A `Handler` class implements this interface. The `Invoker` class creates a `ConcreteCommand` object, sets its state and specifies the `Handler` object for it.

The `ConcreteCommand` object implements `execute()` to call the `action()` function of the `Handler` object and may pass a reference to itself as a parameter. The `Handler` object uses this reference to query for the state of the command object. In this way the `Invoker` only needs to call the `execute()` function on the command object, the invoker is thus totally abstracted from “how” the command is executed.

In the context of TIDE (see Figure 7) the Tide_ConcreteServer class corresponds to the Client as well as the Invoker. The Tide_BaseCommandMediator class corresponds to the Handler and Tide_BaseCommand corresponds to the Command class. The executeCmd() function is the action function of the Tide_BaseCommandMediator class called from the execute() function of the Tide_BaseCommand class. The ConcreteServer can thus handle multiple requests from a client by instantiating corresponding command-handler pairs and calling execute() on the command object. The ConcreteServer can maintain a list of commands from a client and defer execution of certain commands if needed.

2. Observer

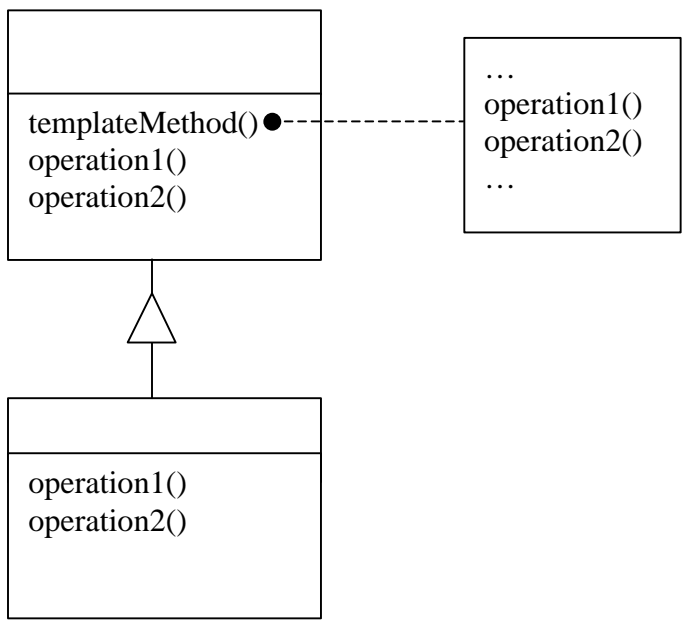
The Observer pattern is generally used to define a dependency relationship between classes. It allows a class to notify a set of dependent classes of any change in its state. The dependant classes are generally termed as observers. The class, which is being observed, is the subject class.



The `Subject` provides an interface to allow observers to register and unresiter themselves. Whenever an event of occurs which causes the subject to change its state the `notify()` function is called which in turn calls the `update` function on all the registered observers. The `Observer` class defines the update interface. `ConcreteObserver` instances can then do the needful in their implementation of the update function. `ConcreteSubect`'s can also provide an interafce to allow the observer to query its application specific state.

The observer pattern is used extensively in TIDE (see Figure 8), the query interface and the visualization interface are all subjects being observed by `TIDEnet_Mediator`.

The template method pattern is used to define a skeleton of an algorithm in a operation. The basic invariant steps in an algorithm are defined and those that vary are deferred to application specific subclasses.



defines a templateMethod() function to implement the steps of an algorithm. Operation1() and operation2() are hook operations which are called in the template method, application specific implementations are provided by the . Thus the Abstractclass defines and implements invariant steps in an algorithm and the Concrete classes need to implement the variant steps.

The Tide_Server class is such an AbstractClass that defines the handleClient() template method, which calls the hook operation handleCommand() whenever a request arrives from a

particular client. The `handleCommand()` function needs to be implemented appropriately by application soecific subclasses.

- [1] Leigh, J., Johnson, A., DeFanti, T., “CAVERN: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments”. In Virtual Reality: Research, Development and Applications, Vol. 2.2, December 1997 (1996), pp. 217-237
- [2] Leigh, J., Johnson, A., DeFanti, T., et al. “A methodology for Supporting Collaborative Exploratory Analysis of Massive Data Sets in Tele-Immersive Environments”. 8th IEEE International Symposium on High Performance and Distributed Computing, Redondo Beach, California, Aug 3-6, 1999.
- [3] Wood, J., Wright, H., Brodlie, K., “Collaborative Visualization”. In Proceedings of the Conference on Visualization 1997.
- [4] Arns, L., Cook, D., Cruz-Neira, C., “The Benefits of Statistical Visualization in an Immersive Environment”. In Proceedings of IEEE VR '99, pp. 88-95, September 1998
- [5] Foulser, D., “IRIS Explorer: A Framework for Investigation”. In Computer Graphics, vol. 29(2): pp 13-16, 1995.
- [6] Argiro D., Kubica S., Young M., and Jorgensen S., “KHOROS: An Integrated Development Environment for Scientific Computing and Visualization”. In Computer Graphics, 29(2): pp 22-24, 1995.
- [7] Young M., Argiro D., Kubics S., “Cantata: Visual Programming Environment for the Khoros System”. In Computer Graphics, vol. 29(2): pp 22-24 1995.
- [8] Abram G., and Ternish L., “ An Extended Data-Flow Architecture for Data analysis and Visualization”. In Computer Graphics, vol. 29(2): pp 17-21 1995.

- [9] Advanced Visual Systems (AVS) <http://www.avs.com/products/index.htm>
- [10] Kuo E., Lanzagorta M., Rosenberg, Simon J., and Summers J., "VR Scientific Visualization in the GROTTO". In Proceedings of IEEE Virtual Reality, 1998.
- [11] Roy Trina M., Cruz-Neira C., DeFanti T., "Cosmic Worm in the CAVE: Steering a high Performance Computing Application from A Virtual Environment". In Special Issue of Presence on Networks and Virtual Environments: Teleoperators and Virtual Environments, Fall 1994, pp. 121-129.
- [12] Bryson S and Levit C., "The Virtual Wind Tunnel: An Environment for the Exploration of Three Dimensional Unsteady Flows". In IEEE Computer Graphics and Applications, July 1992
- [13] Jaswal V., "CAVEvis: Distributed Real-Time Visualization of Time-Varying Scalar and Vector Fields Using the CAVE Virtual Reality Theater". In Proceedings of the Conference on Visualization '97, 1997, pp. 301
- [14] Renambot L., Bal E. H., Germans D., Spoelder H., "CAVEStudy: an Infrastructure for Computational Steering in Virtual Reality Environments"
<http://www.cs.vu.nl/~renambot/vr>
- [15] Leigh, J., Johnson, A., "CALVIN: an Immersimedia Design Environment Utilizing Heterogeneous Perspectives". In Proceedings of IEEE International Conference on Multimedia Computing and Systems '96. Hiroshima, Japan, June 17 - 21, 1996, pp 20-23.
- [16] DeFanti, T., Cruz-Neira, C., Sandin, D., "Surround Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE". In Proceedings of ACM SIGGRAPH '93, Anaheim, CA, July 1993.

- [17] Creel, E., Grossman R., Reinhart, G., “DataSpace: Protocols and Services for Distributed Data Mining and Remote Data Analysis”. Conference: Knowledge Discovery and Data Mining August 20-23, 2000, Boston MA, submitted for review.
<http://www.lac.uic.edu/>
- [18] Harrand V, Choudry A., Zeirbarth J., “Scientific Data Visualization: A Formal Introduction to Rendering and Geometric Modeling Aspects”, In Proceedings of Supercomputing '90, 1990, Pages 775 – 783
- [19] Swami A., A White Paper on “Data Mining with Technology from Silicon Graphics and Oracle”, November 1994, http://www.sgi.com/software/mineset/mineset_data.html
- [20] Cox M., Microcomputer research labs, Intel Corporation, Ellsworth D., MRJ/NASA Ames Research Center, “Application-Controlled Demand Paging for Out-of-Core Visualization”. In Proceedings of the Conference on Visualization '97, 1997, pp. 235
- [21] Collaborative AVS: <http://vis.sdsc.edu/users/documentation/vis/avs5/>
- [22] Parker, S.G. and Johnson, C.R. “SCIRun: A Scientific Programming Environment for Computational Steering”, Supercomputing `95, 1995. <http://www.cs.utah.edu/sci/scirun>
- [23] SGI’s MineSet <http://www.sgi.com/software/mineset/>
- [24] Gamma E., Helm R., Johnson R., and Vlissides J. “Design Patterns: Elements of Reusable Object-Oriented Software”. Addison Wesley, 1995.
- [25] Lin, C., Loftin, R., and Nelson, R. H., “Interaction with Geoscience Data in an Immersive Environment”, In Proceedings of IEEE VR 2000.
- [26] Folk M., A White Paper on “HDF as an Archive Format: Issues and Recommendations January 27, 1998. NCSA/University of Illinois
<http://hdf.ncsa.uiuc.edu/archive/hdfasarchivefmt.htm>

[27] Visualization Toolkit <http://www.kitware.com/vtk.html>

Nikita U. Sawant

Master of Science, Electrical Engineering and Computer Science,
University of Illinois at Chicago, Chicago, IL, 2000

Bachelor of Engineering, Computer Engineering,
Thadomal Shahani Engineering College,
University of Bombay, India, 1996

(August 1998 – Present) Electronic Visualization
Laboratory, University of Illinois at Chicago, Illinois, Chicago.

(October 1996 – July 1998): TATA Consultancy Services, India,
Bombay.

- Demonstrated the current version of TIDE collaboratively at various exhibit booths (notably the National Center for Data Mining, Argonne National Laboratory, Alliance and ASCI booths). TIDE Clients collaboratively queried and correlated data distributed amongst several DSTP (Data Space Transfer Protocol) servers. DSTP is a next-generation data mining tool for the distribution, inquiry and retrieval of data columns for correlation studies.
- Research Demonstrations at SuperComputing'98 titled "Exploring CAVERNSoft TeleImmersive Collaboratories at the I-GRID / NLANR / Alliance Booths."

"The Tele-Immersive Data Explorer: A Distributed Architecture for Collaborative Interactive Visualization of Large Data-sets," with Chris Scharver, Jason Leigh, Andrew Johnson, Georg Reinhart, Emory Creel, Suma Batchu, Stuart Bailey, and Robert Grossman. Proceedings of The Fourth International Immersive Projection Technology Workshop, June 19-20, 2000, Iowa State University, Ames, Iowa