View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Purdue E-Pubs

Purdue University

Purdue e-Pubs

Department of Computer Science Technical Reports

Department of Computer Science

1994

Cooperating Brokers to Support Cooperative Work

Vinod Anupam

Chandrajit Bajaj

Peinan Zhang

Report Number:

94-009

Anupam, Vinod; Bajaj, Chandrajit; and Zhang, Peinan, "Cooperating Brokers to Support Cooperative Work" (1994). *Department of Computer Science Technical Reports.* Paper 1112.
https://docs.lib.purdue.edu/cstech/1112

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries.
Please contact epubs@purdue.edu for additional information.

COOPERATING BROKERS TO SUPPORT COOPERATIVE WORK

Vinod Anupam
Chandrajit Bajaj
Peinan Zhang

CSD TR-94-009
February 1994

Cooperating Brokers to Support Cooperative Work

Vinod Anupam Chandrajit Bajaj Peinan Zhang

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
Email: {anupam,bajaj,pnz}@cs.purdue.edu
Phone: (317)494-6531 Fax: (317)494-0739

Abstract

We motivate the utility of brokering systems in CSCW and highlight the requirements of cooperative brokering. We also demonstrate how we have augmented the infrastructure of a prototype CSCW environment called SHASTRA to accomodate cooperative brokering. Several applications and possible scenarios of CSCW with cooperative brokering are also presented. We describe how cooperating brokers can be used to exploit plurality and commonality of tasks in a cooperative setting, improving throughput for the entire group.

Keywords:

Task Brokering; Load Balancing; Scheduling; Cooperative Problem Solving; CSCW Infrastructure; Groupware; Virtual Team Support;

1 Overview

This section motivates the use of brokering in CSCW, in the context of related work. Section 2 briefly introduces the architecture of Shastra and highlights the main features of the system. Section 3 presents requirements for cooperative brokering, and describes the brokering infrastructure of the Shastra environment. Example CSCW applications that exploit brokering and have been implemented in SHASTRA are briefly presented in Section 4. Section 5 addresses future direction.

1.1 Brokering in CSCW Environments

Task brokers provide a mechanism by which clients programs needing service make requests to brokers and receive responses transparently. Brokers work seamlessly in heterogeneous distributed environments. Clients are typically interested only in the results of service requests, and are not usually concerned with how and where the request is serviced. In brokering, clients are abstracted away from implementations of the services that they require. Brokers are responsible for locating service implementations for a client request, conveying appropriate request information to the implementations, and finally conveying results to the clients. Thus, clients use brokers which interoperate with servers to provide services.

Brokering is useful as a facility that allows access to non-native functionality in applications. They have traditionally been used in single user systems. Service requests used in brokering systems are usually computationally large, to benefit in the classic compute-communicate tradeoff. In the design arena, brokers are used to access servers for analyses, simulations, animations, and other special purpose computation not locally available in an application. They are used to conduct database and file system searches in information systems.

Cooperative brokering refers to the notion of systemic and mutual interoperation of multiple task brokers along with the exchange of supporting information in order to speed up the process of servicing requests for multiple clients. Here, brokers maintain meta-informatic descriptions of the tasks that they are performing, infrastructural information about local workstations, and other state information, in order to possibly optimize on actual service requests issued to servers, effectively reducing total time to service.

In a CSCW setting, users are collaborating to perform tasks. Since they are typically striving towards a common goal, there often is commonality in the computational tasks they need to perform. If the tasks are performed through a brokering system, it is possible to identify the commonalities in task requests and response presentation. This can be exploited to optimize the overall throughput. This is especially true in replicated CSCW systems, where multiple users can simultaneously generate multiple service requests.

1.2 Benefits of Brokering

Brokering has well known benefits.

1. Automatic Resource and Service Location
2. Workstation Cluster Utilization.
3. Load Balancing
4. Scheduling
5. Parallelization
6. Decomposition
7. Modularity

In a distributed system, different types of services may be offered at different points on the network. Brokers relieve clients of the burden of keeping track of where a required service is offered. Brokers can instantiate and use servers on accessible idle machines on the network to service requests. Brokers schedule jobs based on workstation load, speed, and applicability for a particular task, by maintaining relevant information about systems. Dynamic scheduling is used to balance load on server machines. Brokers can utilize the option of executng independent tasks on separate workstations, using the computational power of multiple workstations in parallel to reduce throughput for tasks. It is also sometimes possible to decompose large tasks into multiple independent subtasks that can be executed in parallel. Clients that use brokers consider the service provider to be a black box. This has great utility from the software engineering point of view, since the 'plug-and-play' paradigm it enables promotes software reuse, and eases testing and debugging.

There are additional benefits in the CSCW realm. Cooperating brokers can make optimizations based on the fact that collaborating users often have similar or identical service requests. Such brokers can cache request-response tuples, so that identical requests do not need to be executed again. Thus requests that need results in many sites are executed only once. Application and task dependent partitioning of tasks provides a finer grain for determination of functional overlap, providing further scope for optimization. Coarse grain tasks can sometimes be decomposed into finer grain subtasks. In addition to the being potentially parallelizable, this presents more opportunities for optimization.

1.3 Related Work

Object Management Group (OMG), an industrial consortium, proposed the Common Object Request Broker Architecture (CORBA) [8], which was adopted from a joint proposal of the constituent companies (DEC, Hewlett-Packard , HyperDesk, NCR, Object Design, and SunSoft). The document defines a framework for different Object Request Broker (ORB) implementations to provide common services and interfaces to support portable clients and implementation objects.

Groupware focuses on using the computer to facilitate human interaction for problem solving. Ellis *et al* present an overview of the field in [10]. The Rendezvous system proposes a powerful architecture for multi-user applications and provides high level support for creating groupware [13]. Language based approaches to generating multi-user applications are described in [12]. GroupKit presents a mechanism for creation of realtime work surfaces which are essentially shared visual environments [14]. Weasel is another system for implementing multi-user applications [11].

Networked collocation facilities have received a lot of attention *e.g.* MMConf [9], Rapport [1], etc. They provide useful conference management facilities, and support content-independent shared view-spaces.

The CSCW infrastructure of the Shastra system facilitates creation of collaborative multimedia applications [3]. We adopt an abstract application architecture that enables inter-application communication and cooperation. It supports remote task invocation and brokering. We propose a hybrid computation model for CSCW applications which is very effective in a heterogeneous environment. The system provides intuitive session initiation methods, flexible interaction modes, and dynamic access regulation. We motivate the utility of cooperative brokering in CSCW environments by highlighting some applications that would benefit from such a scenario.

Cooperative brokering has great potential in the scientific design and analysis arena, where brokers are already used to access servers for analyses, simulations, animations, and other special purpose computational tasks not locally available in applications. As we head towards CSCW environments to support team design (*e.g.* [2]), the multiplicity of brokering requests for computationally intense tasks shall provide opportunity for optimizations of the kind we describe. In addition, this would benefit CSCW environments for Air Traffic Control and other applications like Office Information Systems which need to conduct database and file system searches.

Another important application domain is Cooperative Information Retrieval, where multiple users collaboratively browse through information servers looking for specific data. In this application, front end browsers could communicate with brokers and drive them to use information servers to retrieve specified information. Cooperating brokers could share common parts of information searches. Collaborative hypermedia browsing is an important application of this setup. Brokers can cache retrieved information for use by all collaborating sites, eliminating the need for multiple repeated searches and retrievals. This has the advantages of reducing network traffic and turnaround time. An important issue here is information representation, which enable brokers to identify commonality.

2 SHASTRA – A Brief Tour

Shastra [1] is an extensible, distributed and collaborative geometric design and scientific manipulation environment. The Shastra system architecture is described in detail in [3]. Example collaborative multimedia applications are described in [2]. Shastra consists of a static and a dynamic component. The static component, the Shastra layer, is a CSCW infrastructure for building scientific CSCW applications. It defines an architectural paradigm that specifies guidelines on how to construct applications which are amenable to interoperation. Its connection and distribution substrate facilitates inter-application cooperation and distributed problem solving for concurrent engineering. Its communication substrate supports transport of multimedia information. The collaboration substrate supports building collaboration-aware synchronous multi-user applications by providing session management and access regulation facilities. In addition to the distribution, communication and collaboration framework, Shastra provides a powerful numeric, symbolic and graphics substrate. It enables rapid prototyping and development of collaborative software tools for the creation, manipulation and visualization of multi-dimensional geometric data.

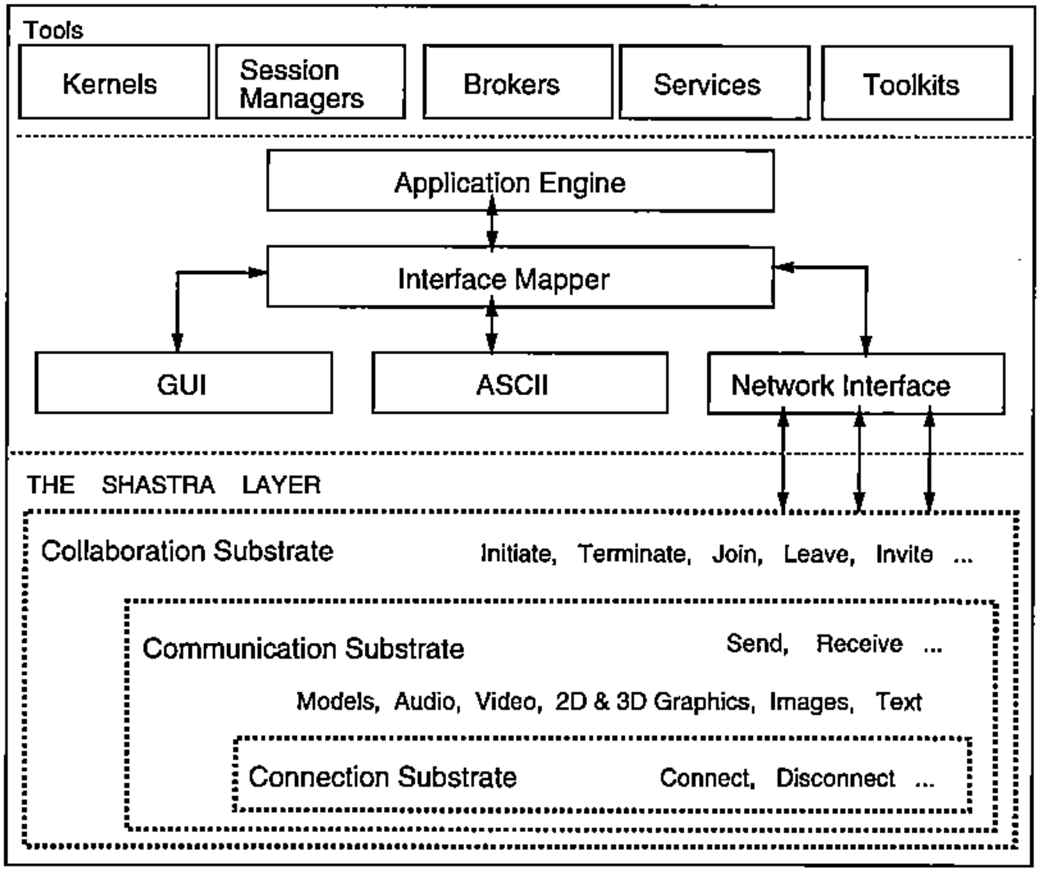
The dynamic component of Shastra is a runtime environment that exploits the benefits of the architectural philosophy and provides runtime support for conferenced applications. The Shastra environment consists of multiple interacting processes, collectively called Tools. Tools are built on top of the CSCW infrastructure of Shastra.

2.1 Tool Architecture

Tools are built with the underlying idea of inter-application cooperation. Every tool is abstractly composed of three layers. The **Engine** is accessed through any of the **Interfaces** via a **Mapper**. The application-specific core, the Application Engine, implements the core functionality offered by the tool. Above the core is a functional Interface Mapper which invokes functionality embedded in the Engine in response to requests from the Graphical User Interface, ASCII Interface or the Network Interface. It also maps requests to alter the user interface or to send messages on the Network Interface. The separation of Engine and Interface makes it easy to build multi-user systems, since it enables maintenance and display of shared state at a user interface due to external commands in a distributed system.

The GUI is application-specific. The ASCII interface is a shell-like front end for the application. Tools communicate with other tools in the environment, via the Shastra substrate, through an abstract Network Interface,

[1] Shastra is the Sanskrit word for Science



tbh

Figure 1: High Level Application Architecture of a Tool in the Shastra Environment – Users interact with tools via the user interface. The Shastra Layer is a connection, communication and collaboration management substrate. Shastra tools inter-operate using facilities provided by this layer, via their Network Interface.

which multiplexes multiple simultaneous network connections, and implements the Shastra communication protocol [3]. The high level block architecture of tools in Shastra is depicted in Figure 1. The architecture makes it easy for tools to connect to other tools and request operations, synchronously as well as asynchronously.

The entire set of connected Network Interfaces of Shastra tools manifests itself as the abstract Shastra layer at runtime (see Figure 1). It maintains the collaborative environment, provides access to functionality of different systems, and provides facilities for initiating, terminating, joining, leaving and conducting collaborations. The connected network interfaces of Shastra tools comprise a distributed virtual machine on which we build problem solving applications.

The Shastra paradigm emphasizes separation of interface and function in applications by building user interfaces as abstract, separate parts. It also makes applications amenable to networked interoperation by allowing access to functionality via abstract network interfaces. This aligns with the concept of configurable user interfaces proposed by [7], where user interfaces are separate, easily replaceable modules which interface with a single back end.

2.2 Runtime Environment

The Shastra environment consists of multiple interacting processes, collectively called Tools. Some tools provide scientific design and manipulation functionality (the Toolkits). Other tools are responsible for managing the collaborative environment (Kernels and Session Managers). Yet others provide specific services for communication and animation (Service Applications). Tools register with the environment at startup, providing information about the kind of services that they offer (Directory), and how and where they can be contacted for those services (Location). The environment provides mechanisms to create remote instances of applications and to connect to them in client-server or peer-peer mode (Distribution). It supports multimedia messaging between different tools (Communication). It provides facilities for starting and terminating collaborative sessions, and for joining or leaving them. It supports mechanisms for different types of multi-user interaction ranging from master-slave blackboarding (Turn Taking) to synchronous multiple-user interaction (Collaboration). Problem solving applications in Shastra consist of a collection of cooperating tools.

Shastra collaborations are implemented using a Hybrid Centralized-Replicated computation model. A central Session Manager regulates collaborative activity of multiple collaboration-aware tool instances.

2.2.1 The Kernel

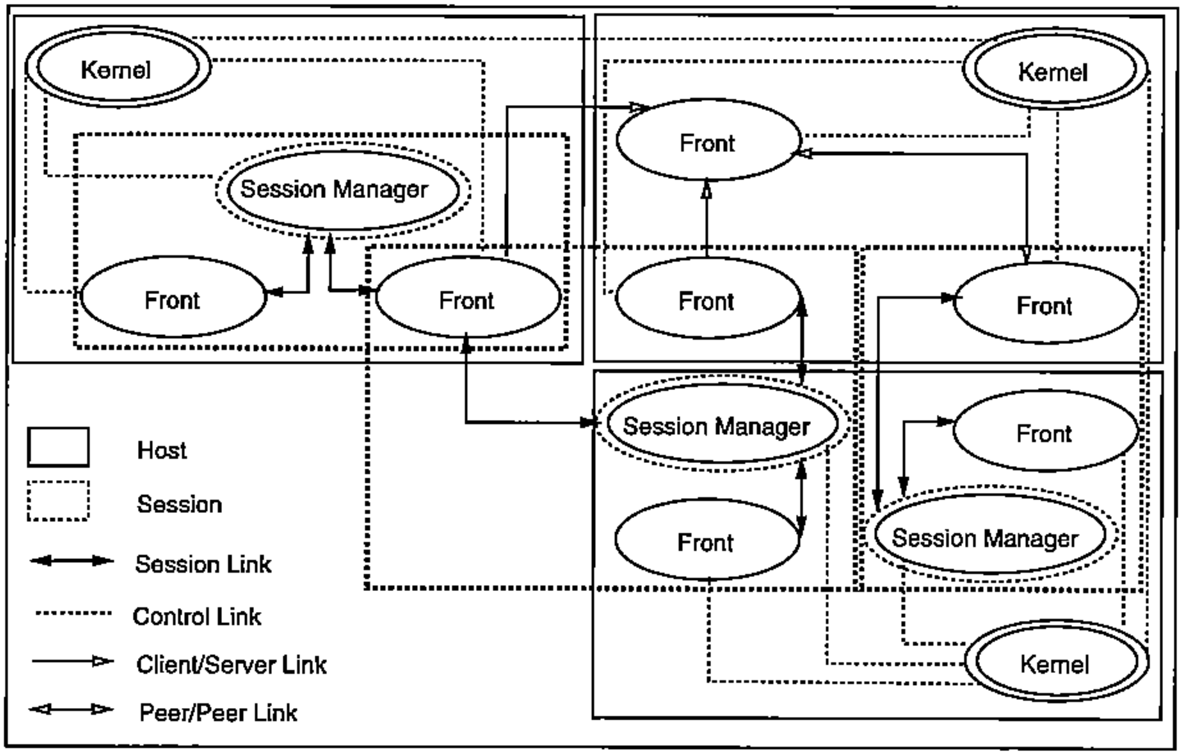
The Shastra Kernel is responsible for maintenance of the runtime environment. It consists of a group of cooperating Kernel processes. It maintains information about all instances of tools in the distributed system, and keeps track of all environment relevant activity. A **Directory** facility lets users dynamically discover what tools (and users) are active in the environment at any time, as well as what functionality they provide. This includes information about ongoing collaborative sessions and their membership. A **Location** facility provides contact information about where the tools are running, letting applications dynamically connect to each other to access functionality. A **Routing** facility enables transport of data and control information between tool instances.

2.2.2 Session Managers

Collaborative Sessions, or Sessions, are instances of synchronous multi-user collaborations or conferences in the Shastra environment. A collaboration in Shastra consists of a group of cooperating tools regulated by a Session Manager, the conference management tool of Shastra. One Session Manager runs per collaborative session. It maintains the session and handles details of connection and session management, interaction control and access regulation. It keeps track of membership of the collaborative group, and serves as a repository of the shared objects in the collaboration. It supports a multicast facility needed for information exchange in a synchronous multi-user conferencing scenario. It has a constraint management subsystem which resolves conflicts that arise as a result of multi-user interaction, enabling maintenance of mutual consistency of operations. It has a regulatory subsystem that controls synchronous multi-party interaction, and provides a floor control facility based on turn–taking.

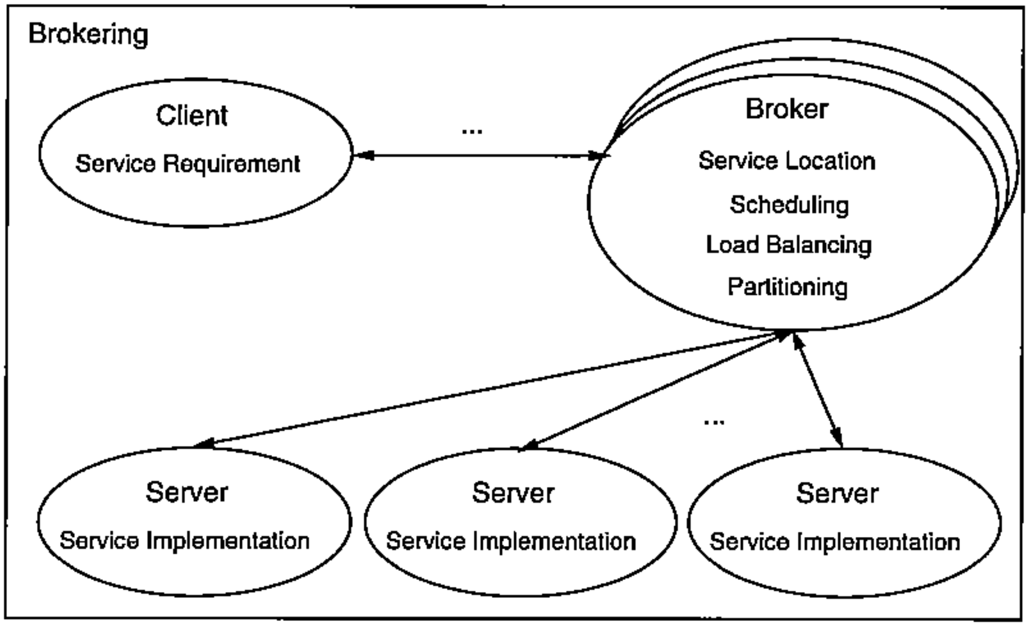
2.2.3 Fronts

Front End or Front is the term used to collectively refer to all tools in the Shastra environment that a user directly interacts with. This includes Toolkits – actual engineering and design applications, Services – special purpose



tbh

Figure 2: The Shastra Runtime Environment – Note that collaborative sessions span multiple hosts, and that tools can participate simultaneously in multiple independent sessions. Arrows and dotted lines indicate paths of flow of data and control information.



tbh

Figure 3: Brokering in Shastra – Client Tools in Shastra can connect to multiple brokers, each of which can use many servers to service requests. Brokers exchange information and cooperate to optimize request servicing.

applications primarily for communication, and Games – recreational applications. Fronts are created by specializing the basic Front End which is a minimal collaboration-aware tool which understands Shastra protocol messages, and generates requests to interface with the Kernel, Session Managers, Brokers and standard Services. Figure 2 shows a view of the Shastra world, where different tools interact to support a collaborative environment.

Fronts add core functionality to the Application Engine. They extend Network Interface signatures to understand new requests, and to offer different services in the environment. This makes it possible for Fronts to connect to each other in client/server and peer/peer settings to access functionality, and to exchange data.

3 Brokering in CSCW

A brokering implementation should ideally meet the requirements in terms of flexibility and ease of use, efficiency of service, extensibility, and interoperability with different implementations. These needs are highlighted in the CORBA specification [8]. In Shastra we exploit the power of application architecture, and the flexibility of connection setup and networked interoperability offered by the substrate, to provide a brokering service to aid in Collaborative Problem Solving. A typical setup utilizing brokering in the Shastra environment is depicted in Figure 3.

3.1 Brokers

Brokers are specialized Service Tools in the Shastra environment. Brokers create many instances of server tools for the different services offered in the environment. This occurs automatically or under control of tools using the Broker. In the simple case a Broker behaves as a surrogate client. Tools send multiple service requests to a Broker, which uses its set of connected servers to service the multiple requests, and sends the results back to the client tools. All Brokers service the following brokering control requests.

- Start – Request to start a server tool.
- Stop – Request to terminate a server tool.

- Connect – Request to connect across the network to an existing server tool.

- Disconnect – Request to terminate a server connection.

- Service – Request to service a computational task.

- Describe – Request to describe a service request to the broker.

Tools use the Start message to control creation of server instances, and the Stop message to terminate them. They can choose the server instances to be used for the task by sending Connect and Disconnect requests to the Broker. They use the Service message to request the Broker to perform operations on their behalf. The Broker forwards the request to an appropriate server for servicing. The Describe message is used to provide meta-information to the Broker describing the request. The description is shared by cooperating brokers to make optimizations.

In a more complicated scenario, Tools send large computational tasks to application specific Brokers which partition them, in a task-dependent manner, into independent subtasks. These subtasks are then serviced using the connected pool of server Tools. The results are put together and transmitted to the requesting Tools. Application-specific Brokers for different tasks are created from the basic Broker which provides application independent connection, communication and brokering control facilities. Such brokers support additional messages for operations specific to the application.

Brokers exploit application-level cooperation to perform multiple tasks in parallel in a distributed setting, by harnessing the computational power of clusters of idle workstations on a network. They use load balancing and scheduling criteria to optimize computation time of large tasks which can be decomposed into independent subtasks.

3.2 Services

We use brokers in Shastra to transparently provide high level services to client applications.

3.2.1 Multimedia

An application area in the context of Shastra, in light of the proliferation of multimedia workstations, is multimedia rendition and multimedia conferencing. Brokers provide a high-level abstract mechanism to handle multimedia interfaces. Client programs that need to incorporate multimedia facilities interact only with the interface device broker. The brokers are designed to support recording and playback of multi-media information. Using Shastra facilities for developing collaboratve applications enables us to configure cooperating brokers to provide media conferencing to the requesting clients. Clients that use this facility need not be collaboration aware and gain the benefit of a conferenced architecture transparently.

The main advantages of this approach are

1. Abstraction

2. Heterogeneity

3. Extensibility

4. Modularity

Clients deal with multimedia device brokers at a very high level of abstraction. This shields them from details of media and device handling. It also supports platform independence. The modularity provides isolation from issues of hardware and software upgrade. Possible optimizations in a heterogeneous CSCW setting include avoidance of redundant media format conversions.

3.2.2 3D Graphics

Another important application area in the Shastra context is that of $3D$ graphics rendering and conferencing. In this setting, brokers handle $3D$ view generation and display, under control of client drivers. Clients need not be aware of the local graphics platform. They simply send data and control requests to the broker. Once again, Shastra facilities are used to make brokers collaboration aware. Cooperating brokers can therefore provide conferenced $3D$

viewing and interaction transparently to collaboration unaware clients. This primarily benefits applications where collaborative visualization is needed.

The advantages of a brokered graphics system are

1. Heterogeneity

2. Modularity

Brokering supports platform and graphics device independence, shielding clients from low level details of graphics interaction. Brokers isolate display and rendering functionality, easing incorporation of $3D$ graphics facilities in clients. The modular interface provides easy upgradability to faster software and hardware platforms. A possible optimization in a conferenced setting is shared view generation. Note that, in general, the network bandwidth requirements of transmitting centrally generated shared views at high rates in a conferenced scenario are too high, even with the most advanced networking and compression technologies.

3.3 Applications

There are the following many possible user, broker and task configuration scenarios in the Shastra brokering system. We are specifically interested in the k user, n broker, m task situation which represents the typical CSCW setting we would like to address. Using a single broker in this setting would make it the hub of a star topology, which would negatively affect performance. Multiple instances cooperate to provide higher throughput.

We have built applications that exploit the Shastra brokering system in the more traditional sense of providing transparent access to services on a network.

Applications that require lots of scientific computation can benefit greatly from the Shastra brokering system. Here brokers are used to schedule and drive scientific compute-servers to execute multiple tasks on a cluster of networked workstations. Brokers use load balancing criteria to control scheduling in order to optimize throughput. They also 'decompose' tasks, if possible, into independent parallelizable subtasks in an application dependent manner. These subtasks can be then be executed in parallel on multiple workstations.

The benefits derived from this setup include

1. Speed

2. Load Balancing

3. Resource Utilization

Scientific computation based brokering applications allow us to reap all of the classic brokering benefits. Idle workstation clusters are utilized to speedily service multiple requests. Tasks that have parallelizable subtasks are completed faster.

3.3.1 Cooperative Design

An example of multi-user cooperative design in the context of Shastra is Collaborative Smoothing using Shilp and Ganith toolkits [2]. This application permits a group of collaborating Shilp users to collectively smooth out a rough polyhedral model by fitting C^1 continuous patches using Hermite interpolation [5]. The Ganith Algebraic Geometry Toolkit is optimized to perform algebraic manipulation – curve-curve, curve-surface, and surface-surface intersection, as well as interpolation. The Shilp Geometric Design and Modelling Toolkit is optimized for boundary representation based solid modelling. A coordinated nexus of the two has let us add a powerful group design facility to the Shastra environment.

The smoothing operation we refer to provides an easy method for generating solid models with curved surfaces from approximate polyhedral models that have been created interactively. Patch computation for a face is independent of that for other faces, except for continuity requirements, and can be done in parallel. However, surface curvature parameters often require interactive twiddling by the designer, in order to adhere to global or local criteria, and to control the goodness of fit. Collaborative Smoothing parallelizes the design steps by allowing multiple designers concurrent access, and thus significantly improves design throughput. The session manager for this task partitions the object into zones which are collaboratively smoothed by a design team. Multimedia communication facilities permit rapid exchange of ideas and resolution of conflicts in this design scenario.

Generation of the surface patch is a compute intensive operation. The actual interpolation operation is performed by using instances of the Ganith Toolkit, or Ganith servers. In the brokered setting, the Shilp instances communicate with their brokers. The brokers get machine load information from the Kernel, and create multiple Ganith server instances on idle machines on the network. Users can explicitly instantiate servers on specific machines. Shilp sends multiple patch computation requests to the broker, which uses machine load information to determine an optimal schedule. The requests are then serviced on the connected servers in keeping with the schedule. Brokers can be instructed to maintain a dynamic, adjustable schedule which is sensitive to change of load on the server machines. They can also be instructed to return scheduling information to the clients.

The brokers keep track of request descriptions, and cache generated results. They exchange description information periodically, and use cached results to service request repetitions. This setup significantly improves the throughput of large design tasks.

3.3.2 Volume Visualization

Volume visualization is a very intuitive method for interpretation of volumetric data [6]. Measurement-based volumetric data sets arise from sampling – medical imaging geophysical measurements, $3D$ scanning etc. Synthetic volume data sets are generated by computer based simulation and modelling – finite element analyses, computational fluid dynamics etc. Volume visualization provides mechanisms to express information contained in these, typically huge, data sets via images. The synchronously conferenced collaborative volume visualization environment in Shastra [4] lets multiple users on a network share volume data sets, simultaneously view shaded volume renderings of the data, and interact with multiple views. It supports several ways of viewing volumetric data and provides facilities for interactive control and specification of the visualization process.

Visualizing volumes is data and computation intensive. Large data sets are visualized using brokers which partition image space (the volumetric data set) appropriately and use a pool of visualization servers on the network to generate the final image. The brokers use load balancing and scheduling strategies to optimize total rendering time.

In a multi-user setting, and even with a single user, brokers for independent visualization tasks share information about previously visualized parts, using techniques like image caching to further improve throughput. They share common images and parts of images to avoid needless recomputation.

4 Future Direction

We have used the brokering facility of the Shastra environment to provide high level services and to improve the efficiency of some collaborative applications. Users in a distributed CSCW setting can exploit this brokering facility to improve throughput of individual tasks in a collaborative activity. Cooperating brokers can be used to exploit the plurality and commonality of tasks in a cooperative setting, improving throughput for the entire group.

We need to explore formal information representation mechanisms to describe tasks to brokers.

Acknowledgements

This work was supported in part by NSF grants CCR 92-22467, DMS 91-01424, AFOSR grant F49620-93-10138, NASA grant NAG-1-1473 and a gift from AT&T.

References

[1] Ahuja, S., Ensor, J., Horn, D., (1988), "The Rapport Multimedia Conferencing System", *Proc. ACM Conference on Office Information Systems '88*, Mar. 1988.

[2] V. Anupam and C. Bajaj. Collaborative Multimedia Scientific Design in SHASTRA. In *Proc. of the First ACM International Conference on Multimedia, ACM MULTIMEDIA 93*, pages 447–456. ACM Press, 1993.

[3] V. Anupam and C. Bajaj. SHASTRA - An Architecture for Development of Collaborative Applications. In *Proc. of the Second Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 155–166. IEEE Computer Society Press, 1993.

[4] V. Anupam, C. Bajaj, D. Schikore, and M. Schikore. *Distributed and Collaborative Volume Visualization.* Computer Science Technical Report, CAPO-93-50, Purdue University, 1993.

[5] C. Bajaj and I. Ihm. Algebraic surface design with Hermite interpolation. *ACM Transactions on Graphics,* 19(1):61–91, January 1992.

[6] A. Kaufman. *Volume Visualization.* IEEE Computer Society Press Tutorial, New York, 1990.

[7] K. Lantz. An Architecture for Configurable User Interfaces, 1986.

[8] OMG and XOpen. *The Common Object Request Broker: Architecture and Specification.* Object Management Group and X Open, 1992.

[9] Crowley, T., Milazzo, P., Baker, E., Forsdick, H., Tomlinson, R., (1990), "MMConf: An Infrastructure for Building Shared Multimedia Applications", *Proc. ACM Conference on CSCW '90,* Oct. 1990, pp. 329-342.

[10] Ellis, C., Gibbs, S., Rein, G., (1991), "Groupware: Some Issues and Experiences", *Comm. of the ACM,* Vol. 34 No. 1, Jan 1991, pp. 38-58.

[11] Graham, T., Urnes, T., (1992), "Relational Views as a Model for Automatic Distributed Implementation of Multi-User Applications", *Proc. ACM Conference on CSCW '92,* Vol. 1.

[12] Hill, R., (1992), "Languages for Construction of Multi-User, Multi-Media Synchronous (MUMMS) Applications", In Brad Meyers (ed.) *Languages for Developing User Interfaces,* Jones and Bartlett, 1992.

[13] Patterson, J., Hill, R., Rohall, S., Meeks, M., (1990), "Rendezvous: An Architecture for Synchronous Multi-User Applications", *Proc. ACM Conference on CSCW '90,* 317-328.

[14] Roseman, M., Greenberg, S., (1992), "A Groupware Toolkit for Building Real-Time Conferencing Applications", *Proc. ACM Conference on CSCW '92,* pp. 43-50.