EVENT BASED ARCHITECTURE FOR INTERACTIVE COMPUTATIONAL COLLABORATORIES

BY MANJOT DHILLON

A thesis submitted to the
Graduate School—New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Master of Science
Graduate Program in Electrical and Computer Engineering

Written under the direction of
Prof. Manish Parashar
and approved by

_____

_____

_____

New Brunswick, New Jersey
, 2003

ABSTRACT OF THE THESIS

Event based Architecture for Interactive Computational Collaboratories

by MANJOT DHILLON

Thesis Director: Prof. Manish Parashar

A Collaboratory is defined as a place where scientists and researchers work together to solve complex interdisciplinary problems, despite geographic and organizational boundaries. Computational collaboratories provide uniform (collaborative) access to computational resources, services and/or applications. These systems expand the resources available to researchers, enable multidisciplinary collaborations and problem solving, increase the efficiency of research, and accelerate the dissemination of knowledge.

The growth of the Internet and the advent of the computational Grid have made it possible to develop and deploy advanced computational collaboratories. These systems build on the advanced technologies and high-end computational resources underlying the Grid, and provide seamless and collaborative interactions between experts, resources, services, applications, and data. As collaboratories become more pervasive across the Grid, they will be required to handle extremely large number of entities and interactions between these entities. Grid entities may be scientific applications involved in computations that produce data which is of interest to the

users, users trying to access resources or services, to monitor or interact with executing applications, and/or collaborating with other users, or mediator components coordinating the flow of data between the users and the applicationsConventional synchronous request-response style communications leads to a tightly coupled architecture, which may not handle the dynamics of the Grid as entities join and leave the Grid, and may not scale as the number of entities grows. Event-based messaging services which support asynchronous message transfer between decoupled information generators and consumers, lend themselves naturally as a scalable messaging middleware for supporting Interactive Computational Collaboratories especially in a Grid based environment. In this thesis we present an event based architecture aimed at supporting collaboration and steering in an Interactive Computational Collaboratory. The thesis also addresses some of the issues of deploying these Collaboratories in a Grid environment. The implementation and evaluation of this architecture, and its application to the DISCOVER computational collaboratory for interaction and steering is also presented.

Acknowledgements

Dedication

To

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Contributions of the Thesis

The thesis makes the following contributions:

1. Investigate the requirements of Event Services for supporting Interactive Computational Collaboratories. These requirements also capture some of the issues of supporting such collaboratories in a Grid Environment.

2. Develop a design which fulfills some of the key requirements as formualted above. The design makes use of the clustered access patterns found in computational collaboratories to reduce network traffice and increase the scalability of the design.

3. The thesis also makes a dictionary of the event types supported by a given Collaboratory. Theis dictionary, like the XML schema helps describe the type of data supported by a given type of collaboratory.

4. Lastly a prototype of the design is implemented on the discover computational collaboratory. The evaluation of this design is presented for geographically distant producer and consumers.

1.2 Background

A collaboratory is defined as a virtual meeting point where scientists and researches collaborate to solve complex multidisciplinary problems, despite geographic and organizational boundaries[2]. Computational collaboratories provide uniform (collaborative) access to computational resources, services and/or applications such as analytical tools, instruments and raw data, summaries and analyses for multidisciplinary research, archival information and tools for synchronous and asynchronous collaboration. These systems expand the resources available to researchers, enable multidisciplinary collaborations and problem solving, increase the efficiency of research, and accelerate the dissemination of knowledge. The growth of the Internet and the advent of the computational "Grid" [3] have made it possible to develop and deploy advanced computational collaboratories[4][5]. Recent efforts include the following:

1. The Upper Atmospheric Research Collaboratory (UARC)[6][7] provides a virtual shared workspace in which a geographically dispersed community of space scientists perform real time experiments at remote facilities, in locations such as Greenland, Puerto Rico, and Alaska.

2. The Diesel Combustion Collaboratory (DCC) [8][9] is a problem solving environment (PSE) for combustion research providing tools such as a distributed execution management system for running combustion models on widely distributed computers (including supercomputers), web accessible data archiving capabilities; electronic notebooks and shared workspaces; visualization of combustion data; and video conferencing and data-conferencing tools.

3. Access Grid[10] is an ensemble of resources that can be used to support human interaction across the grid, and consists of multimedia displays, presentation and interactions environments, interfaces to grid middleware, and interfaces to visualization environments.

4. Netsolve [11] is a client-server system that enables users to solve complex scientific problems remotely and allows users to access both hardware and software computational resources distributed across a network.

5. EMSL [12] is a symmetric collaboration between computer scientists, domain scientists (physical and biological sciences), and sociologists and relies on the development of new communications technologies - shared computer displays, electronic notebooks, virtual reality collaboration spaces - and an integration of these technologies with current videoconferencing and email capabilities.

6. The Astrophysics Simulation Collaboratory [13] involves a community of scientists, researchers, and developers who wish to collaborate on the development of scientific codes for the astrophysics community at large. It builds on Cactus [14], which is an open source problem-solving environment designed for scientists and engineers in the field of numerical relativity and astrophysics.

7. DISCOVER [1][15][16] provides a virtual shared workspace for scientists and researchers to steer and collaboratively interact with large parallel and distributed applications in diverse fields such as oil reservoir simulations, seismic whole-earth simulations, computational fluid dynamics, and numerical relativity.

Most of these systems build on the conventional request response as the means of communication between the various components of the collaboratories. This architecture may suffice for standalone collaboratories, serving a limited number of consumers but as more and more collaboratories are deployed on the Grid, these architectures do not scale with the increased number of consumers in the Grid.

1.3 Problem Statement

A critical component of these Collaboratories is the interaction between the various entities of the collaboratory. As these collaboratories become more pervasive across the Grid, they will be required to handle extremely large number of entities and the interactions between these entities. These entities may be in the form of scientific applications involved in computations and producing data that is of interest to the users, users trying to access various services offered by the Collaboratory or mediators coordinating the flow of data between the users and the applications. These collaboratories normally rely on the conventional synchronous request response style of communication, which leads to a tightly coupled architecture that does not scale easily when new entities are added. To support these collaboratories in a Grid-based environment alternative architectures are needed that scale with the increasing number of users. Event based systems, based on the publish/subscribe paradigm, have been commonly used for providing decoupled architectures while building large-scale distributed systems. Various event models based on different architectures have been proposed for solving problems in different domains. These Event-based messaging services which support asynchronous message transfer between decoupled information generators and consumers, lend themselves naturally as a scalable messaging middleware in the Grid environment. Most of these designs are generic to large scale applications. Hence we need a design which utilizes the types of interactions and access patterns in a Computational collaboratory to provide a loosely coupled traffic-ergonomic design.

1.4 Organization

1. Chapter1 lays the background of the problem by explaining what computational collaboratories and event-based systems are and the need for a scalable event based service for supporting computational collaboratories in a grid-based

environment.

2. Chapter 2 looks at the background work, compares the various event-based models in existence, relevance of some of the models to our problem domain and then discusses the need for a different event based model.

3. Chapter 3 defines the requirements of an Event based service for supporting the computational collaboratories. We develop the schemas for the various types of events required for access, security and messaging among the different components of the collaboratory.

4. Chapter 4 discusses the design for the Event service, analyzing in detail the mechanisms for discovery, publishing of and subscription to the various event types. It also looks into mechanisms for reducing the event traffic among the various components of the collaboratory.

5. Chapter 5 provides the experimentation for the evaluation of effect in latency across a LAN and a WAN when a) the number of clients are changed and b) the consumers send federated and unfederated requests.

Chapter 2
Related Work

2.1 Middleware for large scale systems

With the information revolution and the advances in the information technology, increasing number of machines are being connected to the Internet. With the consistent increase in microprocessor performance and memory speed and capacity, increasing amount of information is being stored in and disseminated to these peripheral devices. Different middleware technologies have been deployed across the Internet to handle the increasing flow of information.

1. **Client Server** The traditional Client-Server communication model is the simplest and most widely used communication model in the Internet. This model is essentially synchronous with point to point communication between the client and the server. Common information is stored on a server for access by clients that access or manipulate this data. A client invokes a method on a server and synchronously waits for the response. The client and server require some knowledge about the location of the other. This leads to a very tightly coupled design unsuitable for large scale distributed systems. A three-tier design with an intermediate broker mitigates the problem to some extent. It is the responsibility of this broker to route the request to the appropriate server. Thus, the location of the server can change without the need to inform all of the clients. In fact in most implementations this can occur dynamically, providing automatic transfer from one server to another when a fault occurs. The middle tier may also be used to translate information into a format acceptable to the clients. The

three tier-model though less tightly coupled than a simple client-server is still synchronous in that the client has to place a request for data before receiving it from the server.

2. **Publish-Subscribe** Publish-subscribe is another middleware technology for interconnecting applications on a distributed network. In the publish-subscribe model, the unit of information is called an event and consumers express interest in particualr categories of these events. The middleware hides the location details of the producers and the consumers. Also the process of information delivery is asynchronous as the events matching the interests of a consumer are delivered to it as they are produced. Two types of publish-subscribe systems are being used to support various applications :

   (a) **Subject-based Addressing** Subject based addressing is the older and more mature form of publish-subscribe. Various applications in the field of finance, network and weather monitoring, transportation and process automation have leveraged this technology. In this technology each of the events in the application space belongs to one of the fixed set of subjects. Channel-based subscription is a form of subject-based subscription where each subject maps to a given channel. The consumers express there interest through a subset of the subjects and then subscribe to the channel(s) corresponding to these subjects. A big strength of this technique is the possibility of mapping a subject onto a multicast group allowing enhanced scalability and performance. Since the number of muliticast groups are limited, only a limited number of subjects may be mapped. Hence the use of multicast for a wide range of subjects is infeasible. Also if there is an overlap between the various subject categories i.e. a given event is part of more than one subject categories then depending on the interest profile of the consumer, it is possible to get more than one copy of the same event

from different subscribed channels. This leads to a waste of bandwidth and resources. example

(b) **Content-based Addressing** An emerging technology alternative to subject-based publish-subscribe is content-based publish-subscribe. In this technology the information contained in an event is described with the help of an *event schema*. The consumers express their interests by specifying predicate expressions over these event-schema. This allows the filtering of information at a finer granularity as the consumers need not have knowledge about pre-defined subjects. example

2.2 Publish-subscribe systems

The publish-subscribe paradigm is based on the notion of information consumers called subscribers, information generators called producers and a framework which gathers, formats, filters and disseminates the information produced according to the interests of the consumers. A subscription language helps specify the predicates for building subscription objects and a matching algorithm matches the generated events against the subscriptions. Different publish-subscribe systems are the realizations of the above basic idea. Now we look at some of the publish-subscribe models:-

1. **Sienna** Sienna is a large-scale publish-subscribe system with a distributed middleware framework . For scaling in a wide area environment, the middleware is based on a network of servers called event servers. The work also evaluates the connection of these servers in various topologies and proposes various optimizations for efficient routing of events. These include the advertisement operation which optimizes routing of subscriptions and notifications, and the use of IP multicast to replicate events as close to the consumer as possible and application of filter objects as close to the producers as possible.

2. **Eco** Eco stands for Events, Constraints and objects. Event consumers subscribe directly to the event producers by registering a callback reference. Eco specifies three types of constraints, namely notify constraints, pre and post constraints. Notify constraints provide filtering capabilities and pre and post constraints are used for event synchronization and concurrency. Since there is no centralized component like the event channel, this provides for a distributed architecture, without single point of failure. Besides notify constraints, Eco provides another feature to limit the scope of event delivery called zones. Events for a particular zone may not be delivered outside that zone, even for matching constraints.

3. **Gryphon** The work mainly concentrates on efficient matching of the subscriptions to the subscribers interests. They are also investigating the use of IP multicast for dissemination in publish-subscribe systems.

2.3 Middleware support for computational collaboratories

The computational collaboratories require the sharing of resources across geographically distributed entities. As the computational collabratories are increasingly deployed on the computational Grid, the number of entities sharing the resources is expected to become too non-trivial to be supported by centralized messaging systems. A few of the collaboratories which build on scalable middleware substrates in general and publish-subscribe based systems in particular are described below.

1. **Salamander** Salamander describes a push-based publish-subscribe system for collaboratories. Access to the middleware substrate is provided via data server, which may be connected in a suitable topology. The client subscriptions are in the form of persistent queries comprising attribute expressions. The data produced by the publishers is marked by attribute value pairs which are used to make the routing decisions. This is a simplistic publish-subscribe model and

issues related to data discovery, effecient filtering and data aggregation are not addressed.

2. **Narada** A more recent work which addresses wide area event brokering targeting large scale collaborations in education and science is the NaradaBrokering peer-to-peer messaging system. It is based on the publish subscribe model and defines a network of broker peers that can be dynamically deployed to provide efficient events distribution among groups of interested peers. The NaradaBrokering system was built on JMS and then integrated with JXTA.

Chapter 3
Design of the Event Framework

3.1 The Collaborative Environment

As the computational collaboratories are deployed across the Grid, increasing number of users and applications would be connected to these collaboratories. To support this increased load on middleware resources, these collaboratories should be built on truly scalable architectures. Publish-subscribe technology enables building highly decoupled and truly scalable large scale distributed systems. The consumers and publishers are decoupled from each other and the intelligence stored in the data packets allows them to be routed appropriately. Hence minimum state information needs to be stored inside the network and the network scales easily. Optimizations to the basic publish-subscribe model leads to further economies of scale and reduction in network traffic. Any change in type of supported information does not cause a change in the central programming logic and is handled by changing the event and subscription types at the end points.

3.2 Requirements for a Event Framework to support Collaboratories

Computational collaboratories differ from regular publish subscribe systems because of the absence of a simple producer-consumer relationship between the application and the clients. A generic Event Service for supporting computational collaboratories should fulfill the following requirements:-

1. **Information Publishing and Discovery** As applications connect to and disconnect from collaboratories, the meta information related to these applications should be dynamically available to the clients.

2. **Subscription** Clients should be able to subscribe to events of interest exported by an application. The subscription language should support specification of logical operations so that clients can subscribe to any logical subset of the event information published.

3. **Access Control** Different level of access privileges should be associated with different categories of clients. A junior scientist may just be allowed to access the monitoring and collaboration services and steering services may be accessible to only senior scientist.

4. **Authentication and Security** The credentials of a consumer may be authenticated by the event-service before providing access to the resources. The event-service may provide a secure communication environment by encrypting or signing the event-messages and the various communication channels supporting the applications and the clients

5. **Event persistence and archival** Collaboratories support an environment where clients are dynamic in nature and are not always connected to the same domain. As the clients move across the domain-space, the event framework may buffer the events produced while the client is disconnected and forward them to the correct domain when the client reconnects. The Event service may also provide a per client history of the events consumed and produced. hh

6. **Quality of Service** The Event-Service may support various parameters for the QoS needs of different applications. An application may require end-to-end reliability or a best effort service. Different priority values may change the way the events are queued and served along the path of event-delivery. This is also

important for applications that require real-time guarantees for time sensitive information.

7. **Efficient Event Dissemination** Some simulation applications may generate astronomical amount of event data. Hence mechanisms are required to efficiently manage and distribute this event data across the Event-Service network.

8. **Scalability** As Grid-based computing becomes more pervasive and the access to applications and communications amongst them becomes standardized, more and more applications will be supported on the Grid. Hence the Event-Service should provide a scalable architecture to support the increasing number of applications and the corresponding consumers accessing these applications.

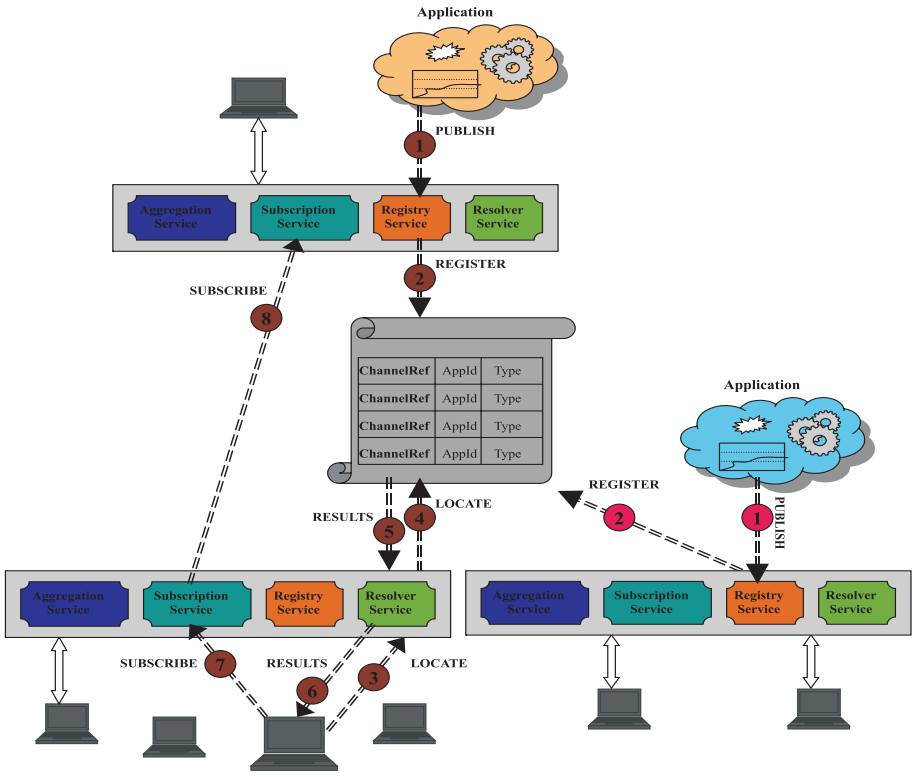

Figure 3.1: Component view of the Event Framework

3.3 The Event Framework

The overall architecture of the Event framework is presented in Figure 3.1. The figure shows the relationship and interactions between the clients of the Event service, the middleware substrate and the registry. The client is a user of the services provided by the Event framework and in publish-subscribe terminology, a client may be a producer of events, consumer of events or both. This enables true peer-to-peer connections between various "'clients"' of the Event framework. The design also leverages the access patterns in computational collaboratories. Collaborating parties generally consist of teams of scientists and a given team is very likely to be geographically collocated. We divide the system-space into domains where each domain comprises of geographically adjacent consumers and producers. For each domain, the Event framework provides services for handling the monitoring, steering and collaboration interactions inside a domain and across the domains.

3.3.1 Domain Services

All collocated clients are managed by the services provided in that domain. There are four basic domain services as below:-

1. **Registry Service** The system uses a registry for storing the information related to the published applications, their location and the interfaces exported by them.The Registry service has a handle to the central registry and registers all the publish requests in its domain. The Registry Service also unpublishes exported information by removing it from the central registry. The central registry contains information about the applications published, the notification channel supporting this application and the type of information exported by the application. Since the central registry only contains meta data information about the application, it is relat ively lightweight and hence is not a bottleneck.

2. **Resolver Service** The Resolver Service enables a consumer to discover and

connect to applications of interest. The need for the resolver service arises because of the fact that the producers and the corresponding consumers may be distributed across various domains. Hence a Service is required which selects a producer based not only on the type information published but also the domain of the producer. The Resolver Service provides the consumer with interfaces for querying the Registry. The result of the query is returned as a list of matching applications and the corresponding notification channel references.

3. **Subscription Service** The consumers subscribe to applications of interest using interest profiles. For applications in the same domain as the consumer, the Subscription Service attaches the subscriptions to the local event channel. For applications in a remote domain, the Subscription Service obtains an aggregated profile for that domain using the Aggregation Service. It then attaches an aggregated subscription to the remote channel and individual subscriptions to the local channel.

4. **Aggregation Service** Profile aggregation aims at hiding multiple non-local consumer interests behind a single aggregated interest profile. Hence all consumers in a remote domain can be represented by a single interest profile. Thus a single copy of the matching event is relayed to the remote domain. The remote domain further matches this event against the individual interest profiles in that domain. This leads to a very scalable architecture, where all the consumer interests for a remote domain can be aggregated into a single interest profile and effectively a single consumer.

5. **Authentication and Access control** The division of the design space into domains eases the manageability of checking the authentication and controlling the access privileges of different. Hence authentication and access control for all the clients in a domain is provided by the authentication and control service in that domain. Figure 3.2 shows the case where profile aggregation is not used.

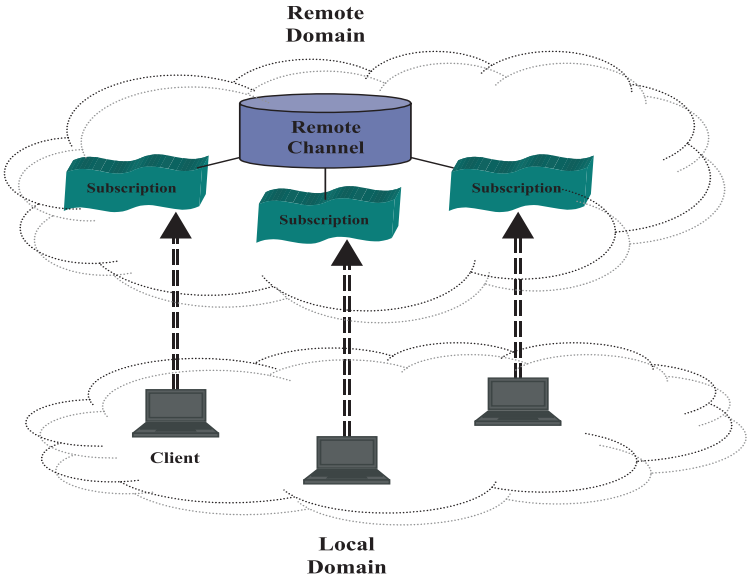

Figure 3.2: Subscription with no Aggregation

The three local users attach three seperate subscriptions to the application in the remote domain. If an event matches all the three subscriptions, then three seperate copies traverse the length of the network between the remote and the local domain.

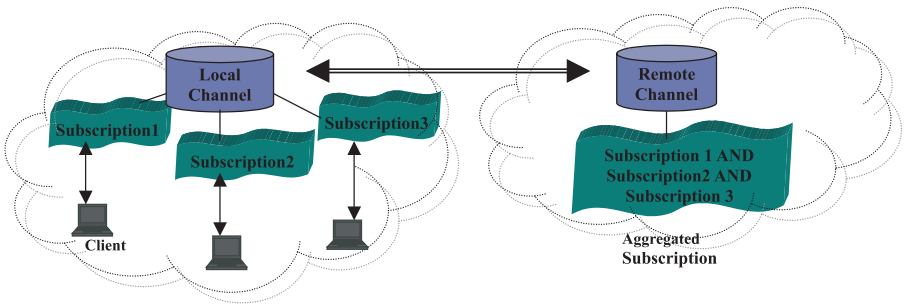

Figure 3.3: An aggregated subscription scenario

Figure 3.3 shows the case where an aggregated subscription, representing the consolidated interests of the local consumers is attached to the remote channel. In this case, only one event copy of an event matching all the three subscriptions is sent to the local channel. The local channel further evaluates this event against the local subscriptions and delivers accordingly. Hence though there is

an exta matching operation, we use only 1/3rd of the network bandwidth. Now the saving in Network bandwidth also depends on the logical overlap between the interest profiles. In case of completely disjoint interest profiles, no event matches more than one profile and hence no savings in network usage. An aggregated profile does increase the complexity of the matching process, but this increase in complexity is more than offset by the decrease in Event copies sent by the remote channel. Also addition of every consumer in the local domain, having an interest in a remote application only causes a modification of the aggregated profile in the remote domain and an additional interest profile in the local domain. Hence the effects of increase in the number of consumers is largely constricted to local domains hosting these consumers.

g

Chapter 4
Implementation of the Event Framework

A publish-subscribe sytem comprises of four major components namely the type of information to be published, a way to publish this information and correspondingly discover the published information, specify interest to the information and finally filter, format, match and disseminate the information to the matched parties. Hence the specification of these four components would enable the implementation of a publish-subscribe system. We build our Event Framework on top of Corba Trading and Notification Services.

1. **Event Dictionary** The Event Dictionary contains information about the type of the Events supported by the system and the information contained in these events. In publish-subscribe sytems, information contained inside Events is used to route them to their destinations. Hence the composition of the Event message should allow the information to be specified in a strcuted way so that it can be easily filtered and matched against the interest profiles. CORBA Notification Service provides the support for Structured events . A Structured event is made up of a header and a body. /reference[FIG] shows the composition of a structured Event message. The header of the event message further consists of two portions :-

   (a) **Fixed Header** The fixed portion of the header contains three fields namely:-

      i. *Event Domain* Originally defined for specifying the industry to which a given event belongs to for example, telecommunications, aviation,

transport etc. Computational collaboratories are application-centric, hence we define this field to be the name of the application exporting this event.

ii. *Event Type* Event type defines the specific type of event inside a given domain. For example a *flightCancellation* event inside the Aviation industry. For our system, this field captures the type of interactions possible inside a computational collaboratory. Capturing the event interactions in a computational collaboratory is complicated by the fact that applications and clients can be both producers and consumers of information. Figure 4.1 shows the roles of applications-clients for different types of events. Hence simple client subscription to application

Entity / Types of Interaction	**Application**	**Client**
Monitoring	Producer	Consumer
Steering	Consumer Producer	Producer Consumer
Collaboration	None	Producer Consumer

Role

Figure 4.1: Application client roles for various event types

published data does not suffice and the application also has to subscribe to events generated by the client. The main event types based on application-client communications inside the DISCOVER Computational Collaboratory are as below:-

A. **Monitoring** Application sends periodic messages about the state of the application and the state of the computations inside the application. An application can be in four states, namely running, computing, interacting or paused.

B. **Steering** Steering messages are commands from the clients to change/query the simulation specific data inside the application. Event channel in a given domain may have more than one applications attached to it. Hence, the application subscription for receiving client commands contains application domain as the event domain and "Steering" as the event type.

C. **Collaboration** Two types of collaboration is addressed. A client may be interested in all the actions and corresponding results of a client. This is achieved by replicating the subscription of the target client and attaching it to the proxy of the interested client. Two clients collaborate by specifying subscriptions for collaboration events generated by each other.

iii. *Event Name* Name specifies the name for a single event instance.

(b) **Variable Header** This portion of the header contains name value pairs used for specifying the QoS qualities for event delivery.

The body of the event message is also composed of two parts.

(a) **Filterable body** This part of the body is composed of name value pairs. Constraint expressions specified in the Filter objects are evaluated against the Fixed header and the name value pairs specified in this portion of the event message.

(b) **Remainder body** This portion carries data attached to the event message in the form of a Corba ANY. The data could be plain text, an executable or a multimedia file.

2. **Publishing information** Publishing event information involves providing means of accessing and searching the meta information related to the published event types and then locating and accessing the publishers satisfying the search criteria. We use the CORBA Trading service for publishing, searching and locating

information of interest. A record of meta information in the Trading Service is called a Service offer and consists of *Service Type* which defines the meta information to be associated with the offer and and *Object Reference* which gives the location of the Object offering the sevice. The following steps are involved in offering, searching and locating information in the Trading Service:-

(a) **Offer Description** The*Service Type* is a description of the Service offer and consists of :-

i. **Interface Name** In our implementation, an Application is bound to the local notification channel and all the event information produced by the application is distributed through the notification channel. Hence for the Clients, the notification channel acts as the Server of information and the Interface Name is the IDL repository ID of the Notification Channel.

ii. **Property Types** Property Types describe the behavour of the Service. A property type consists of a *name*, zero or more *types* and *mode* of the property. A meaningful name provides an automatic description of the property. The type of the property specifies whether its a integer boolean or a complex data type. But the Trader Constraint language can handle only simple data types. The *mode* specifies whether the value for this property is mandatory or optional while exporting the Service Offer. A property which is mandatory for all application service offers is the network number of the local domain of the application. Clients use this value to determine whether the application is in the local or remote domain and propagate their subscriptions accordingly. Queries for selecting offers are evaluated against the values of the property types. Common properties for simulation applications

are the name of the application, category of clients having access privileges to the application etc.

(b) **Service Offer** A *Service Offer* gives information about the service being exported by the server. A Service Offer consists of the following:-

i. **Service type** This is the name of the service type as described in offer description above.

ii. **Object Reference** This is the object reference of the Service exporting the Service offer.

iii. **Property values** The Service Offer also specifies values for all mandatory and zero or more optional properties in the service type description.

The exported offers are queried by specifying the service type, constraints using the Trader Constraint Language and the number of properties to be returned. Common constraints could be the name of an application or all applications for which the client has access privileges. The result of the query operation is returned as a list of Service offers where each offer contains a reference to the object exporting this offer and the property values specified while exporting this offer.After the client obtains the object reference, it interacts with the object directly without any mediation from the Trading Service. The OMG Trading Service specification also has the provision of exporting service offers across different instances of the Trading Service by linking these instances to form a federation of Traders. Thus a local Trading Service may be configured to return results from the local offer space and only forward the query to the federation when no matching offers are found locally. This further leads to a distributed solution to our problem and the interests of the publishers and consumers are restricted to their local domains unless required otherwise.

3. **Subscriptions** Subscriptions are the interest profiles of the consumers and

are matched against the published information to generate events of interest. Specifying the subscriptions requires a query language and we use the Trader Constraint Language which is a standard language for querying the Trader object.

4. **The Event Dictionary** The event dictionary captures the data space of system by specifying the type of events used and the information contained in these events.

5. **Event service interfaces**