Semantic Communication for Distributed Information Coordination

Pravin Bhandarkar and Manish Parashar
Department of Electrical and Computer Engineering, Rutgers University, 94 Brett Road, Piscataway, NJ 08854

Abstract

In this paper we present a new paradigm for information coordination to support real-time collaboration, where the number of collaborating clients, their locations and interests are dynamic. The approach essentially applies the "pull" knowledge management model to distributed information management by defining semantically enhanced messages, and using state-based interactions techniques to communicate and replicate these messages. The approach underlies a Java based collaboration framework. The architecture of the framework and a prelimenary experimental evaluation is presented.

I. INTRODUCTION

The objective of this paper is to formulate, realize and evaluate a new paradigm for information (event) coordination to enable real-time collaboration amongst dynamic groups of distributed clients. The fundamental innovation of the approach is the application of the "pull" knowledge management model to distributed information management. This is achieved by semantically enhancing messages and using state-based[1], interaction techniques to communicate and replicate these messages in real-time. Our goal is to enable each client to have direct and immediate access to all information defined by its needs and capabilities, without having to maintain and update global rosters of all existing clients and theirs interests.

A key requirements for real-time collaborative systems for distributed multimedia applications (collaborative medicine, strategic battle planning and aerospace simulations) is the definition of an effective and efficient model for information coordination and replication in real-time. A further requirement for heterogeneous clients is the ability to locally interpret the event to reflect the interests, capabilities and resources of each client. Traditional event management approaches are based on global naming services, where all communications use unique names assigned to clients. In such a system, every application client that enters a session must register itself with the naming server, explicitly stating its interests. The server then assigns capabilities to the entering clients and informs existing client about the new client's interests. Existing clients can now forward events from the existing collaboration session to the entering client. Clearly, the dynamics of such a collaborative framework is limited by the rate at which the network can synchronize distribute names, interests and capabilities. Furthermore, protocols for system reconfiguration, reorganization and addition/deletion of clients are centralized at the server and can become exceedingly complex.

A. Related Work

Most existing information coordination frameworks for collaboration use the traditional approach to event management and replication outlined above. The Habanero[2] framework developed at NCSA, UIUC, uses a combination of a "central arbitrator" and "central router" to replicate events amongst collaborating clients. Application events are trapped by the "distributed arbitrator", who marshals them and passes them to the central arbitrator. The central arbitrator then uses the central router to multicast them to other clients. All events are managed, replicated and communicated through a centralized server that houses the central arbitrator and central router. The DISCIPLE[7] framework developed at Rutgers University has a session manager to manage event replication and communication using a collaboration bus. Clients in a session are assumed to static in this system, however their interest can change. The distributed collaborative framework at University of North Carolina[6] uses an approach based on bus-agents. In this framework a central server (a Java remote object), located at a well-known resource within a given Internet domain, maintains a central registry and allows users and other bus agents to obtain remote references to active bus agents. Bus agents wishing to allow initiation of communication by another bus agent or process, must register with the central registry. Info-Bus technology[4] provides a push-pull model for the transfer of events, but is limited to a single machine and does not support distributed sessions.

B. Overview

In this paper we present a new paradigm, called semantic event management (SEM), for event coordination and replication to support real-time collaboration amongst distributed application clients. Our approach implements the "pull" distributed interaction model using semantically enhanced events and state-based[1] communication techniques. In this scheme, each client maintains a profile that defines its current state, its interests, and the

capabilities it has been granted. All interactions in this scheme are then addressed to profiles rather than explicit names. As a result, the group of clients receiving a message (e.g., notification of an event) is determined only at run-time. In this formulation, clients can join or networks and the Internet, and will result in truly distributed computation, liberating application clients from static sites and complex tracking protocols, allowing them to migrate freely and utilize available resources. Clients can leave a collaboration session by appropriately defining their profiles, without having to update complex membership rosters. State-based interaction techniques are the application of semantic content-based resolution techniques, used by the naming service, directly to the run-time interaction between clients. These techniques are naturally suited to broadcast/multicast communication.

II. SEMANTIC EVENT MANAGEMENT (SEM): A PARADIGM FOR REAL-TIME EVENT COMMUNICATION:

The Semantic Event Management (SEM) approach formulates all interactions between dynamic sets of distributed, collaborating clients as state-based broadcasts or multicasts. Each client in a collaboration session will locally export a "profile". A client's profile is a mutable set of attributes that specify its type, state, interests and capabilities. Profiles are maintained and modified by the clients to reflect their current interests. All event communications between the collaborating clients are now defined as state-based broadcast or multicast messages where a event message is semantically enhanced to include a sender-specified "semantic-selector" in addition to the message body. The semantic-selector is a prepositional expression over all possible attributes and specifies the profile(s) of clients that are to receive the message. Thus the conventional notion of a static client or client group name is subsumed by the selector which descriptively names dynamic sets of clients of arbitrary cardinality (Conventional names of clients or clients groups are non-descriptive and statically bound.). State-based messages are received by matching object profiles against the message selector. This matching is performed locally and asynchronously at the site of each client. Clients can join or leave a session on the fly. Further, clients control their own profiles and can add, delete or modify attributes in the profile at any time. The modification of a profile is an atomic action with respect to its matching with a selector. The only global knowledge upon which such an interaction depends are the program-specific meanings of the attributes. This implies that all clients have information about possible application objects that can participate in a session and their states; information that is inherently known to all clients.

III. SEM-BASED COLLABORATION ARCHITECTURE

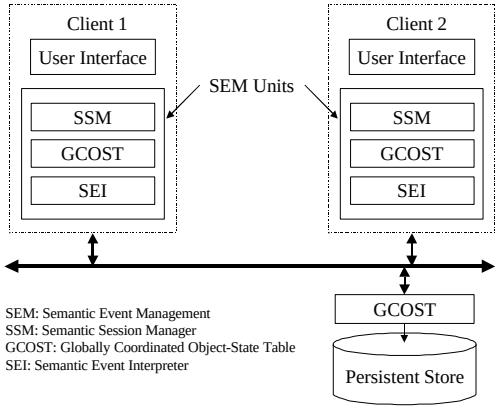

Figure 1 - SEM collaboration framework

A schematic overview of the SEM-based real-time collaboration architecture is shown in Figure 1. The architecture consists of an archival server and a dynamic number of collaborating application clients. The archival server maintains a repository of objects in a collaboration session and their current states, and is used by new application clients joining a session to obtain the current state of the ongoing session. The archival server does not actively participate in a session. The SEM-Unit resides at each application site and is responsible for transparently transmitting local events to all other interested clients as well as communicating interesting remote events to the local application client. The SEM-Unit is composed of (1) an overall SEM Session Manager, (2) a Globally Coordinated Object-State Table, and (3) a Semantic Event Interpreter. These components are described below.

A. SEM Session Manager

The SEM session manager is responsible for locally orchestrating an application client's collaboration session. It monitors all local objects of interest to the client and encodes their state as entries in the Globally Coordinated Object-State Table (GCOST) (see section 3.2 below). Every time an object changes state, a corresponding entry is updated in the GCOST. This update is then transparently propagated by GCOST to all interested clients in the session. Similarly, when a remote instance of the object changes state, the change is received by the Semantic Event Interpreter (SEI) and forwarded to the session manager (via GCOST) who in turn updates the client's session. For example, consider a client session that contains a drawing canvas object. The client's interest in this object and its current state is locally recorded by the SEM session

manager as an entry in GCOST. Each time the clients interacts with the canvas (e.g. draws on it or changes its background color), the SEM session manager updates the GCOST entry to reflect the change. The GCOST hash key is hierarchically generated using the name of the object, its state and the name of the event, and is used to directly index into the multi-level hash table. This GCOST entry update is then propagated using a associative broadcast to all clients in the session who are interested in the canvas object. Similarly if another clients changes the background color of the canvas object, this change in state is captured by the local GCOST and forwarded to the session manager which then updates the client's canvas object to reflect this change.

A. Globally Coordinated Object-State Table (GCOST)

The primary function of GCOST is to ensure that all instances of objects in the collaboration session have consistent state. The table is implemented as a coordinated hierarchical hash table. Each client in the collaboration session maintains a local GCOST, and all local GCOST's in a collaboration session are globally coordinated so that if an entry is exists in more than one GCOST, all its instances are consistent. Coordination is achieved asynchronously using state-based communications. Each GCOST entry is itself a hierarchical structure and corresponds to an object in the collaboration session capturing its current state and interests. The existence of a particular entry in the GCOST implies that the local client is interested in the associated object: the value at different levels of the entry defines its state and the events it is interested in. The formulations of GCOST entries and the construction of the table are described below.

1. GCOST Design

GCOST is structured as a hierarchical and extendible hash table. The highest level of the hash table registers objects of interest to a client; subsequent levels maintain information about the current state, attributes and interests of each object. The lowest level of the table contains lists of "parameter – value" and "parameter – data" pairs. The storage at the lowest level is maintained in dynamically sized buckets which are managed using extendible hashing mechanisms. Extendible hashing [5] is a technique for efficiently managing dynamic databases by merging and splitting storage buckets. As storage requirements increase when new (or larger) entries as inserted, corresponding buckets are locally split. Similarly as entries are deleted or shrink, multiple buckets are merged together. The GCOST hash key space is hierarchically constructed by concatenating keys at each level. Searches into the table are performed by traversing the table hierarchy level by level. The most significant portion of a query index indexes into the highest level of the table identifying a particular object and subsequent portions identify object states. The least significant portion of the key corresponds to a particular attribute of the object.

2. GCOST Coordination

An asynchronous coordination of GCOST entries is achieved using a "publisher – subscriber"[8] interaction pattern built on state-based communications. Each insert or update into GCOST results in a call to the Semantic Event Interpreter that associatively publishes the update. The message semantic-selector is uniquely generated. Similarly an active listener process monitors message selectors and subscribes to messages when the semantically interpreted selector matches an entry in the GCOST. It then updates the local GCOST entry. The update is then forwarded through the session manager to the application object.

B. Semantic Event Interpreter

The semantic event interpreter is responsible (a) for associatively broadcasting or multicasting events over the communication media, and (b) for interpreting incoming messages, corresponding to remote events, for relevance and translating them into local events. Every time a local interaction/event generates an update to a GCOST entry, this update is intercepted by the semantic event interpreter. The update to the entry is encoded into a semantic event message containing a semantic-selector generated from the GCOST hash key. A unique id containing the machine IP address and port of the source is prefixed to the selector before transmission. The message is then broadcast (or multicast) on the communication media. A listener thread in the SEI component monitors incoming event messages. For each incoming message, the SEI extracts the message selector and semantically matches it against the client's profile. This is done by translating it into the corresponding hash key and hierarchically querying GCOST.

1. Event Filtering Mechanism

Application clients in a collaboration session are typically not interested in all objects in the session and hence should only process only the relevant events. This set of relevant events typically changes over time, specially when the clients changes its interests rapidly. For example, a client may iconize a object display or hide it behind another object's display. In this case, the processing of events related to the iconized or hidden object need not be processed. The SEM approach provides an efficient means for locally filtering such events without having to update central rosters. Each event is enhanced with a semantic header that encodes information about the nature of the event and the object state to which it is relevant. The unique id (generated from the machine IP address and port)

prefix enables the event filtering mechanism on the client to determine if the message has been generated by itself and therefore to ignore it or take note of it. When this event message is locally received, the header is interpreted and matched with the object state stored in the GCOST. If the event is relevant to current state of the object it is processed, else it is ignored. We have thus invested the ability to filter the events with the end points rather than the channel, this enables us to cut down on the amount of intelligence to be provided to the channel and deal with the issues at the end points.

2. Event Transmission

The semantic event messages are transmitted over the communication media via broadcast within a sub-net and using multicast beyond a sub-net. The client transmits the event asynchronously to all the other interested clients in the collaboration session, without the any knowledge of the current membership of this group or notion of global time.

C. Archival Server

The archival server is a passive client with the sole purpose is to act as a repository of the current state of all objects in a session so as to enable incoming clients to obtain the current state of an ongoing session. The archival server maintains its GCOST in the persistent storage. It passively listens to every published message and updates corresponding entries in the GCOST. Note that this server is only loosely synchronized with the clients and is only accessed when a new client joins a session.

IV. EXPERIMENTAL EVALUATION

A Java collaboration framework based on the SEM event management paradigm is currently being developed. The framework uses the architecture outlined in Figure 1. A preliminary experimental evaluation of this framework was conducted using 4 clients on two separate subnets. Multiple broadcasts (one to each subnet) rather than multicast was used. Experiments consisted of varying the number of clients and their interests. Setup time for creating the SEM GCOST table and initilizing object states required on an average of 25ms. Updates to the clients local GCOST were extremely inexpensive (2-3ms). Similary, event filtering based on client interests required less than 5ms. Remote updates were sensistive to network traffic and averaged 15ms for updates within a subnet and 45ms across subnets. These values are comparable to typical CCD (effective Client-to-Client delays) of other systems[9]. These number will improve once we switch to using multicasts. Note that the numbers above have been normalized to account for clock skews across the different machines. Finally, the SEM based framework scaled well with increasing number of objects and clients. Experiments also verified that rapidly changing client interests did not effect the event update time. Similar test performed on a centralized server implementation showed that event update time was extremely sensitive to changes in interests and capabilites, and increased as the number of collaborating clients and objects increased. Also, in this case, a faster clients caused the server to choke. Although these initial are envouraging, a more thourough evaluation is required.

V. CONCLUSIONS

In this paper we decribed Semantic Event Management (SEM), a new paradigm for information coordination to support real-time collaboration, where the number of collaborating clients, their locations and interests are dynamic. SEM defines semantically enhanced messages, and uses state-based interactions techniques to communicate and replicate these messages. The approach underlies a Java based collaboration framework. Initial evaluation of the framework confirms that SEM alleviates bottlenecks of traditional, centralized server , collaboration approaches for dynamics groups of clients with changing interest and capabilites. Current effort is concentrated on completing the framework development and more thourogh evaluation.

VI. REFERENCES

[1] B. Bayerdorffer, "*Distributed Programming with Associative Broadcast*," Proceedings of the 27th Annual Hawaii International Conference on System Sciences. Volume 2: Software Technology (HICSS94-2), Wailea, HW, USA, pp.353-362, 1994.

[2] "*Habanero Documentation*," available at www.ncsa.uiuc.edu/SDG/Software/Habanero/

[3] T-Harrison et al, "*The Design and Performance of a Real Time CORBA Object Event Service*," OOPSLA 1997.

[4] "*The Info Bus Specification*," available at java.sun.com/beans/infobus/ibspec.html

[5] H.F. Korth, Abraham Silberschatz "*Database system concepts*," 2nd edition, 1991.

[6] J. Munson, "*Collaboration Bus Infrastructure: Bus Agents*," http://www.cs.unc.edu/~munson/DARPA/busagent.html

[7] S.K. Sundaram, "*A Collaboration–Enabling Framework For Java Beans*,"MS thesis, Rutgers University, 1997.

[8] R.Rajkumar, M.Gagliardi and L.Shua "*The Real-Time Publisher Subscriber Inter-Process Communication model for Distributed Real-Time Systems*," Technical report, SEI, Carnegie Mellon University, 1997.

[9] Shervin S, J.C Oliveira and N.D.Georganas, "***Applet-Based Telecollaboration: A Network-centric Approac***h", IEEE Multimedia, Spring/Summer 1998.P