Experiences with Collaborative Applications that Support Distributed Modeling

James Lee *jlee@cmi.arizona.edu*
Conan Albrecht *conan@cmi.arizona.edu*
Jay F. Nunamaker Jr. *nunamaker@cmi.arizona.edu*
Center for the Management of Information, University of Arizona

Abstract

Awareness of the need for business analysis has grown faster than the evolution of tools to support collaborative development of business analysis models. Involvement of key personnel is important for model accuracy and buy-in, which is not trivial, especially if they are distributed geographically. Traditionally, models have been developed by individuals or small groups because of the complexity of collaborative modeling.

Researchers at the Center for the Management of Information (CMI) at the University of Arizona have created specialized electronic meeting systems tools and methods to support several types of collaborative business models. This paper discusses the creation of a collaborative server created to support the development of distributed, collaborative electronic meeting systems tools. The server and collaborative tools serve as "proof-of-concept" that web-based tools can support collaborative meeting processes in face-to-face and distributed settings. Flexibility integrated into the tools and the server enables them to support a wide range of tasks.

1. Introduction

Researchers in the Center for the Management of Information (CMI) at the University of Arizona have worked for the past several years creating process-specific electronic meeting systems (EMS) tools and methods to support the development of various types of business models. These models include IDEF0 activity models and enterprise data models [1, 3, 5, 6, 12]. The tools and methods were designed to effectively involve users in the development of business models and were geared towards face-to-face settings. With increasing globalization and the use of virtual teams, the need to support both synchronous and asynchronous distributed teams has grown significantly.

The EMS modeling tools are an important part of the Collaborative Software Engineering Methodology (CSEM) [2]. Although there are general-purpose collaborative tools and special-purpose single-user modeling tools available, there are not any tools specifically designed to support *collaborative, parallel* user development of models. This paper focuses on the creation of the CMI collaborative server, the Collaborative Distributed Scenario and Process Analyzer (ColD SPA, hereinafter referred to as SPA) tool, and GroupWriter (GW) which were developed to support this aspect of CSEM. This is only a small part of the research to date on CSEM and collaborative tools. Other papers discuss aspects of CSEM and collaborative tool development not detailed here [2, 7, 8, 9, 13].

2. Background

The CSEM was developed after years of experience using EMS technology to support various requirements gathering activities. It combines advanced group collaboration techniques with the best elements of systematic re-use, data integration, and rapid prototyping methods in order to produce integrated and interoperable systems. It is divided into four phases: Planning, Requirements, Design, and Implementation. The methodology focuses on user involvement throughout the software engineering process and collaborative tools, both general purpose and specialized, are used to support this involvement. CSEM includes a detailed description of each step in the process including roles and responsibilities of project team members. Collaborative meeting tools played a key role in the vision of CSEM and additional tools have been developed since its inception to support various phases of the methodology. For more information on CSEM, see [2, 10].

The development of collaborative meeting tools has been a part of CMI research for more than fifteen years [4, 12, 15, 16, 17]. These tools have evolved over the years from DOS-based, to MS Windows-based, and currently to Java-based, distributed, collaborative tools.

3. CMI Collaborative Server

The CMI Collaborative Server is a second-generation server designed to support the Rapid Application Development (RAD) of collaborative applications. The

first-generation server, a Windows/Delphi-based thin server, supported the initial, distributed applications built within CMI. However, application needs soon outpaced the capabilities of the server, and the need for a new server design became apparent. The second-generation server has been in development since Summer 1999 and is currently deployed as version 3.50.

This section outlines the design and implementation of the new server. A design overview is first presented, followed by specific implementation details.

3.1. Design overview

The design of the server involved the following goals: abstracted collaborative behavior; thin, common client; distributed; real-time; efficient; portable; scalable; standards-based; rapid application development; robust. These goals are described in the following paragraphs.

3.1.1. Abstracted collaborative behavior. The server contains common collaborative behavior found in most applications. A review of historical collaborative applications developed within CSEM and abroad provided a good foundation of functionality that most applications need. Examples of these functions are security, replication, and others. These behaviors were abstracted into a framework and then coded directly into the server.

3.1.2. Thin client. The initial cost of Java clients is the Java Virtual Machine (JVM), which is currently a 5MB download from java.sun.com. Since full-featured client are required for CSEM applications, there is no way around this dependence upon the JVM. However, all framework code beyond the JVM is thin, light, and small. This allows downloaded applets to start very quickly, even on slow modem connections. The entire client application download is less than 100K.

A common client. All applications written to the server framework run within a common client. This common client takes care of the initial bootstrapping process, which includes connecting to the server, establishing event queuing, logging in and sessioning, and application invocation. The client also provides a standard frame that application panels are placed into.

A further advantage of the common client is increased potential for caching. Since most browsers cache files downloaded from the Internet, once a user has downloaded the common interfaces and classes, he or she can run different framework-based applications without needing to download the common client again.

Multiple client environments. Since framework applications are written to an interface which allows them to be placed into a common client, multiple client environments have been programmed. Three environments are currently supported. First, applet-based programs can be hosted by web browsers. The applet performs special functions to allow applets to work through RMI. Second, application-based programs allow for local installations of CSEM applications. Programs can be run as applets or applications with no change in client code. Finally, servlet-based programs use pure HTML pages and forms via the common gateway interface and client web browsers. Minimal client code changes are required for servlet-based applications.

User-interface driven. Since the server provides access to data as well as common collaborative behaviors, client programmers focus almost entirely on user interfaces. The clients download interface classes upon demand; if a user does not open certain screens or dialogs of the application, these class definitions are not downloaded.

In addition, all client programs are based upon Java's Swing architecture. Swing provides standard UI components such as lists, text fields, tables, and graphics. Since javax.swing classes are included in the JVM package, they are already on client machines and do not need to be downloaded (see the use of the plug-in below).

3.1.3. Distributed. The new server supports fully-distributed applications. For purposes of this framework, distributed applications are programs that run off of remote computers with no required local installation. In addition, data are kept on the server or set of servers and not on client machines. The applications are accessible from any client computer on the Internet, providing they have firewall and security access to the applications.

3.1.4. Real-Time. Applications built upon the framework are multi-user applications. They are *real-time* in their ability to replicate data very quickly to all connected clients. Therefore, when one client modifies data on the server, all other clients accessing that same data immediately see the changes on their screens. While data are real-time, users are in control of their applications (the messaging system that manages real-time behavior is described later in this paper).

Locking. Since multiple users access the same data at the same time, the server provides a locking system. Locking is automated as much as possible, but in some cases, applications must specifically ask the server for explicit locks. However, the actual locking system is always controlled and managed by the server.

Security. The server (either through the CMI server, the firewall, the web server, or the EJB server) manages access to data and applications. Clients do not make decisions on security; rather, decisions must be made at the server level. Having the server manage security helps prevent rogue client programs from undermining the security system.

Clients are assigned session tokens upon login that contain their rights. These tokens are good for a period of time (usually 30 minutes) and expire when clients go dormant. All server access is carried out through these tokens, and the server may refuse access based upon the token rights. Sessioning also prevents usernames and passwords from crossing the network unnecessarily.

3.1.5. Portable. The server is programmed in Java for portability reasons. It has been run successfully on Windows, Mac, and Linux platforms. No proprietary ties to any EJB server have been made. While many EJB servers provide proprietary user objects and messaging systems, the framework includes custom services where the EJB standard does not provide. The inclusion of custom objects helps ensure the framework is portable to different EJB servers.

The server is also portable in its data persistence scheme. Since the server is the only portal to data, different data storage routines (relational databases, OO databases, flat file schemes, etc.) Can be plugged into the server with no change in client code. Client programmers see data only as a set of objects.

3.1.6. Efficient. While the server was developed as a prototype, every effort has been made to promote scalability and quick replication. Several caching techniques have been implemented into the data and messaging system, and others are planned for future implementation. These caches do not affect the overall server or client interface, but integrate seamlessly whether or not the cache is enabled.

In addition, replication only takes place between clients viewing the same data. Data updates for parts of the application that are not currently visible (other than for caching) are unnecessary. Therefore, the server keeps track of the location of users and what data they are interested in so it can determine which events to send.

Finally, all processing that can be done by client machines is done there; the server manages only core collaborative functions.

3.1.7. Scalable. The server currently supports small groups of clients (1-50 machines). It is currently hosted on an open source EJB server.

Server code has been written to support n-tiered server architectures. Client machines view their server as the only worldwide server for their data. However, the server might in actuality be one in a large set of n-tiered servers, either in hierarchical or workgroup fashion.

3.1.8. Standards-based. The framework is based upon industry standards, adhering to no proprietary code or interfaces. Specifically, the server and client code is written to Java's Enterprise Javabeans (EJB) standard. EJB provides a standards set of interfaces with which enterprise data can be accessed. Currently, over 30 companies and open source projects are writing products to meet this standard.

In addition, Java's Remote Method Invocation (RMI) is used for client/server communication. RMI is used by the EJB standard and works over CORBA's IIOP. RMI provides schemes for working within firewall situations and allows several different on-the-wire protocols. The common client attempts to connect with three different schemes when it encounters firewall situations, with the final scheme being a "lowest-common-denominator" HTTP-only technique.

Finally, the client-side use of Java's plug-in helps ensure application code is portable across browsers and operating systems.

3.1.9. Rapid Application Development. CMI is a research organization that builds new and innovative prototypes. The nature of this type of development does not always allow for top-down, traditional application development. The needs and requirements of new application are often not known until after the first or second release. (In retrospect, these assumptions are also becoming more and more common in the general computer world.) Therefore, programmers should be capable of programming applications very quickly within the framework. Preliminary data show that once programmers understand the framework API, they can quickly develop robust and efficient collaborative applications.

3.2. The property hierarchy

Properties are the heart of the CMI Collaborative Server. The framework decomposes objects at least one level further than traditional object-oriented (OO) programming. Properties are then assigned to each atomic piece of data; these Properties control the life, client access, persistence, and security of the data. Properties are arranged in the hierarchical fashion typical to OO techniques. The following diagram depicts traditional OO and the framework Properties:

Figure 1 shows a simple user class modeled in traditional OO programming. The main structure is a user object, which contains a name object (recursively holding Strings for first, middle, and last names), e-mail address, and phone number. The user object holds references to

each of the sub-objects it contains, and it manages their existence in memory. Object-oriented programming typically includes all code required to manage these objects.

The framework diagram on the right is a modified version of the previous one. It is intended to show the modifications the architecture makes on traditional OO programming. The data values of First Name, Middle Name, Last Name, E-Mail, and Phone Number still exist.

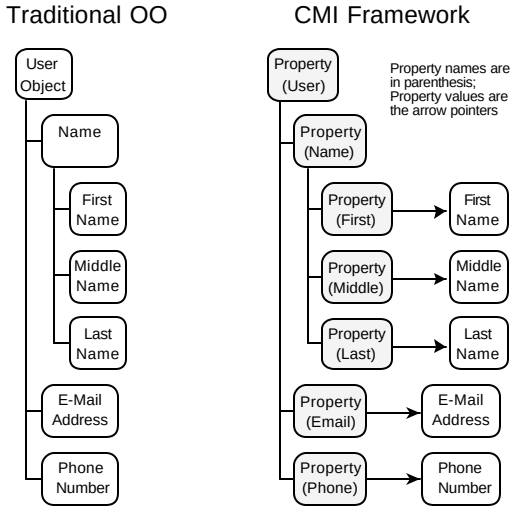

Figure 1. Traditional OO properties and OO properties within the CMI Framework

However, no governing user object is managing them. Rather, each value has a Property object that manages it. Thus, all management, references, and pointers are moved to the property structures and away from the actual data objects.

The top-level property in the diagram above still represents the "User" object in theory. However, no actual data is associated at this level, so it does not point to a data object (although it could in other instances). It is assumed that this top-level property is managed by other, parent properties above it. The entire structure finally ends in a top-level, root property that manages the data of an entire application.

The values managed by the Properties are atomic. While this is the normal case, these values could also be more complex structures, such as arrays, lists, and even large graphs. The granularity of the property structure is governed by each specific application and circumstance.

Each property governs a single piece of data (even that data might be a complex structure). It manages the following on its data:

- **Security Access**: All access to the data behind a property is governed by an access control list in each Property.
- **Messaging**: Property objects are basic collaborative objects. All changes to their data are automatically routed to all interested clients.
- **Viewing**: Data within properties can be viewed in any number of ways. A Property manages the view access to its data.
- **Locking**: Properties can be locked at nine levels of access. This locking gives a client exclusive access to the data of the property.
- **Persistence**: Properties are responsible for saving their data to the database.

Parent and child references kept within each property maintain the property hierarchy. Child properties can be referenced with any number of indices, such as name index, sequential index, etc. The data behind the properties does not need to keep references to its parent or child data because this information is kept at the property level.

3.3. Client viewers

Client applications are mainly composed of Viewers that live within the common client framework. These Viewers are based upon the Model-View-Controller (MVC) architecture. MVC was first conceived at Xerox PARC in the late 1970's. It was often used in the Smalltalk language, but never became popular until the middle 1990's. Part of its recent popularity may be due to its adoption by the Java language architects, who based Java's Swing foundation classes on MVC.

In its simplest form, MVC separates the data (the Model) from the user interface (the View) and allows multiple Views to act upon the same Model. The Controller object manages the interaction and updates between the Model and View. Several forms of MVC are found in the literature, from heavyweight local implementations to lightweight distributed implementations.

A major difference between traditional MVC is that the framework distributes the objects across JVMs. Most MVC applications target local (within process) applications and, consequently, utilize heavy-bandwidth event systems. The framework's MVC implementation is targeted at efficient network usage.

Another difference is that the Properties (Model) normally drive which Viewers (View/Controller) are used. This is opposite of MVC where different Views are windows into non-active data. However, since each Property controls a piece of data, it also dictates the default viewer for that data. This allows applications to

deal abstractly with all kinds of Properties without being tied to certain Viewers.

3.4. Messaging system

The framework includes a messaging system, based upon a modified publish/subscribe model, written specifically for this server. This event system provides the foundation for client/server communication and the replication abilities of the server

3.4.1. An Overview of Messaging. Messaging refers to a system of communication between computers, usually on the application level. Messaging is a foundation piece of any collaborative application, since multiple clients need to share data and screen views. Messaging techniques have existed from the early days of computers, and have evolved throughout their history. The following list categorizes different messaging models (these items are in no way mutually exclusive of each other):

Direct connection. A direct connection is the most basic method of communication. It typically streams bytes socket to socket or utilizes Remote Procedure Call (RPC) to directly call routines on a target machine. It does not use events but sends more basic data structures such as strings or integers. In the instance of byte streams, the source and target machines open a socket connection and send bytes to each other. Case or switch statements on each end parse control characters from the byte streams to determine the specific data and command being sent. Direct connection messaging is synchronous in nature; it requires both the source and target machines to be connected in real-time.

RPC allows source machines to call procedures or methods on target machines. While this is more abstracted than byte streams, the end result is much the same: target procedures are called with basic data structures. RPC has been a very popular method of communication for several decades in many different application spaces.

Shared database. The advent of database systems, particularly relational database systems, provided a more automatic method of collaborative data management. Most database management systems (DBMS) provide multi-user support, two-phase locking, security, and other mechanisms common to collaborative systems. These features are even found in desktop DBMS's such as Microsoft Access and Xbase systems. Since all client applications connect to the same data source, changes to the data are automatically reflected throughout the system.

This method of collaboration has proved very effective for many applications. Shared databases are used in most corporations today for data access throughout the organization. However, the actual update mechanism involved is dependent upon each database. There is no way to ensure that the system uses direct connections, polling, or other mechanisms.

Polling. Polling shifts messaging responsibilities to clients. Client machines are responsible for querying the server at specified intervals for changes to the data. This method does not result in true real-time collaboration since data is only refreshed at a periodic rate. However, with sufficiently short refresh period, real-time collaboration can be approximated. For example, GroupSystems.com's GroupSystems suite of applications refreshes every few seconds. As another example, HTTP (the protocol of the World Wide Web) is a request/response protocol, which requires a polling scheme since only clients can initiate data transfer. While methods to get around this limitation exist, the protocol remains based in polling schemes.

Polling has several disadvantages. First, since clients must initiate data refresh, applications cannot ensure the entire network of clients is up-to-date. This may result in data inconsistency and update anomalies. Second, clients poll at specified periods whether or not the data on the server has been updated. Therefore, this method usually results in unnecessary bandwidth usage and server processing. While this may not be an issue on a local area network, distributed collaborative applications often suffer from polling.

Shared events. Shared Events provide a mechanism to quickly turn an event-based system into a *multi-user*, event-based system. In this scheme, events that are normally relayed to only one client are copied and passed to all clients. Therefore, all clients receive the same data changes and stay in sync with each other. In addition, if UI events are also shared, shared events allow a facilitator-type client to control the user interfaces of all other clients. This method allows for the rapid development of collaborative systems. However, since all clients share all events, the system is not always efficient or optimized.

A system that uses shared events is the NCSA's Habanero environment. “The Habanero framework or API is designed to give developers the tools they need to create collaborative Java applications. The framework provides the necessary methods that make it possible to create or transition existing applications and applets into collaborative applications. Using the Habanero Wizard, developers can easily convert applets by selecting the objects and events they want to share. The Wizard then rewrites the code to take advantage of the Habanero API.” [14]

Publish/Subscribe. This scheme is a more advanced version of shared events. It requires that messages are passed within event objects, which usually subclass an Event super-object. The event system contains routing information that pass event objects from event queues to interested targets. The source usually knows nothing about the targets __ it simply posts events to a specified event queue. Clients register themselves as listeners to different queues. When events are posted to each queue, copies are passed to each listening object. The listeners are then responsible for unwrapping the Event object and dealing with the information as they see fit.

Message queues allow for asynchronous communication. Source applications simply post their events and continue with their execution with little delay. The event queues are then responsible for the method, timing, and route of the events.

While the publish/subscribe methodology has existed for many years, it has recently gained increased popularity. This is in part due to the adoption of message queuing by the Object Management Group , Microsoft , the Java community, and many others. Most current middleware products support some type of publish/subscribe scenario.

3.4.2. Messaging Transparency. The event system should be as transparent as possible. Application programmers should need to know very little about the messaging taking place. They should not need to subscribe or unsubscribe from event queues if the framework can do this automatically. Encapsulating the messaging system within the framework supports the RAD goals of the overall framework.

3.4.3. System description. The CMI messaging system is based upon a modified publish/subscribe scheme. When a server or client initializes, it creates a local event queue and publishes this object in the JNDI. The server or client also starts a thread pool to manage the items in its event pool. One thread is responsible for event forwarding in the routing system; the remaining threads are assigned to events at their destinations.

The framework allows for n-tiered event queues. The source object creates an array of GUIDs describing the route the event should go through. The event queues push the event along this route, popping a route ID off the array at each stop. When only one ID remains, the event queue assumes it is the ID of the destination object. It then forwards the event to this object within the same JVM as the last event queue, where the event is processed.

Routing and subscription. The messaging system uses an abstracted routing system. Routes define the series of stops between server and eventual client. Clients calculate their routes to their server when they log on and pass this route to all Properties to which they connect.

Two potential errors occur when events are propagated through their routes. A null reference is encountered because a) the Viewer no longer exists, or b) the client disconnected during an earlier server session and the server has since restarted, resulting in a null remote reference (since the local stub no longer exists). Second, a connection error could occur because the client disconnected during this server session. In this case the local stub still exists but has lost its socket connection.

Either of these error signals an invalid route: the publishing Property needs to be notified to remove this route from its list of event listeners. Since the remote connections link only JVM to JVM, the event must be propagated backwards through its route to its origin. This is accomplished by setting a rollback flag on the event. When it arrives at the source JVM, the Property is notified to remove the route. Using this mechanism, listener lists stay clean and current. Viewers also unsubscribe automatically during a clean logoff, which is described in the next section.

One reason we describe the framework's messaging system as a *modified* publish/subscribe scheme is that Viewers automatically subscribe themselves as listeners to Properties. Application programmers do not need to explicitly listen to properties they receive data from. The server manages the connections and determines which Viewers to send data to.

Publishing events. Each Property holds an array of listening routes. In this way, each Property is a virtual event queue (see Socket Management below). The destination of each event is not set by the Property. Rather, the Property simply sets the route. When the Event reaches the final event queue in its route, that destination event queue reads the targeted Viewer from the route and sets the actual reference of the destination. The reason for this is that the Viewer is not a remote object and cannot publish a distributed reference to itself. Therefore, the reference cannot be set until the Event reaches the JVM in which the Viewer exists.

Socket management. RMI currently works by establishing a socket connection between a local object and a remote object. Since many Viewers within the system need to connect to many Properties, all available sockets would soon be used up--especially on a server where any number of clients may be connected at a time. Socket limitations are one of the foremost reasons we designed our own event system. To better manage socket resources, only one JVM-wide event queue publishes itself to JNDI. Viewers and Properties actually register their references with their local event queues. Property listener lists store routes rather than actual remote

references. These routes are passed through the messaging system until events reach their destination JVM. The destination queue maintains a list of GUIDs and actual references, which it uses to set the real destination reference of the event.

3.4.4. Events. Since destination objects in the framework are always Viewers, we programmed the events to be "smart". When an event reaches its destination JVM, the event queue references it simply as general framework event. The event queue sets the event's destination reference and then runs its `doEvent()` method, which all framework methods implement. The event acts upon the destination object (opposite of traditional event theory, where the destination acts upon an event). This change significantly decreases bandwidth required for the event system because additional listener interfaces are not required.

Because a Property represents an atomic piece of data in hierarchical format, very few actions can be performed on that data. These include: locking, updating the value, adding children, and removing children. The framework provides four events that match each of these actions. While the framework allows for additional events to be defined by applications, the default four events suffice for most purposes.

4. CMI Collaborative Tools

Previous CMI research led to the development of collaborative modeling tools to support the development of IDEF0 activity models and enterprise data models [1, 3, 4, 6, 11, 12]. These tools were designed to support face-to-face meetings and have been used successfully to gather important requirements information for multiple systems development efforts. The development of the CMI Collaborative Server facilitated the creation of distributed collaborative tools; these types of tools are now the primary focus of CMI collaborative tool development. The following sections describe two new tools that have recently been added to the CMI collaborative toolset: SPA and GroupWriter.

4.1. Collaborative Distributed Scenario and Process Analyzer (ColD SPA)

Previous research in the development of IDEF0 activity models showed that they were very well suited for describing "what" an organization does, but lacked important details such as timing, sequence, and decision logic of activities [1]. The objective in creating SPA was to create a tool that supports process modeling and captures different perspectives such as functional, informational, behavioral, or organizational perspectives. Some of the earliest process models (e.g., data flow diagrams) took a functional perspective. Business process reengineering and other process improvement initiatives have focused on the behavioral and organizational perspectives for modeling general business processes. These business process models include information such as process sequence, decision criteria, and who performs the process. SPA combines these perspectives with an easy to use, highly customizable user interface that supports collaborative, distributed, and asynchronous process model development.

4.1.1. Primary SPA components. The user interface for SPA is composed of three primary components: a hierarchical process decomposition tree, a textual area for describing various aspects of the process, and a graphical process diagram. These three components are used to capture detailed textual descriptions of the process and also to show sequencing and decision logic.

Hierarchical process decomposition tree. The left side of Figure 2 shows the hierarchical process decomposition tree of SPA. This tree structure allows any type of process, whether unstructured, semi-structured, or well structured, to be iteratively decomposed into more fundamental sub-processes. This decomposition may result in many different levels of abstraction for the process or task at hand. Users have the ability to create new process nodes, to move nodes, and to further decompose existing nodes until the necessary level of task detail has been specified. Different users may work on different parts of the process decomposition simultaneously.

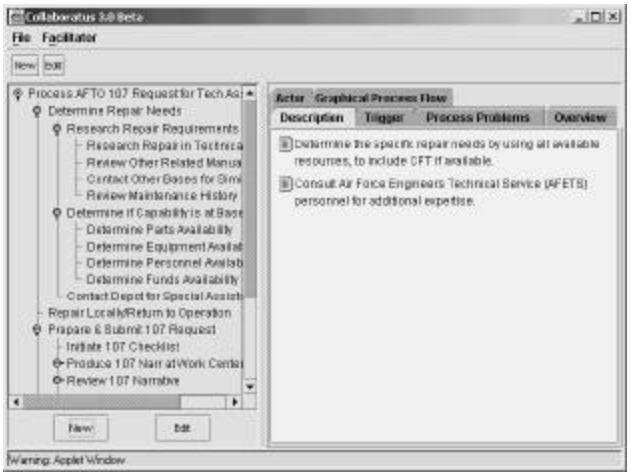

Figure 2. Hierarchical process tree (left) and textual panels in SPA.

Textual process description panels. Once the process has been sufficiently decomposed using the hierarchical

tree structure, specific textual data may be recorded for any process or sub-process contained in the tree (right side of Figure 2). There are two types of data that can be recorded: fixed-field information and freeform text information. Fixed-field information deals with the costs, frequencies, and specific personnel roles that are involved with each process or sub-process. Freeform textual information can be used collaboratively to have users record descriptions of each process, identify required resources to perform each process, list inputs and outputs of the process, etc. SPA has the flexibility to add any free-form textual panel as needed to support the specific requirements of each model.

This approach lends a great deal of flexibility in the use of SPA. Each different category of textual data is captured using a different "tabbed panel." Panels (which correspond to different types of information being recorded about a particular process/sub-process) may be individually turned on or off. For example, all panels could be made invisible except the "Description" panel to help focus the users' attention on capturing only process description data.

Graphical process diagram. The graphical process diagram of SPA allows users to provide a specific arrangement of the process and sub-process nodes that are contained in the hierarchical tree view (see Figure 3). This includes arranging processes into specific sequences or order of operation. Divergence (decision points) and convergence process flows may be indicated using additional graphical symbols. This part of the SPA tool allows the user to indicate which processes occur in parallel, and which processes occur in series.

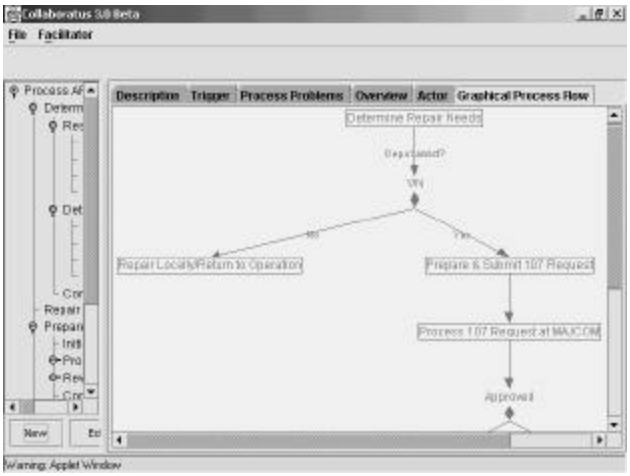

Figure 3. Graphical process diagram view of SPA.

4.1.2. Initial SPA results. The SPA prototype serves as "proof-of-concept" that a web-based tool can be developed to provide support for collaborative process modeling in both face-to-face and distributed settings. Flexibility integrated into the architecture and design of SPA enables it to support a much wider range of process modeling and problem analysis tasks than originally planned, and has, in effect, resulted in creation of a first attempt at a "build-your-own" collaborative tool. The SPA prototype has been used in several modeling sessions, including same time/same place, same time/different place, and different time/different place settings.

4.2. GroupWriter

CMI's collaborative writing research has been designed to advance the understanding of how individuals learn to write together. Collaborative writing requires negotiations between persons in the group as to content and meaning of text. Collaboration affects the allocation and distribution of attention as well as the common ground that is essential for a shared understanding by the group. Social dynamics are altered when using Group Support Systems (GSS) to promote collaborative writing. Studying the use of a collaborative writing tool provides the opportunity to observe the writing process, and reveals much about the special needs of writers.

Collaborative writing is an challenging task. The tools used to facilitate such sessions must be simple and concise; there should be minimal complexity in learning to use the tool with no technological distractions. In collaborative writing, issues of group process, communication, and organizational policies are introduced into the mix. There are various strengths and weaknesses in conjunction with using a collaborative writing tool. Our research approach has focused on collaborative writing, collection and analysis of requirements for government documentation, review of research and commercial group writing products, design and testing of a Java GroupWriter tool, and development of facilitation processes for use of the technology to support the group writing system. GroupWriter is collaborative writing software that was created to improve the process of collaborative writing tasks within organizations.

4.2.1. Primary GroupWriter components. The user interface for GroupWriter is composed of three primary components: a document outline tree, a textual input box for each section of the outline, and an annotation dialog box. These three components are used to capture textual content for each section of the group document and to allow participants to place annotations throughout the document as needed.

GroupWriter document outline tree. The document outline tree of GroupWriter is shown at the left side of

Figure 4. The outline is used in much the same manner as the outline view of MS Word or PowerPoint. Participants use the outline to create the structure of the document being developed. The outline is typically the first part of the group writing process to be completed. Users can drag and drop sections of the outline to arrange and rearrange as necessary. Different users may work on different parts of the outline simultaneously.

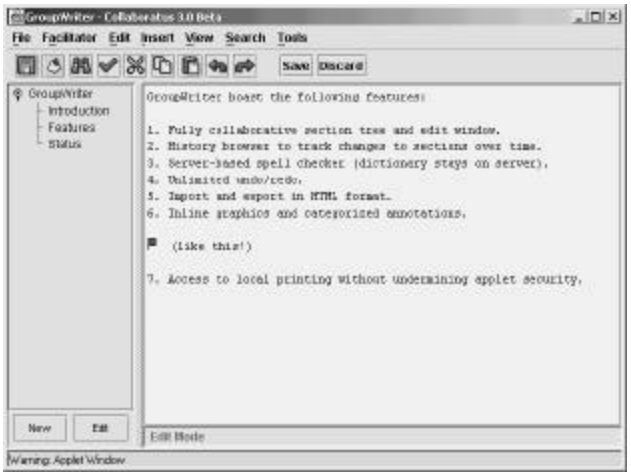

Figure 4. Main GroupWriter screen.

Textual input box. When the outline of the document is complete, users can select any section of the outline and enter the appropriate text (right side of Figure 4). GroupWriter is not intended to be a full-featured word processor. The focus of the tool is to allow users to collaboratively write the document without being concerned about formatting aspects of the document such as font attributes and outline numbering. This allows users to concentrate on what really needs to be contained in the document rather than how the document will look when completed.

Annotations. When creating a group document, it is often necessary to limit which sections of the document can be edited by users. Additionally, once a section has been written and the authors feel that it is complete, it can cause real problems if different users change the text. The GroupWriter annotation feature provides a mechanism for commenting on various sections when you do not want the users to make any changes to the actual text. An example of the GroupWriter annotation screen is shown in Figure 5.

Other GroupWriter features. Though GroupWriter is not a full-featured word processor, there is still a rich set of features built into the tool. Because the documents are often created in a distributed setting, the document owner can control which users can work on which sections and can also lock sections of the document to ensure that no changes are made once the group decides a section is complete. Every time a user modifies a section and saves the changes, the previous version the section is saved and time-stamped. If changes are made to a section of the document and the users later decide that they prefer a previous version of the section, they can review the *Version History* for that section and restore the desired version to the document. GroupWriter allows users to insert graphics into the document and a spell checking utility is also included in the tool.

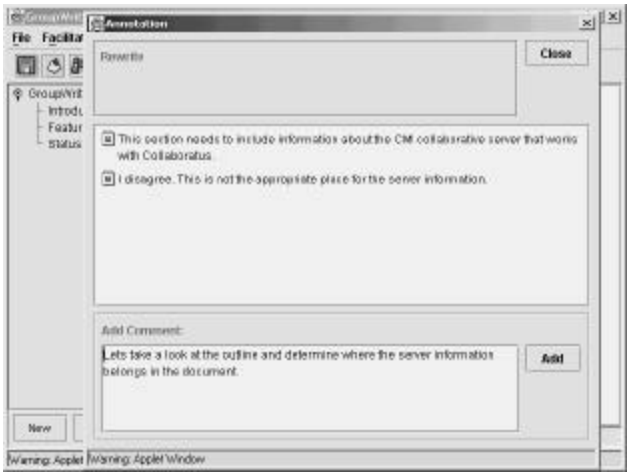

Figure 5. GroupWriter annotation screen

4.2.2. Initial GroupWriter results. Though the Java version of GroupWriter is still under development, previous versions of the tool have been used successfully in meetings with various groups including DESCIM, U.S. Army Training and Doctrine Command, and USA CERL. Meeting participants found GroupWriter easy to use and very useful for both creating and editing documents as a group.

5. Conclusion and Future Research

The CMI Collaborative Server provides common collaborative services to real-time, multi-user, distributed applications on the Internet. These applications are Java-based and portable between environments and systems. Several CMI applications are currently being written within the Server framework, including a collaborative word processor, a process flow application, and an online code reviewing system. The Property hierarchy has proved valuable in supporting the rapid application development of these applications.

The initial SPA prototype served as “proof-of-concept” that a web-based tool could be developed to provide support for collaborative process modeling in both face-

to-face and distributed settings. The flexibility integrated into the architecture and design of SPA enables it to support a much wider range of process modeling and problem analysis tasks than originally planned, and has, in effect, resulted in creation of a first attempt at a "build-your-own" collaborative tool. The next version of the tool and the new version of GroupWriter will build on the lessons learned to create even more robust and useful collaborative tools.

6. References

[1] Dean, D.L.; Lee, J.D.; Orwig, R.E.; and Vogel, D.R., "Technological support for group process modeling," *Journal of Management Information Systems,* 11 (3), 1994-95, 43-63.

[2] Dean, D.L.; Lee, J.D.; Pendergast, M.O.; Hickey, A.M.; and Nunamaker, J.F., Jr., "Enabling the effective involvement of multiple users: Methods and tools for collaborative software engineering," *Journal of Management Information Systems,* 14 (3), 1997-98, 179-222.

[3] Dean, D.L.; Lee, J.D.; and Vogel, D.R., Group tools and methods to support data model development, standardization, and review. In J.F. Nunamaker, Jr. and Sprague, R.H., Jr., (eds.), *Proceedings of the Thirtieth Annual Hawaii International Conference on the System Sciences*. Los Alamitos, CA: IEEE Computer Society Press, 1997, 386-420.

[4] Dean, D.L.; Orwig, R.E.; Lee, J.D.; and Vogel, D.R., Modeling with a group modeling tool: Group support, model quality, and validation. In J.F. Nunamaker, Jr. and Sprague, R.H., Jr., (eds.), *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on the System Sciences*. Los Alamitos, CA: IEEE Computer Society Press, 1994, 214-224.

[5] Dean, D.L.; Orwig, R.E.; and Vogel, D.R., "Meeting methods for use with EMS tools to enable rapid development of quality business process models," *Group Decision and Negotiation,* Forthcoming.

[6] Dean, D.L.; Pendergast, M.O.; and Aytes, K.J., "Computer-supported collaborative modeling : the Enterprise Analysis Project," *ACM SIGOIS Bulletin,* (April, Special Issue on Enterprise Modeling), 1997.

[7] Hickey, A.M., *Integrated Scenario And Process Modeling Support For Collaborative Requirements Elicitation*. Unpublished Doctoral Dissertation, Management Information Systems Department, Tucson, AZ: University of Arizona, 1999.

[8] Hickey, A.M.; Dean, D.L.; and Nunamaker, J.F., Jr., Setting a foundation for collaborative scenario elicitation. In R.H. Sprague, Jr., (ed.), *Proceedings of the Thirty-Second Hawaii International Conference on System Sciences [CD-ROM]*. Los Alamitos, CA: IEEE Computer Society, 1999.

[9] Hickey, A.M.; Dean, D.L.; and Nunamaker, J.F., Jr., "Establishing a foundation for collaborative scenario elicitation," *DATABASE,* Forthcoming.

[10] Hickey, A.M.; Dean, D.L.; and Vogel, D.R., Participative analysis of systems integration opportunities. , *Proceedings of the Americas Conference on Information Systems*. Indianapolis, IN: Association for Information Systems, 1997.

[11] Lee, J.D., *Group data modeling support for business process reengineering*. Unpublished Doctoral Dissertation, Management Information Systems Department, Tucson: University of Arizona, 1995.

[12] Lee, J.D.; Dean, D.L.; and Vogel, D.R., "Tools and methods for group dat a modeling: A key enabler of enterprise modeling," *SIGGROUP Bulletin,* 18 (2), 1997, 59-63.

[13] Lee, J.D.; Hickey, A.M.; Zhang, D.; Santanen, E.; and Zhou, L., "SPA: A tool for collaborative process model development". *Thirty-Third Annual Hawaii International Conference on System Science,* Wailea, Maui, HI: IEEE Computer Society, 2000.

[14] NCSA, "NCSA Habanero," *http://havefun.ncsa.uiuc.edu/habanero/,* National Center for Supercomputing Applications, 2000.

[15] Nunamaker, J.F., Jr.; Briggs, R.O.; Mittleman, D.D.; Vogel, D.R.; and Balthazard, P.A., "Lessons from a dozen years of group support systems research: A discussion of lab and field findings," *Journal of Management Information Systems,* 13 (3), 1996-97, 163-207.

[16] Nunamaker, J.F.; Dennis, A.R.; Valacich, J.S.; Vogel, D.R.; and George, J.F., "Electronic meeting systems to support group work," *Communications of the ACM,* 34 (7), 1991, 40-61.

[17] Nunamaker, J.F., Jr.; Dennis, A.R.; Valacich, J.S.; Vogel, D.R.; and George, J.F., Group support systems research: Experience from the lab and field. In L.M. Jessup and Valacich, J.S., (eds.), *Group support systems: New perspectives*. New York: Macmillan, 1993, 125-145.