Abstract. The objective of this paper is to develop an effective multi-user computer environment supporting design collaboration. As design teams are distributed in different positions in time-space, coordination becomes a challenging problem for any collaborative projects. This paper addresses the coordination problem by modeling the dependencies between activities. The prototype of a future generation of collaborative design systems is presented. It concentrates on establishing a software infrastructure towards a process-centric, asynchronous collaborative environment.

1. Introduction

Computer connectivity and digital communication offer the potential for collaboration between designers who may be dispersed in different geographical locations. Significant efforts have been going on for the past few years to develop an effective multi-user computer environment supporting collaborative design [Maher and Rutherford, 1997] [Eastman (ed.), 1998][Seebohm, T. and Wyk (eds.), 2000]. Many of these efforts have emphasized synchronous communication capabilities, with less concern for the logical process dependencies underlying communication or formal underpinnings. Other work recognizes the collaboration problem at ever higher levels of construction of social relationship [Malone, 1994] [Kunz et.al., 1998] [Huang, 1999]. This approach has emphasized coordination of process dependencies, yet ignoring the semantic data relations that are often required to articulate design processes. It has been argued that the most desirable way to support design collaboration is to develop modular and extensible process modules in a distributed computer environment, which can be composed into higher-level forms for representing design activities, specifying coordination protocols, and reasoning about design progress [Jeng, 1999].
In this paper, I present a web-enabled, multi-tiered prototype system for developing a process-centric, asynchronous collaborative design environment. An important goal is to off-load some of the designers’ responsibilities from recording, disseminating, and tracking coordination information, so that the apparent complexity of coordination and the effort involved in the collaborative process can be reduced.

2. Process Concepts

In order to understand and facilitate the description of collaborative processes, here I make a distinction between different levels of process abstraction, each of which has its own scope of work dealing with particular design issues. As shown in Figure 1, a process can be modeled in terms of four levels of granularity, including organizational, functional, behavior and system perspectives.

![Process abstraction in multiple levels of granularity.](image)

In general, a process can have different levels of abstraction, using both composition-decomposition and specialization-generalization abstraction mechanisms. At one level, we can identify a high-level Phase that is an aggregation of several Tasks. A Task can be decomposed into several sub-Tasks. Orthogonal to the aggregation-composition structure is specialization of the process. At the most general level, a sub-Task can be undertaken by an actor and specialized into an Activity. Some activities may share a similar structure that is generally described in a Task. Activity is event-dependent in that occurrence of an Event would change the performance of Activities. An Activity can be decomposed into sub-Activities. An Activity that cannot be further decomposed becomes an atomic one. An atomic Activity realizes an action that defines a set of abstract methods for operations. When specializing the Activity, an Operation is performed by a computer tool or by human and implements the methods defined in the Action.
This paper focuses on the behavior level with emphasis on some aspects of coordination of activities. Details about the dependencies between activities are addressed in the next section.

3. Modeling Dependencies Between Activities

As an increasing number of intelligent CAD applications are utilized in architectural practice, design collaboration can be characterized by a coherent set of activities operated by the execution of applications. There is strong motivation for reasoning about coordination processes by incorporating the logic of dependencies into a variety of computer applications.

The logic of dependencies can be realized in a more structural way. In one dimension, dependencies can be realized in different level of abstraction. Here I characterize the dependencies between activities into three classes: data dependencies, process dependencies, and temporal dependencies.

- **Data dependencies** are determined based on the notion of data sharing. Data dependencies refer to the input-output relation where one operation maps input data to output data. In the mean while, the operation generated data that are needed as input to the other operations. The input-output relation can be formalized by pre-conditions and post-conditions. A formal definition of data dependencies can refer to [Eastman, Parker, and Jeng, 1997].

- **Process dependencies** refer to occurrence or precedence among events where one event satisfies particular dependency requirements that are assumed by the second one. Two fundamental process dependencies are the precedence ordering of events and the coupling of events. In architectural practice, process dependencies exist in varied kinds of structures, including task composition and state transition. Most process dependencies are socially constructed in terms of responsibility and progress assessment. A further elaboration of process dependencies can refer to [Jeng, 1999].

- **Temporal dependencies** are determined based on the notion of time-sharing. Varied processes can be discrete, jointed, overlapped, embedded, or co-existent in terms of the temporal aspect.

All these together impose a partial ordering on design operations, constraining the sequence in which CAD applications are executed. This paper attempts to model dependencies between activities for use in realizing the logic of coordination. For example, if any activity identifies problems, portions of the design must be refined, requiring iteration of most operations on the changed elements. The refinement process iterates, resulting in a feedback loop in design information flows. In order to reduce the feedback loop, realization of dependencies become important for controlling design information flows. One can define an interactive activity between two individuals, managing the process dependencies between the activities. The interactive activity provides a
coordination protocol that may deal with data transfer, conflict, or any dependencies between a pair of activities.

This section outlines varied kinds of dependencies for modeling the logic of coordination as shown in Figure 2. The work suggests an expanding set of system capabilities provided in future multi-user collaborative design environments. These capabilities can be structured according to the process model presented. The basic level is coordination protocol, relying on various dependencies to facilitate the execution of complex and interleaved design operations and control the transition of the overall design progress. This capability raises several issues in realization of a process-centric, asynchronous collaborative design environment. They include reasoning about design progress, specifying dependencies between activities, and defining coordination protocols for interaction.

4. A Multi-tiered System for Distributed Collaboration

Given the capability-based starting point, I propose that the solution to the collaboration problem is the development of process modules and their dependencies integrated in a distributed computer environment. This section introduces a multi-tiered distributed collaborative system architecture that is made of three layers: application client interfaces, application server, and server database.
application client interfaces through process information can be captured with regard to the execution of computer applications, application server that implements the process logic of designing and provides various facilities for distribution of process management modules, and server databases that provide a range of database capabilities for storing persistent objects and maintaining semantic integrity of design.

The prototype system builds a distributed system architecture that is comprised of three tiers: server database, model server, and application interface. The tier in the bottom is a centralized database that provides a data repository storing a collection of persistent objects. On top of the database is a logical layer that is comprised of a set of independently developed software modules that may be accessed from distributed sites. The modules are a set of distributed objects that can be dynamically loaded, linked, configured, and extended with richer functionality. Finally, the highest level is a layer of application interfaces that allow the application running on top of these layers, using the middle-level network services and the low-level database management system to manipulate persistent objects and perform coordinated activities across the network. The system architecture of the multi-tiered distributed collaborative system is shown in Figure 3.

Figure 3. A three-tier software system architecture consists of application interfaces, model server, and server database
The model server will implement a process model, capable of monitoring change on those design elements. When an element is changed, the model server will be notified of change and, in turn, distributes the change to those clients who have selected their interests. Note that the group space is a place where people can get together virtually coordinating dependencies.

The value of three-tiered software system can be recognized in development of flexible and extensible coordination protocols. Traditionally, we have client-server software applications supporting a multi-user environment, which can be described in terms of two-tier system architecture. The first tier consists of multiple computer applications as clients that generally has a uniform presentation, which we refer to it as a presentation layer. The second tier in the bottom contains a centralized data repository through which the clients share information and communicate with each other. This layer is generally referred to as a server. In the traditional two-tier architecture, the logic of coordination is either embedded in the client applications or built in the server database. It is hardly possible to customize the coordination protocols dynamically without rewriting the applications and the database server. Adding a middle tier isolates the database models from the external information processing of the applications. It also allows us to program the logic of coordination without changing the built-in or embedded protocols. Isolating database connectivity from client applications gives us more flexibility to develop the logic of coordination for an open system.

The proposed architecture is open to a certain extent that one can dynamically extend the software components in response to evolving requirements without rewriting the applications and the database server. In this three-tier architecture, we can add new components in the middle tier and dynamically tailor them to a particular coordination style. Components are referred to as distributed objects that reside over the network, each of which can directly interact with remote objects as if they reside locally. One of the advantages using a three-tier architecture is to provide a flexible way of specifying the logic of coordination and make the development of client programs logically independent from the server program. That is, the middle-tiered components can be developed in parallel based on the coordination requirements rather than being influenced by the computer applications and the database server.

5. Research Prototype: A Web-Enabled Process Tool

Given the principal components by which the multi-tiered distributed collaborative system has been designed, this section provides an overview of the prototype system in terms of the software system architecture and other detail issues about system implementation. I have recently designed and implemented a web-enabled, multi-tiered coordination tool, called Design Back Office. Design
Back Office has been implemented with the capabilities of specifying dependencies, asynchronous communication, shared whiteboard, and 2D geometric editing. The prototype system allows people to communicate and manage inter-dependencies from different positions in time-space. The research prototype attempts to address several key ideas, including reasoning about design progress, specifying dependencies between activities, and defining coordination protocols for interaction.

5.1. SYSTEM OVERVIEW

Design Back Office provides a web interface for accessing a project. After logging on the project, the system automatically pops up a menu containing four menu items: project, workflow, process, and e-mail. The “project” menu item links to the project home page as displayed on the background. The “workflow” menu item allows the user to create, manipulate, and display the activities and their dependencies, which is illustrated in Figure 4. The “process” menu item allows the user to load a graphical viewer. The “e-mail” menu item allows the user to send messages to other collaborators. In general, the user can select different tools provided by the Design Back Office system by pointing with a mouse to either one of the menu items.

![Figure 4. Coordinating dependencies between activities in a web-enabled process tool](image)

The collaborative process is managed and controlled by the process components distributed by the Design Back Office system. A human designer can start an activity by loading an application to its local workspace. The workspace is the virtual place where the application is executed. When loading the application, the process components are distributed by the Design Back Office system, which monitor the significant events with respect to the execution of the application. It also coordinates the dependencies and reports the progress back to the server. The
distributed process components include the activity corresponding to the application and other related coordination protocols. When coordination is required, the human designer creates a group space by selecting an interaction channel. The interaction channel contains the coordination protocol for managing the dependencies between activities.

5.2. SYSTEM IMPLEMENTATION

In the prototype implementation of the Design Back Office system, I use JAVA and the JAVA-centric distributed object protocol called Remote Method Invocation (RMI) as the implementation environment. Using the JAVA/RMI distributed environment, I have built a three-tier system architecture where one can develop the logic of coordination in the middle tier. In this prototype, users can load a graphical viewer, check out the data from the server, and remotely exchange information across the network.

I use Microsoft SQL Server© as the backend database engine. All the entities generated by the applications are stored in the backend database. Each entity is assigned a global identifier (ID) by the backend database when it is created. ID information is used as a handle to identify the object across multiple applications.

The Design Back Office system runs on a Windows NT server and several Windows NT workstations with Internet connection. The JAVA server program and the Microsoft SQL Server© database are operated in the Windows NT server, which is also a Web server that provides Internet access to the database. The client programs are always executed on the client’s Windows NT workstations. The client programs are actually Applets that are special JAVA programs running inside a Web Browser. With Internet access, all the process components can be dynamically loaded and executed in a local site.

The idea is that one can remotely load the process components and link them to local CAD systems or applications. This would form the first tier of the proposed architecture. Here the Design Back Office system provides a graphical viewer to interface with the local CAD systems or applications. I believe that it will become possible to advance the graphical viewer as an extension of CAD systems, in an easy-to-use manner, allowing the process components to be linked with the CAD systems.

6. Summary and Future Work

Coordination is a challenging problem for any collaborative system, but for design of open and distributed design environments, it is particularly so. In this paper, I propose that the solution to the coordination problem is development of modular and extensible processes and their dependencies integrated in a process-centric design environment. In such an environment, processes can be
defined as distributed components scattered across organizational networks, corresponding to organizational tasks, designers’ activities, and many other operations of computer applications. Distributed components can be dynamically loaded from the project server that links the local applications to the backend database. This allows defining and articulating varied processes and their dependencies in a partial order. The backend database serves as the common repository of dependency rules that monitor and coordinate the execution of the applications in a remote manner. Collaboration relies on the project server and the backend database, together serving as the process manager who represents plans for distributed actions, monitors state-dependent operations, and, in particular, coordinates actions in varying degrees of concurrency. This approach supports an open and distributed design environment, allowing new distributed process components to be developed by independent parties, with existing organizational policies for different projects, management assistance for cross-disciplinary use of CAD systems and so forth, allowing improvement of collaboration quality.

The paper presents an overview of the research prototype allowing distribution of persistent objects across Internet. Clearly, the research prototype would involve changes in the working style of designers. Usability testing will be undertaken in a digital design studio for developing varied collaboration scenarios. Some important aspects of implementing the process-centric, asynchronous collaborative design system require more integration efforts, involving process/product exchange standards and articulating product models to process information.

Acknowledgments

This paper is part of my doctoral dissertation in Georgia Institute of Technology. I would like to thank my thesis advisor Professor Chuck Eastman for his invaluable guidance and support.

References

