A Relational Database Synchronization System for the Process Handbook

by

Edward I. Hsu

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirement of the Degree of

Bachelor of Science in Electrical Engineering and Computer Science and

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 1999

[June 1999]

Author_____
Department of Electrical Engineering and Computer Science
May 19, 1999

Certified by_____
Professor Thomas W. Malone
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

A Relational Database Synchronization System for the Process Handbook

By

Edward I. Hsu

Submitted to the
Department of Electrical Engineering and Computer Science

May 18, 1999

In Partial Fulfillment of the Requirement of the Degree of
Bachelor of Science in Electrical Engineering and Computer Science and
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

The functionality of the Process Handbook has grown rapidly in recent years. The number of users using the Process Handbook application has continued to grow as well, as development of an official process modeling database continues at CCS. The synchronization of client and CCS research databases must be maintained to allow clients to benefit from CCS research of these new process models. This thesis explains the design and implementation of a server-side transaction recorder and playback system for the Process Handbook. More specifically, the new design adds a new file type to the Process Handbook to record and playback transactions, namely the Process Handbook Transaction Log, or PTL file. The Record and playback system is very modular and very extensible, allowing for future functional expansion. This most recent functionality expansion allows user groups to receive and send updates for their Process Handbook database, facilitating the communication of new process modeling innovations.

Thesis Supervisor: Professor Thomas W. Malone
Title: Patrick J. McGovern Professor of Information Systems

Acknowledgments

I would like to thank many people for their support and guidance throughout my years at MIT, especially during my graduate year. First, I would like to thank Professor Thomas Malone, for the inspiration and opportunity to work on this project. My most sincere thanks to John Quimby, who was more than just a thesis supervisor. Thanks for his personal advice on my career, and thanks for his confidence in my abilities. He guided me and provided help with technical and design problems. In addition, I thank him for tolerating my running to his office asking many "quick questions" a day. I would also like to thank everyone at the Center for Coordination Science, who made my work there not only interesting but fun and memorable as well.

My deepest thanks to my family, especially my mother (her guidance and love are the primary reasons for my academic success) who helped me maintain focused during my years at MIT. Thanks for their support, confidence, and love. Thanks to all my friends, particularly Felicia, Leaf, and Janet for their encouragement and support when I needed it throughout my years at MIT.

Table of Contents

Chapter 1:
Introduction

The Process Handbook project at the MIT Center for Coordination Science involves collecting examples of how different organizations perform similar processes, and organizing these processes in an on-line "Process Handbook" (Malone, et al., 1997). The Process Handbook is intended to help people: (1) redesign existing organizational processes, (2) invent new organizational processes, (3) learn about organizations, and (4) automatically generate software to support organizational processes. The methodology used in the Process Handbook is semantically very rich. It allows the users to represent and analyze complex processes.

Currently, users groups of the Process Handbook exist in several institutions. Each of these groups has process models stored in their own copy of the Process Handbook database. As new process models are developed in each of these places, it is often useful to share them with other research groups, and merge them into new, combined databases. The current way for process models to be exchanged is with the Process Interchange Format (PIF) application.

The PIF application exports process models to files that can then be imported to other databases. Although in theory this method works well for transferring processes from one database to another, the application has not been very successful. Reasons for its limited success include its limited functionality. The PIF application is declarative in its transfer of process models, and does not record sequences of transactions. For example, deletions are not recorded by PIF. In addition, a series of small changes in different parts of the database is very awkward to transfer. The limitations of this current system is further discussed in section 4.2.

These leads to the problem of how to synchronize, or update the databases of clients to

include process models that are continuously being developed at CCS and vice versa. Database synchronization software exists commercially, but the Process Handbook database requires that the client database be modified while still being updated with Process Handbook Process models from the official CCS database. This requires the development of our own synchronization system.

Members at the Center for Coordination Science, including Prof. Thomas Malone, and CCS research scientist John Quimby, took an active part in defining the motivation and requirements of such a database synchronization system. Statements such as "we" in this thesis, I refer to either John Quimby, and/or Prof. Malone, in addition to myself.

In the following chapter, background of coordination science and the Process Handbook application developed at the Center for Coordination Science is discussed. Chapter 3 will describe the goals to be achieved with this new system. Design overview of the system is in Chapter 4. This chapter is useful for developers for understanding the architecture of the application expansion.

Developers interested in expanding the functionality of the Process Handbook Transaction Recorder and Playback System would find chapter 5 particularly useful. Aside from detailed specifications of the implementation, also included are command hierarchy and frame diagrams that clearly illustrate the architecture of the system. Chapter 6 evaluates the achievements of goals from the implementation, and recommendations for additional future expansions are suggested in chapter 7.

Chapter 2:
Background

2.1 Coordination Theory

Coordination science forms the mainstay of the Process Handbook. To understand the methodology of the Process Handbook, it is important to understand the concepts of coordination science. Coordination theory allows the Process handbook to be a very effective tool for modeling and inventing new processes.

Coordination theory is an emerging research area that deals with the interdisciplinary study of coordination. The research in this area uses concepts from such varied fields as computer science, organization theory, operations research, economics, linguistics, and psychology (Malone et. al. , 1999). Coordination means being able to get different processes to work together. The more precise definition of coordination is managing dependencies among activities. In the context of coordination science, activities and processes are synonyms.

The Process Handbook uses this more precise view of coordination to model processes. Processes are decomposed into sub-processes and dependencies between these sub-processes. Section 2.2 gives a more comprehensive treatment of this topic. The dependencies between these sub-processes can be managed by processes that manage activities (managing activities). The ability to view and modify the managing activities for these dependencies gives the users of the Process Handbook the power to invent new processes. It also gives them the ability to more easily understand existing processes in detail and improve on these processes.

Processes can be invented using the Process Handbook through four steps. First, the higher level goals must be defined. Next, these processes are decomposed into sub-processes. Then the dependencies between the various sub-processes are specified. In

the final step the managing activities for these dependencies are specified. This last step, the most important, specifies the nature of coordination between these sub-processes.

2.2 The Process Handbook – Ports, connectors, etc.

Functionality of the Process Handbook has grown significantly over the last five years. Development has progressed to the stage where user interaction efficiency has been addressed. The enhancement for an user-friendlier interface has lead to newer objects that are primarily for efficiency of users' interaction with the Process Handbook. These objects include attribute-type, Navigational Node, and much more. This section describes the activities, dependencies, ports, connectors, attributes and bundles, including the specializations and decompositions of these objects, which are the fundamentals of the semantic model of the Process Handbook.

2.2.1 Decomposition of Activities

The decomposition of activities allows users to view activities at many and varying levels of detail. Users can choose to examine only the higher level activities, or view their breakdowns to very basic activities. In theory, any activity can be decomposed infinitely, but users of the Process Handbook add activities to a level they feel relevant and useful.

Figure 1 illustrates a very simple decomposition of the activity *Sell Product*. The decomposition of this activity consists of activities such as *Identify Potential Customers* and *Inform Potential Customers*. However, it is obvious that Figure 1 cannot be used as a complete specification for the *Sell Product* process. The specification cannot be complete without dependencies. For instance, we need to identify potential customers before we can inform them and we need to obtain the order before we can receive the payment.

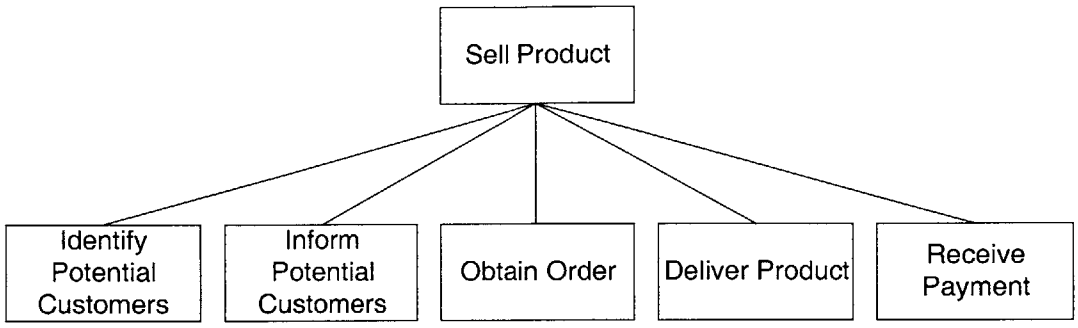

Figure 1: The decomposition of Sell Product (adapted from Farooq, 1997)

2.2.2 Dependencies and Ports

Dependencies represent the interaction between different activities, and form an essential part of the Process Handbook methodology. Figure 2 illustrates the three basic kinds of dependencies: *flow*, *sharing*, and *fit*. Flow dependencies arise when a resource produced by one activity is consumed by another activity. Sharing dependencies arise when a single resource is consumed by multiple activities. Fit dependencies occur when multiple activities produce a single resource (Malone, et al., 1997).

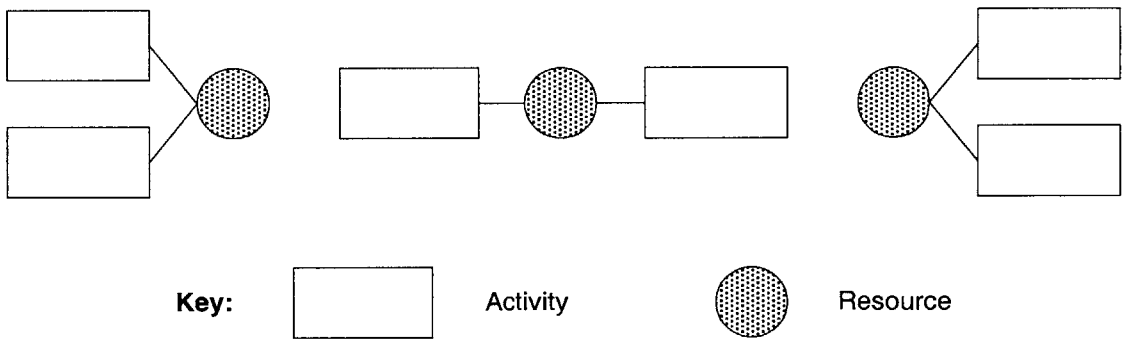

Figure 2: Three basic types of dependencies among activities (adapted from Zlotkin, 1995)

Figure 3 shows a simple flow dependency commonly; an example commonly used at the

Center for Coordination Science. The figure shows the decomposition of the process *Make Coffee* into two sub-activities: fetching the coffee beans and brewing coffee. The dependency is important as it specifies three properties. First, the beans have to be fetched before the coffee can be brewed; this is a prerequisite constraint. Second, the beans have to be transported to the place where the coffee will be brewed (accessibility constraints). Third, the brewing mechanism should be able to use the coffee beans that are fetched (usability constraint). This simple dependency example has only one producer and one consumer activity. However, dependencies can become quite complicated when there is multiple producer and consumer activity interaction.

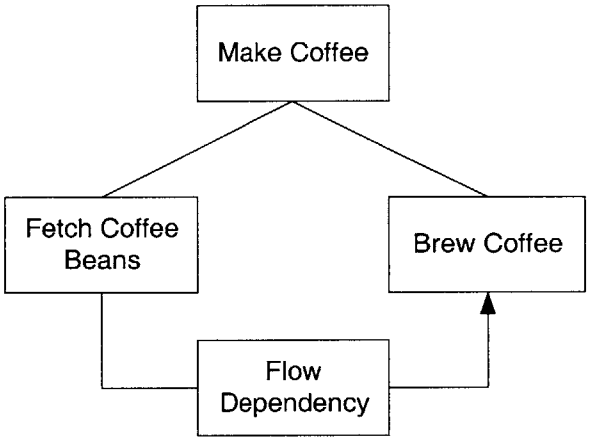

Figure 3: A simple flow dependency.

In the Process Handbook, the activities and dependencies interact with each other through their ports. Ports can be thought of as inlets and outlets for an activity or dependency. An activity or a dependency can have any number of ports. In the *Make Coffee* example, the activity *Fetch Coffee Beans* would have a port that would be the outlet for the coffee beans that have been fetched. Similarly, the activity *Brew Coffee* would have a port through which the coffee beans are consumed. The *Flow Dependency* would also have ports that are used in the mediation of the coffee beans that flow from *Fetch Coffee Beans* to *Brew Coffee*. (Farooq, 1997)

Coordination can be defined as managing dependencies among activities in the Process Handbook. The users of the Process Handbook can set managing activities for the dependencies represented in a decomposition. The ability to replace existing managing activities with other managing activities from the repository of activities in the Process Handbook greatly enhances its power for analyzing processes.

2.2.3 Specialization of Activities

In the process handbook, activities span two dimensions: the dimension of generality and the dimension of detail. Figure 4 illustrates these two dimensions.

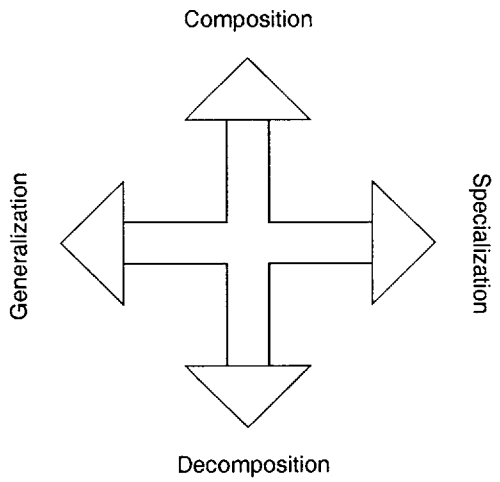

Figure 4: The Dimensions for Expressing Activities

Figure 4 is analogous to a compass in the plain of activities. Here, the North-South dimension represents the composition and decomposition of an activity. For example, the activity Make Coffee can have Brew Coffee in its decomposition. At the same time, Make Coffee can itself be in the decomposition of Prepare Breakfast. People are generally accustomed to thinking about activities mainly in the terms of the Composition-Decomposition (or North-South) dimension.

The Process Handbook methodology also allows characterization of activities in the Generalization-Specialization (West-East) dimension. Combined with inheritance (discussed in 2.2.6), these characterizations of processes make the Process Handbook a very powerful tool. An activity in the Process Handbook can have many specializations and generalizations. For example, Make Coffee can have Make Mocha Frappaccino as a specialization and have Make Beverage as a generalization. All activities in the Process Handbook are part of a specialization hierarchy, which has very generic processes at the top level and increasingly specialized processes at lower levels. Figure 5 illustrates how the two dimensions work together in enhancing the power of the Process Handbook.

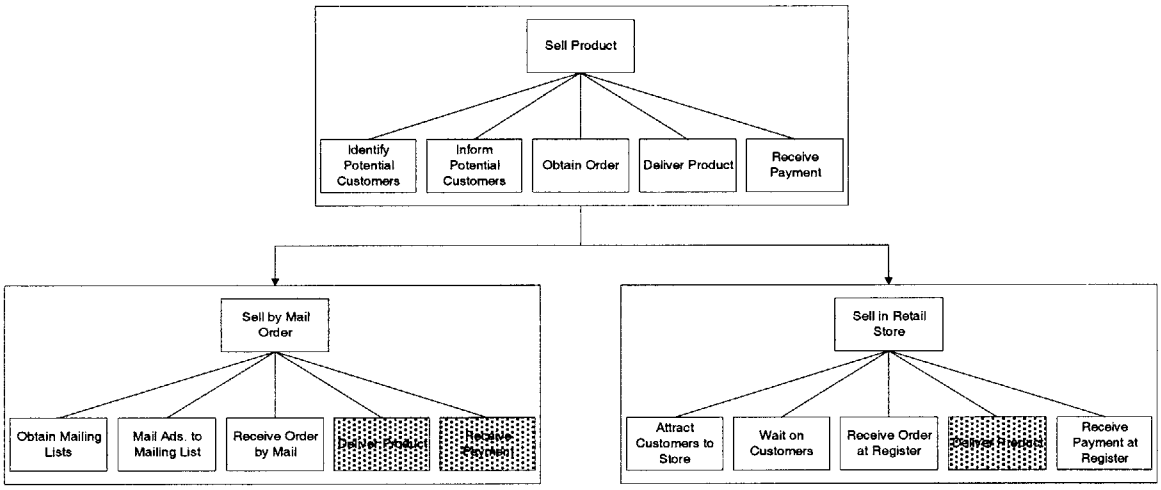

Figure 5: The representation of three different sales processes. Sell by Mail Order and Sell in Retail Store are the specializations of the generic process Sell Product. Shaded sub-activities are inherited without change. (Adapted from Malone, et al., 1997)

The two specializations of the *Sell Product* activity inherit its decomposition. Some of the sub-activities of *Sell by Mail Order* and *Sell in Retail Store* have to be further specialized to fully express these activities. For instance, the activity *Identify Potential Customers* is replaced by a more specialized activity *Obtain Mailing Lists* in the decomposition of *Sell by Mail Order*. Other sub-activities might not need to be changed. The shaded sub-activities in Figure 5 are inherited without any change. Besides the advantages of combining the specializations and decomposition, this approach allows

conciseness of representations and generative representations (Malone, et al., 1997).

2.2.4 Attributes

All the entities (activities, ports, dependencies, navigational nodes, attribute-types, bundles, and resources) in the Process Handbook have attributes. Some of these attributes distinguish the entity from the other entities in the Process Handbook. Examples of these attributes include *Name*, *ID*, *PIFID* (Process Interchange Format ID), and more.

Some attributes are system-generated (the attributes that uniquely identify the entity are among these attributes). For instance, the attribute Name is generated by the system for each newly created entity. The user also has the ability to add any attributes that he or she might find necessary. For example, the user might want to include the names of the people who are interested in a particular activity as an attribute of that activity.

2.2.5 Bundles

The specializations of a process are arranged in bundles. Each bundle contains a set of alternative processes. Bundles significantly enhance the power of the Process Handbook. Figure 6 shows an example of grouping specializations in bundles. The specializations of the activity *Sell Something* are arranged in two bundles (shaded boxes). It should be clarified that the bundles are not a part of the specialization hierarchy. The bundles are actually attributes of the activity *Sell Something* and the activities in the bundles are attributes of those bundles.

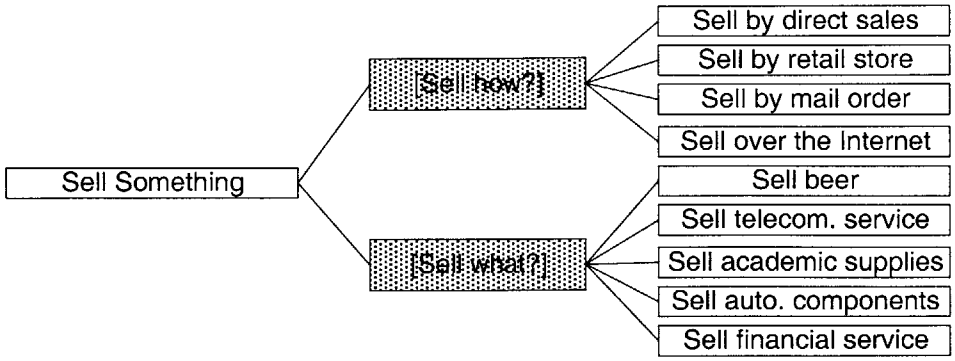

Figure 6: Specializations of the activity Sell Something arranged in the bundles Sell how? and Sell what? (Adapted from Malone, et al, 1997)

Bundles are primarily used in two ways. First, they are used for comparing the alternatives. These comparisons are facilitated by trade-off matrices that compare the important characteristics of the alternatives (these trade-off matrices are generated by using the information stored as the attributes of the bundles). For example, an interested user could match the cost of selling by direct sales and the cost of selling by retail store (if cost of selling is one of the characteristics compared in the trade-off matrix for the *Sell how?* bundle). Second, the bundles are used for restricting certain kinds of inheritance. Alternatives in a bundle can only inherit alternatives from other bundles. For example, someone selling beer should be presented with the alternatives for direct mail, retail store, and selling over the Internet, but it does not make much sense for him to be presented with alternatives of selling automotive components or financial services.

2.2.6 Inheritance

The inheritance in the Process Handbook considerably increases its power and value to users. The specializations of an entity inherit its attributes and decomposition. The Process Handbook borrows the concept of inheritance from object-oriented systems. Whereas, traditional object-oriented systems have a hierarchy of objects where the specialized objects inherit the functionality of the more general objects, the Process Handbook has a hierarchy of both verbs (activities) and objects (the resources in the Process Handbook specialization hierarchy are actually objects) that behave in a similar

fashion. It is quite important to understand the representational semantics of the inheritance of attributes and decomposition in the Process Handbook.

Chapter 3:
Goals of the New Synchronization System

3.1 Updating Client Process Handbook Databases

New changes are always being made in the Process Handbook database as coordination science and process modeling progresses in the Center for Coordination Science (CCS). In order for sponsors and users to benefit from these new changes, they would need to incorporate these changes into their own local Process Handbook database. A means is needed to deliver the new information in the CCS database to other users of the process handbook.

A new database synchronizer is needed because Process Handbook users will also continue to modify their own PH database for their process modeling needs. New changes from a CCS PH database would allow users additional tools for modeling or inventing new processes, but we cannot require that users rebuild their PH database every time a newer PH database is created at CCS. An analogous situation is development software vendors and their users. As new functions are created within the programming language software development package, users should have access to these additional functions, but they should not be required to modify any of their existing code. In addition, distributed users may make changes that CCS wants to incorporate in future versions of the Process Handbook. It is desirable to have an easy way of merging these changes into the main CCS database.

One means of delivering this new information is with either a "diff" utility to compare two databases, and make the necessary changes to the client database so that it incorporates all the information from the CCS database. Another approach would be a transaction recorder on the CCS database. The changes made in this database would be recorded in a transaction log and sent to Process Handbook users. The transaction log would then be played back on their database.

Since entities make up the mainstay of the process handbook, the record and playback system will be based around the entities as well as their Process Interchange Format ID's (PIFID). The transaction log approach was chosen for implementation because of its benefits. These benefits covered in Chapter 4.

3.2 Database Integrity

In many cases, it is important to maintain the integrity of the database in this transaction recording system. During playback, we cannot expect the client user to monitor every change made in the update. As a result, if the integrity of the database were compromised, the user would not immediately know. This can lead to many problems. The playback system must have all the information necessary to make each transaction playback call flawless and no different than calls made from the PH_TNG editor client during normal editing. This property implies several requirements that must be met in the Process Handbook transaction record and replay system.

The goal of the recording and playback system is to transfer all the changes made from a source Process Handbook database to client databases. To make this possible, the first requirement that must be met is that replay of transaction logs on previously identical databases should yield identical databases, given playback is successful and the transaction recording was not interrupted.

An additional requirement in our system is that if a transaction cannot be recorded, or there was an error in recording, the recording system should tell the user. This gives the user the option to cancel or undo the transaction. If this were not implemented, then the recording would not be representative of the changes that have occurred in the source PH database. In this case, playback would not yield an identical database.

Another requirement is that transaction recording be made only when that transaction is successful. If errors occurred in a transaction on the source PH database, but the transaction was recorded in the transaction log, this will lead to problems later on. For

example, the same error may or may not occur in the client database during playback. If a successful transaction is made in the client, we have lost the objective in our design, since this transaction was not successful in the source PH database. Various permutations of similar problems must also be under consideration.

3.3 Maintainability and Maintaining Abstraction

3.3.1 Non-Intrusive Addition

To make the transaction recorder and playback system a success, minimum changes should be made to the existing Process Handbook. In addition, this system should not place any requirements on the design of the process handbook in the future, and maintain a low profile in the application.

The recording system should be as least intrusive as possible. Calls made from the existing process handbook to the transaction recorder modules should be minimal. And only after each successful transaction, should the transaction parameters be recorded. To achieve this goal, we could need to record and playback these transactions at highest possible level in the Process Handbook server's architecture. The transaction log should also save at least the minimum amount of parameters possible to replay the transaction.

3.3.2 Modularity of Design

The design of the Process Handbook transaction recorder should be modular, to allow for easy maintenance of the system. To achieve this goal, we require that there be independence of record/playback modules and file output modules. Simply put, the functions that are responsible for writing and retrieving data from the transaction log should be independent of the playback and recording of the transactions. This allows for the transaction log format to be changed or modified without any other changes to the transaction recorder.

Another requirement of our system is that record and playback be dependent for each

transaction, but independent across each transaction. The former is simple, since replaying a transaction relies on the data on the transaction log from recording the transaction. However, different transactions require different arguments to be used in its function call. These arguments can be different in number, type, scope, and more. The recording and playback scheme of different transactions should be independent from each other in design.

3.3.3 Server-Side Implementation

A final technical requirement is that the record and playback system must stay mainly on the server side. User interfaces may be made in the existing PH_TNG (Process Handbook, the Next Generation) client editor of the Process Handbook to handle input and notify the user of the recorder and playback system's status. However function calls made in the recording and playback must be made on the Process Handbook object server only, transparent to the client.

The reason for this abstraction is that in the future, other versions of PH clients will be implemented. Their transactions may also need to be recorded. These additional clients include autonomous software agent systems that will also manipulate processes and entities modeled in the Process Handbook database. This design decision was made after discussion of design with research scientist John Quimby at CCS.

3.4 Forward Compatibility

The Process Handbook has been under development and redesign for more than five years. Forward compatibility and function expandability is certainly a requirement since entities, relations, and connectors may evolve in the future.

Recording and playing back of transaction will certainly remain a requirement. Although transactions may be made differently in the future, the transaction will still need to be record.

Another dimension of forward compatibility in the recorder and playback system is that not only may existing transactions be changed, but new transaction types may be created. These would need to be recorded as well.

3.5 Other Considerations

Other design considerations include how the user interface would be structured. The interface needs to be intuitive, and simple, since no help file will be available on line within the application.

The question of whether or not playbacks should be recordeable comes into consideration. Ideally, if multiple layers of updating must be made across a hierarchy of databases then recordability of playback transactions should be required. However, since the main objective of this design is to allow clients of CCS and users of the Process Handbook to receive updates from CCS only, playbacks will not be recorded in this implementation.

However, although not implemented, out design would allow for easy expansion to enable playback recording. The scope of this application design and development project is to create a reliable way to distribute changes made in the Process Handbook database at CCS to other Process Handbook users. Recorability of playback transaction will therefore be deferred for future work. However, modularity in implementation of the record and playback system would allow for easy expansion of the system to include that functionality.

Chapter 4:
System Design and Overview

Overview

This chapter discusses the logic behind the design scheme chosen for the database synchronization system. And gives an overview of the technical aspects of the Process Handbook Transaction Recorder. Section 4.1 discusses the reasons for choosing the implemented design scheme. Section 4.2 gives an overview of the implementation. The steps of recording and playback is described in section 4.3, and details of recording and playback are discussed in section 4.4 and 4.5 respectively. Section 4.6 discusses the Process Handbook Transaction Log file (PTL file).

4.1 Design Strategy

As mentioned in Chapter 3, two designs were considered for implementation in the Process Handbook database synchronization system. Those were the transaction recorder and the "diff" utility. The "diff" utility would work by comparing two databases. It would then make changes to one or both database(s) so they would be identical, and each retains information that was in either of the databases.

The transaction recorder works by simply recording transactions in to a text log file. To update another database, the client copy of the Process Handbook must be able to playback the log file and execute transactions identically as the source database Process Handbook application.

Both designs have their advantages and drawbacks. The "diff" utility would be useful because it offers the ability for PH databases to be updated up the information distribution hierarchy. For example, CCS can get updates of client Process Handbook users without the client having anything other than the current (pre synchronizing-

capable) process handbook application.

Drawbacks of a "diff" utility are many. One problem with this approach is that the Microsoft Access database used in the Process Handbook gets quite large, currently with over five-thousand entities and ninety-thousand attributes, and over 20 megabytes. Since manipulation of entities in the Process Handbook can get quite slow, in all but the fastest of today's PC's, the synchronization system would take a long time. This is especially true in synchronization given that the "diff" utility had to juggle two PH databases simultaneously. In addition, even if only small modifications are needed for updating, the entire database must still be compared to ensure synchronicity

A transaction recorder on the other hand, does not have this limitation. Implementation of a transaction recorder simply adds minimal complexity to the existing Process Handbook application. A transaction recorder and playback system would be very efficient for small or large updates. In addition, distribution of updates would also be very convenient, since transaction log files are ASCII text files that don't get nearly as large as a Process Handbook database. One other benefit of choosing the transaction recorder is that it allows users to monitor changes as they are made to a PH database. Thus, a transaction recorder allows for a snapshot of the PH database in time.

A limitation of the system, however, is that it cannot handle the synchronization of two databases once they are already different and there was no transaction log that records effectively the difference of these database. Only a "diff" utility implementation would make this possible.

However, a final benefit of using a transaction recorder is that it can actually be a subset of a "diff" - type utility. For example, an application can later be developed to generate a transaction log based on two different databases. This is discussed in detail in Chapter 7.

Because of the benefits of the transaction recorder scheme, it was chosen for the design of this process handbook relational database synchronization system.

4.2 Advantages Over Existing Systems

The Transaction Recorder and Playback System offers many advantages over existing Process Handbook process interchange systems. The Process Interchange Format (PIF) application currently is used to export and import processes from one Process Handbook database to another. There are limitations to this system, however.

One limitation is that the PIF application is declarative in its transfer of process models, and does not record sequences of transactions. This leads to problems when updating databases. For example, PIF would not show the replacement of a dependency by a specialization. Such changes are important in process modeling, since it can give users insight why certain processes have evolved to their current structure. Transaction recording allows changes that cancel out other changes to be recorded and examined for future analysis.

In addition, one benefit that the Transaction Recorder has over PIF is that a transaction log can store changes to a database. And if somehow a database error was created through legitimate and successful higher level calls, Process Handbook developers can more easily identify what functions cause the problem. This allows bugs within the Process Handbook to be isolated and dealt with.

4.3 Overview of Entity-Based Design

Once the decision was made to create a transaction log recorder, the next design decision was to find a way to record these transactions. The mainstay of the Process Handbook database are the entities that exist within the database and the relations between them. The object server of the Process Handbook presents these entities as objects. Each transaction in the Process Handbook translates to functions performed on or by these entities. Since these function calls are made by the PH_TNG editor client, they are the highest level calls possible within the object server to record transactions.

In the process handbook, transactions are made by the user by selecting an entity and selecting the desired transaction. To replay transactions, the playback system executes them in the same way. The playback system will first get a pointer to the entity and then call a function of that object, passing in the required arguments to make a successful transaction.

Several forms and modules, and class modules have been added to the Process Handbook in the development of the transaction recorder. An overview of these new additions and their functions are described in Chapter 5.

4.4 Steps of Recording and playback

The transaction recorder is initialized immediately after the user has chosen a Process Handbook database to be opened. Just before the database is opened however, a call to initialize the transaction recorder is made. This sets all the parameters necessary to commence recording. Aside from internal variables, a corresponding PTL filename must also be supplied. If a corresponding PTL is not found for the PH database being opened, the user is prompted to choose a filename he/she chooses to start recording. The PTL file is described in detail in section 4.6. Each step of the transaction recorder are outlined below:

1. Examines the PH_database filename, checks for corresponding PTL File for database.
2. Asks user to use PTL file (if found), or create a new one
3. User turns on transaction recording.
4. Transaction recorder initialized.
5. Database opening is recorded in PTL file.
6. Recording Commences
7. Each transaction also includes the contact making the transaction.
8. If recording is not possible, the user is notified.

9. When database is closed, an entry is made on the PTL file of the closure.

For playback to occur, transaction log recording must be turned off. Since this is possible only when a database is being opened, and since each execution session of the Process Handbook opens only one database, recording must remain off as soon as the application is started. The user can then choose to replay a PTL file by pressing the *transaction handler* button. After the file is selected, replay of the PTL file begins once the PTL is determined valid. These steps are summarized below:

1. Examines the PH_database filename, checks for corresponding PTL File for database.
2. Asks user to use PTL file (if found), or create a new one
3. User chooses not to record transactions.
4. User chooses a PTL file to replay.
5. Transaction Replayer verifies if the PTL file is valid.
6. Playback commences if PTL file valid.

4.5 Transaction Recording

The code supporting each entity's method has been modified to include a call to record transactions if the method call is successful. This call generates a transaction entry that is parsed into strings and written to the PTL file. An invariant of transaction entries is that each record must include at the very least TransactType, EntityPIFID, and EntityName.

The RecordTransaction call includes the TransactType, which describes what transaction has been made, along with relevant arguments. Each TransactType transaction is not necessarily identical or similar to those of other TransactTypes.

TransactionProperty function records information from lower levels of the transaction call. If these properties are not available from the entity level at the time the recording method is called. For example when a new specialization is created, the PIFID for this

entity is set by functions not in the scope of entity. TransactionRecorder retrieves this information to write to file as well by using the static TransactionProperty function. TransactionProperty was not implemented as an object because of the simplicity of its function.

The transaction data written to file are comma separated values format (CSV). The PTL file stores all the information necessary to replay all of the transactions made during recording, given they are recordable.

In parsing recorded data, if arguments contain objects, they are parsed to string to be rebuilt on playback. For example, arrays are parsed with the "#" delimiter between each array elements. This poses a requirement to the arrays that can be parsed. For example, for successful parsing and rebuilding, arrays cannot have the "#" character. However, because these arrays are created by the PH_TNG, and only contain integers, longs, or other numbers, the "#" delimiter does not pose any potential bugs. In the current implementation of the Process Handbook, "#" is not allowed in the strings that describe a process. If this limitation should be removed from the Process Handbook in the future, a different delimiter must be used for the transaction recorder. Boolean values and numeric values are written directly to the file. Section 4.6 describes in detail the PTL file.

4.6 Transaction Playback

Playback is possible when transaction recording is turned off. In essence, the playback system tries to emulate the PH_TNG Editor client in executing transactions. As the user clicks on an entity with a mouse and chooses a transaction, the playback procedure first obtains a pointer to an entity and chooses the transaction. With the correct parameters available for this transaction, a successful replay of the transaction is made.

As mentioned in the previous section, all transaction records in the PTL file include the entity's PIFID and entity's name. When the user chooses to playback a PTL file, a

pointer of the opened PH database and the PTL filename is passed to the server-side transaction replay database object.

This object, named TransactionPlayback_DB, takes the pointer of the opened database and begins replay. All transactions recordable translate to function calls made on entities. Since the PIFID of that entity is included in the PTL file, the replay object gets the entity by PIFID, then make the same function call. Information on parameters needed for these function calls are stored in each transaction entry in the PTL file. A detailed spec on playback is described in Chapter 5.

4.7 The Process Handbook Transaction Log (PTL file)

4.6.1 Design Strategy

Several design considerations were made in the implementation of the Process Handbook Transaction Log, or the PTL file. These files have a "`.ptl`" extension, and are recognized by the Process Handbook transaction recorder and playback system. Several identifiers in the PTL file make it easily understandable by human readers and also make it identifiable to the transaction replay system.

Requirements of the PTL file include that it be in ASCII format, so that it would be easy to read in case manual editing was required. This feature is important because from a quick scan a user can see what object will and will not be modified. For that reason, writing to PTL's in binary format were not a possibility, even though that might have made information retrieval simpler. A second concern was portability. For easy manipulation of the data on PTL files, transactions are recorded with parameters in comma separated values format, also known as CSV, usually with a "`*.CSV`" extension. Therefore a PTL file may be considered a subset of CSV files. This permits the data in PTL files to be viewed and manipulated by spreadsheet applications such as Microsoft Excel, and edited if necessary. Figure 7 is a sample of a small PTL file.

4.6.2 Design Overview

There are so far three simple entry types within the PTL file format. These include:

- Version Property
- Database access and closure
- Transaction Record

At top of the PTL file is the title. The version property immediately following the title displays the version of the PTL file, which has progressed in the past several months.

"**Process Handbook Transaction Log**"
"Version 2.7"
"**Open_Transaction_Log** C:\Data\edhsu\Database\addnewattribute.ptl FOR PHdatabse
C:\Data\edhsu\Database\Exception1217v3-edcopy.mdb opened at 5/6/99 7:53:44 PM",#NULL#
"**AddNewAttribute**","MaxVelocity","AddNewSpec","Edward Source",
"990505170702UF13156","testing old function","990506202039UF13186","ed",#NULL#
"**Close_Transaction_Log** C:\Data\edhsu\Database\addnewattribute.ptl FOR PHdatabase
C:\Data\edhsu\Database\Exception1217v3-edcopy.mdb closed at 5/6/99 8:27:32 PM",#NULL#
"**Open_Transaction_Log** C:\Data\edhsu\Database\total-ait-war.ptl FOR PHdatabse
C:\Data\edhsu\Database\Exception1217v3-edcopy.mdb opened at 5/7/99 7:56:29 PM",#NULL#
"**AddNewSpec**","Default Parent Activity","981218102442UF12785","Assemble Stike Team",
"990507195742UF13197","Edward Hsu",#NULL#
"**AddNewDecomp**","Nato AirStrike","990507195724UF13195","Assemble Stike Team",
"990507195742UF13197","Edward Hsu",#NULL#
"**AddNewSpec**","Default Parent Activity","981218102442UF12785","Assess Targets",
"990507195753UF13199","Edward Hsu",#NULL#
"**AddNewDecomp**","Nato AirStrike","990507195724UF13195","Assign Units",
"990507195807UF13203","Edward Hsu",#NULL#
"**AddNewAttribute**","990507160020UF13193","Airstrike","Examiner","#0#","#0#","John Quimby",
"TransactProperty(6) - object not parsed",5,#FALSE#,#NULL#
"**RemoveFromDecomp**","990507195753UF13199","Assess Targets",
"990507195724UF13195","Nato AirStrike",3380,#NULL#
"**AddNewDecomp**","Deploy Strike","990507195814UF13205","Determine Strategy thing a",
"990507135342UF13189","YourName/CCS",#NULL#
"**AddNewDecomp**","Nato AirStrike","990507195724UF13195","Airstrike",
"990507160020UF13193","ed",#NULL#
"**RemoveFromSpec**","990505202603UF13175","An Attack to be moved to How",
"950717143947AB705","Destroy",#NULL#
"**AdoptSpec**","950609162618AB521","Consume Product","950613181812AB532",
"Beer consumption chain",#NULL#
"**RemoveFromSpec**","950613181812AB532","Beer consumption chain",
"950609162657AB522","Consumption Chain",#NULL#
"**Close_Transaction_Log** C:\Data\edhsu\Database\total-ait-war.ptl FOR PHdatabase
C:\Data\edhsu\Database\Exception1217v3-edcopy.mdb closed at 5/7/99 8:08:19 PM",#NULL#

Figure 7: Sample Process Handbook Transaction Log (PTL file)

The transaction playback system checks for these two properties as well as the value of the version counter in the version property to ensure that no problems would be encountered during playback due to PTL versioning problems. Checks are currently done in a manner that does not maintain backward compatibility of PTL files since the PTL has evolved tremendously during its development. No other checks are done to ensure the integrity of the PTL file. As a result, if changes are ever made manually, they will have to be done carefully to maintain the PTL's validity.

4.6.3 Database access and closure

The database access and closure properties record the time and date the PTL file was opened for a particular database. The full path of the database is included to ensure that no conflicts occur with different PTL files in other directories. Detailed information on methods called during this procedure is included in Chapter 5.

4.6.4 Transaction Records

The payloads of the PTL file are the transaction records. Each store the data for a particular transaction applied to an entity in the Process Handbook database. Of the 23 methods for each entity, 16 of these transactions modify the PH database. Of these 16 transactions, 14 are recorded and replay-able by the Process Handbook transaction recorder and playback system in the current implementation. These limitations are due to difficulties in parsing an object. This problem is still being addressed. There are therefore 14 different variations of transaction records. Each variation carries the information necessary for replaying its particular transaction.

Each transaction record includes a header, which stores the transaction type of this transaction. The rest of the record includes information that is used in this transaction. Each of these individual pieces of information is called a *transaction property*. The first two transaction properties are always *Entity PIFID* and *Entity Name*. The entity referred to here is the entity on which methods were called to generate the transaction.

All data used in the transaction are parsed to transaction properties in text in the PTL file. This includes strings, all types of numeric data, booleans, objects and arrays.

4.6.5 Expandability

To allow for easy reading and future expansion, additional information has also been stored in each transaction record. PIFID's are used in identifying entities in playback. Retaining the entity name, as well as the contact name, allows for human eyes to easily identify what transactions were made to the Process Handbook database.

Future expansion of the use of PTL files could include allowing the user to selectively choose not to playback transactions made by a particular person (in the contact field), for example. Since the contact is always a transaction property, implementation of this functionality would be very simple.

4.6.6 Independence from Rest of Recording System

Care has been taken to ensure that the PTL file format is not dependent on the method of transaction recording or playback and vice versa.

For example, if objects exist in a transaction property, these objects are passed to the functions saving the objects to file. The functions responsible for writing to the PTL file take responsibility in parsing objects to the ASCII string necessary for writing to the PTL. The same situation is true for playback.

As a result, the PTL file format is not dependent on recording or playback methods and vice versa. This is particularly beneficial if either of the two properties need to be changed or re-implemented for whatever reason.

4.8 Charts of Forms

Very few forms have been added to the Process Handbook in creation of the PH transaction recorder and playback system. The addition of this system remains quite transparent to the user, and the interface is very simple. This may change, however, when further functionality is added to the system. These additional functions are described in Chapter 7.

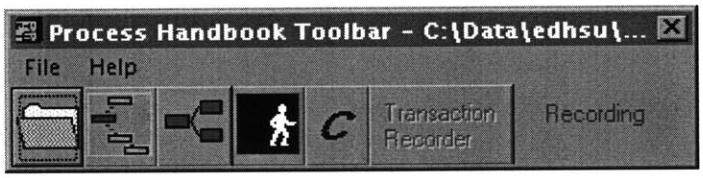

Figure 8: Process Handbook Main Menu

The main menu of the Process Handbook has been expanded to include access to the transaction recorder's functions. In addition, a display indicates to the user whether transaction recording is currently turned off or not. The Process Handbook Main Menu (Figure 8) shows the window when transactions are being recorded.

When the user selects to open a Process Handbook database, the database name is checked against the Process Handbook record of PTL files associated with previously

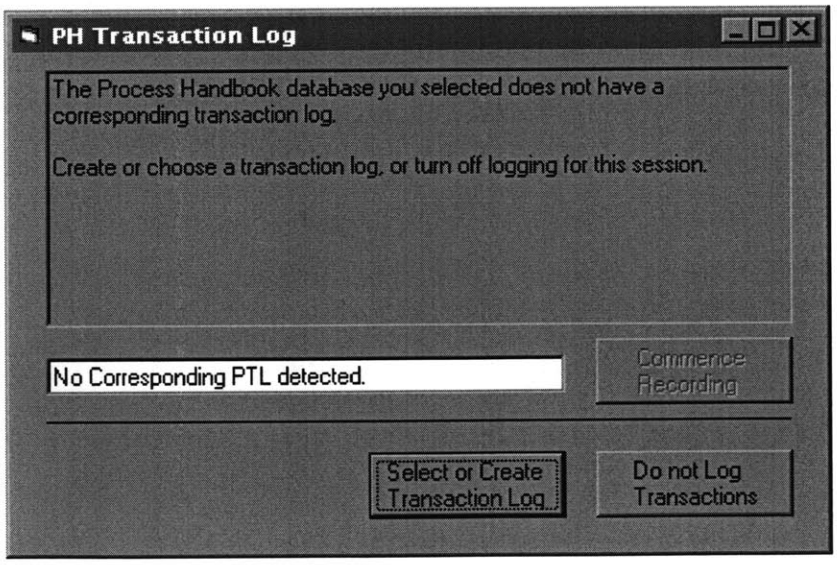

Figure 9: Process Handbook Transaction Log Prompt

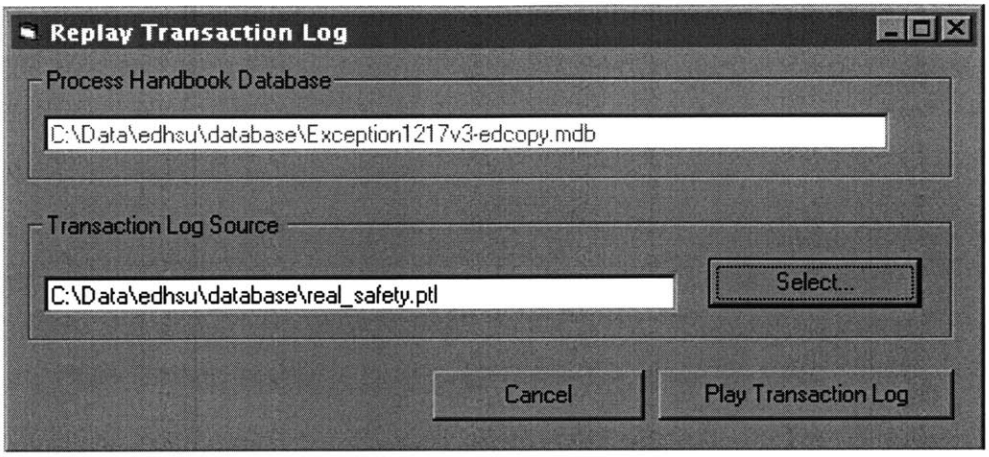

Figure 10: Choosing to replay a PTL file

opened databases. The user then can decide if transaction recording should begin or to turn off recording. In Figure 9, no PTL file was detected.

If transaction recording is turned off, the "Transaction Recorder" button on the Process Handbook main menu is enabled. This allows the user to choose a PTL file to replay. The Process Handbook database currently opened is displayed next to the selected PTL source to allow the user to confirm that the correct PTL file has been selected for the opened database (Figure 10). The PTL file is filtered to allow easy locating of a particular PTL file to be replayed (Figure 11).

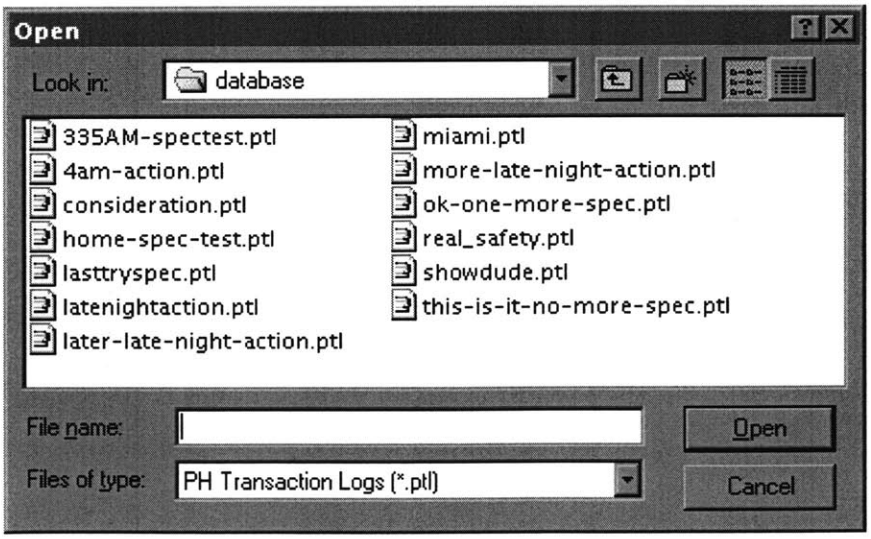

Figure 11: Choosing a PTL file

Chapter 5:
Implementation of the Recording and Playback System

5.1 Transaction Recorder Architecture

The current implementation of the Process Handbook is designed as a three-tier client/server application. Figure 12 shows the three-tier architecture for the Process Handbook. The architecture consists of three separate layers: the database layer, the logic layer, and the client layer.

The viewers/editors for the Process Handbook form tier 1. These can either be GUI clients or other applications that interact with the Process Handbook. These clients can only interact with the Process Handbook by using the object API. These clients deal with graphic user interface (GUI) and representational issues so do not need to contain algorithms dealing with the implementation of the underlying semantics.

This holds true for the transaction recorder as well. Modules relating to user interface and functions calling initialization methods of transaction recorder objects are made from this layer in the PH_TNG client editor. Additions made in this tier are mostly cosmetic and relatively small compared to that in the PH Server.

The object server forms the middle layer. It contains the code for managing the database and for supporting the Process Handbook functionality. The object server exposes the object API to the client applications. The clients in tier 1 can only interact with the Process Handbook by using this server. Thus, the clients are prevented from performing illegal functions. The API abstracts the implementation of the algorithms and the storage device. Most importantly, the logic layer provides automatic support for inheritance. Whenever a client modifies the description of a process, this layer automatically creates/updates all the records in the database that are effected by these changes. (Farooq, 1997)

Since the Client Layer (TIER 1) may be changed or expanded to include other clients in the future, the PH Object Server (TIER 2) is the most logical location to effect recording of transactions in the PH transaction recorder and playback system. Minimal changes are made in tier 1, since any additional functionality added directly to the PH_TNG editor client would also have to be added to other tier 1 clients in the future.

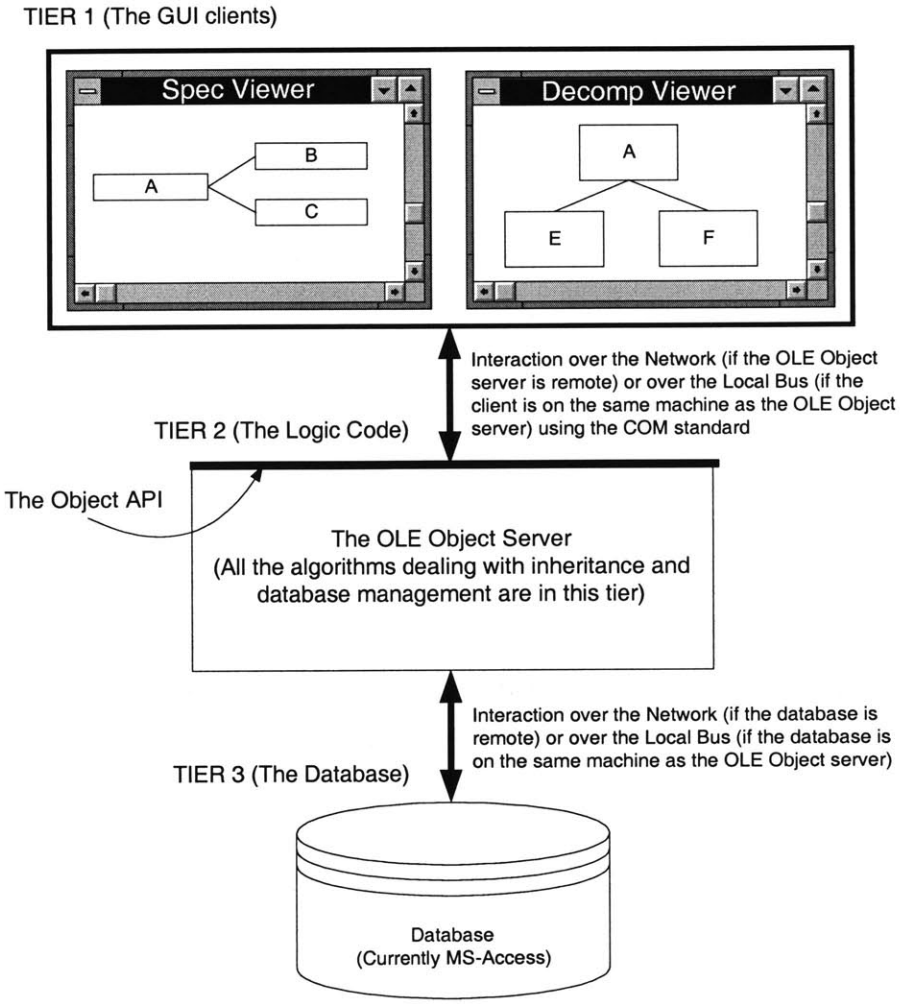

Figure 12: The Three-Tier Client/Server Design (adapted from Farooq, 1997)

5.2 First-Class Objects

The object server exposes a total of fourteen first-class objects as listed in Table 1. The usage of some of these objects in performing the basic Process Handbook functions are

discussed in Farooq's 1997 MEng thesis entitled "An Object Server for the Process Handbook." However, many changes have been made to the process handbook since the object server's initial development. The newest objects are denoted in the table with "*."

PH_Server

db	ObjRelation	ObjAttribute
PH_DB	ObjRelations	ObjAttributes
Entity	*TransactionAgent**	ObjRelAttribute
Entities	*TransactionReader**	ObjRelAttributes
Generic	*TransactionReplayDB**	

Table 1: PH Server First-Class Objects
***denotes New Objects**

5.2.1 Transaction Recording and Database Interaction

The PH_DB object is used to interact with the database directly. The server opens the database for exclusive use because if multiple servers were to interact with the same database, the changes made by one server could effect the validity of objects in another server. For example, a PH_TNG editor and PH_Webserver can both open the database simultaneously. However, no notifications of changes are made to these clients if either of the clients modify the database. This is a problem due to caching of the information in the database.

After the database is opened using the OpenDb method, the transaction recorder is initialized with the TransactionRecorder("Initialize"), which is a method of the TransactionHandler module in the PH_Server. This call sets parameters necessary for recording including the PTL target filename, as well as global variables such as "Recording = True." This global variable is verified to be true before any transaction is recorded. This allows for the expanded functionality of turning on and off recording during sessions.

Recording calls are made from the methods of entity. Each recordable transaction's corresponding entity method has been modified to include a call to RecordTransaction function in the Transaction_Handler module if the transaction is successful and no errors are generated. In the current implementation, 14 of the 16 entity methods that modify entities can be recorded and played back.

After the client finishes the interaction with the database, the client should execute the CloseDB method as well as the TransactionRecorder("Save") method. CloseDB executes database maintenance routines and closes the database, while TransactionRecorder("Save") does the same for the PTL file.

5.2.2 Transaction Replay and Database Interaction

After the database is opened using the OpenDb method, the transaction recorder is initialized with TransactionRecorder("Initialize"). But in this case, when the user is prompted, the user must turn off recording. This causes transaction recorder initialization to set global variable Recording = False. Current implementation of the transaction replayer requires that transaction recording be turned off for playback.

When the user chooses to replay a PTL file from the main Process Handbook window, the Replay method of TransactionAgent object is called from the PH_TNG editor client in tier 1. The TransactionAgent then calls the ReplayPTLfile in the Transaction_Handler module. This function initializes the TransactionReplayDB, which plays back each transaction by retriving each entity via its GetEntityByPIFID method and then executing the transaction.

If replay of the PTL file is successful without errors, the user is notified and calls are then made to refresh some of the windows in the Process Handbook Application at the client side.

5.2.3 Process Handbook Entities

The Entity object is the heart of the object server. There are eight sub-types within the Entity object: thing, activity, bundle, port, navigational node, dependency, resource, and attribute-type. All of these can be specialized and decomposed. This sets them apart from the relations and attributes, both of which can neither be specialized or decomposed. Moreover, the decomposition relations, connectors, and attributes only have any meaning in the context of an entity. The navigational relations (discussed later in the thesis) also have no meaning without the entities that form their end points.

Methods of the entity object have evolved in the past few years. The current implementation has the methods listed in Figure 13. Methods that modify the entity object or its attributes are labeled with "*". Those that can be recorded and played back are labeled with "+" as well. The user is notified if the transaction being executed cannot be recorded.

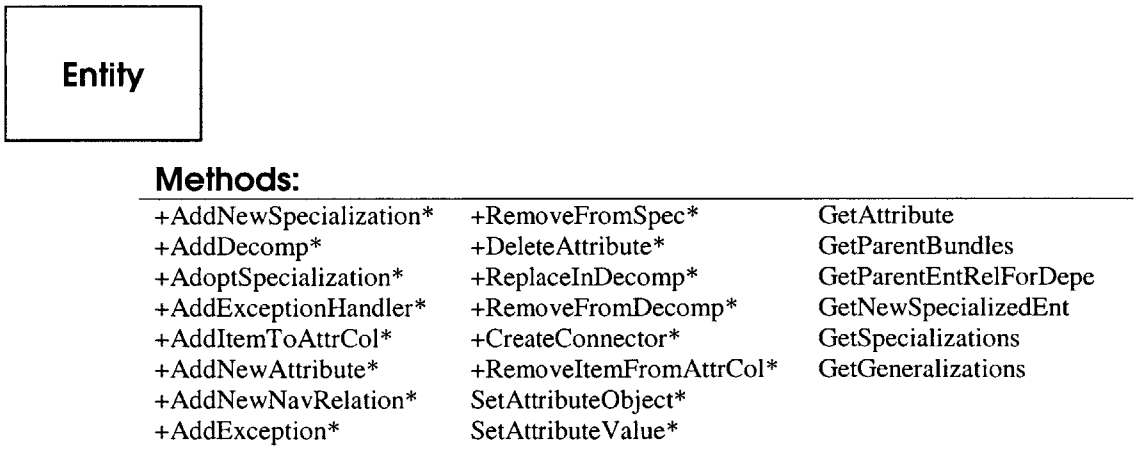

Figure 13: Methods of the Entity Object

5.3 Detailed Description of Classes and modules.

This section describes the modules of the PH_TNG editor client portion of the Transaction Recorder and the PH_server portion. Little changes have been made to the existing modules Process Handbook as possible. As a result new modules were created in both the server and the editor. The relevant modules and their functions are described below:

5.3.1 Relevant modules in PH_TNG editor client:

Modules in the PH_TNG editor client have been minimized since most of the code for our system lies mainly in the PH_server portion of the Process Handbook.

- Form: frmTransactionLog

  This form indicates to the user if a Process Handbook Transaction Log (PTL file) has been detected for the database being opened. If detected the user to chooses if he wants to record transactions or not. If not detected, the user can choose to record with a new PTL file or not.

- Form: frmReplayPTL

  This form is displayed when a user chooses to play back a transaction log. It displays the current database opened, and the PTL file to be played back once the user chose it.

- Module: Transaction_Handler

  The Transaction_Handler module contains functions that set parameters to initialize the recording process and functions that complete the recording process.

Other functions in this module take calls from the PH_TNG editor client modules and communicates with the PH_Server.

5.3.2 Relevant modules in PH_Server:

Most of the code in the transaction recorder exist in the PH_Server since that is where all the activity directly relating to recording are.

- Module: Transaction_Handler

  Module contains functions for general transaction recording, TransactionHandler initialization and save, file parsing, and transaction log file manipulation. It also contains a static function for setting and retrieving values that are generated prior to record-time in lower levels of object entity's function calls. See appendix A for more information

- Class Module: TransactionPlayback_DB

  This is an object that controls an opened database, initialized by a pointer to the opened PH database. It retrieves the entities on which the recall of methods for transaction is made

- Class Module: TransactionReader

  This small object is used to simplify transferring information from PTL files or to PTL files.

- Class Module: TransactionAgent

  This object serves to communicate between PH_TNG editor client and PH_Server.

5.3.3 Transaction Recording Command Hierarchy

To give future developers of the Process Handbook a clearer description of the relations among the modules and their functions, the command flow of the Recording is illustrated in the following figures. These figures do not contain any detail of what how each function works or their requirements. However, they do provide a very clear presentation of interactions within the Process Handbook Transaction Recording and Playback System's architecture.

Figure 14 illustrates the command path for transaction recording, starting from the calls made while a database is being opened. From frmSpecHier, the user opens a PH database by triggering the mnuFileOpenDB_Click() event. In this procedure, Transaction

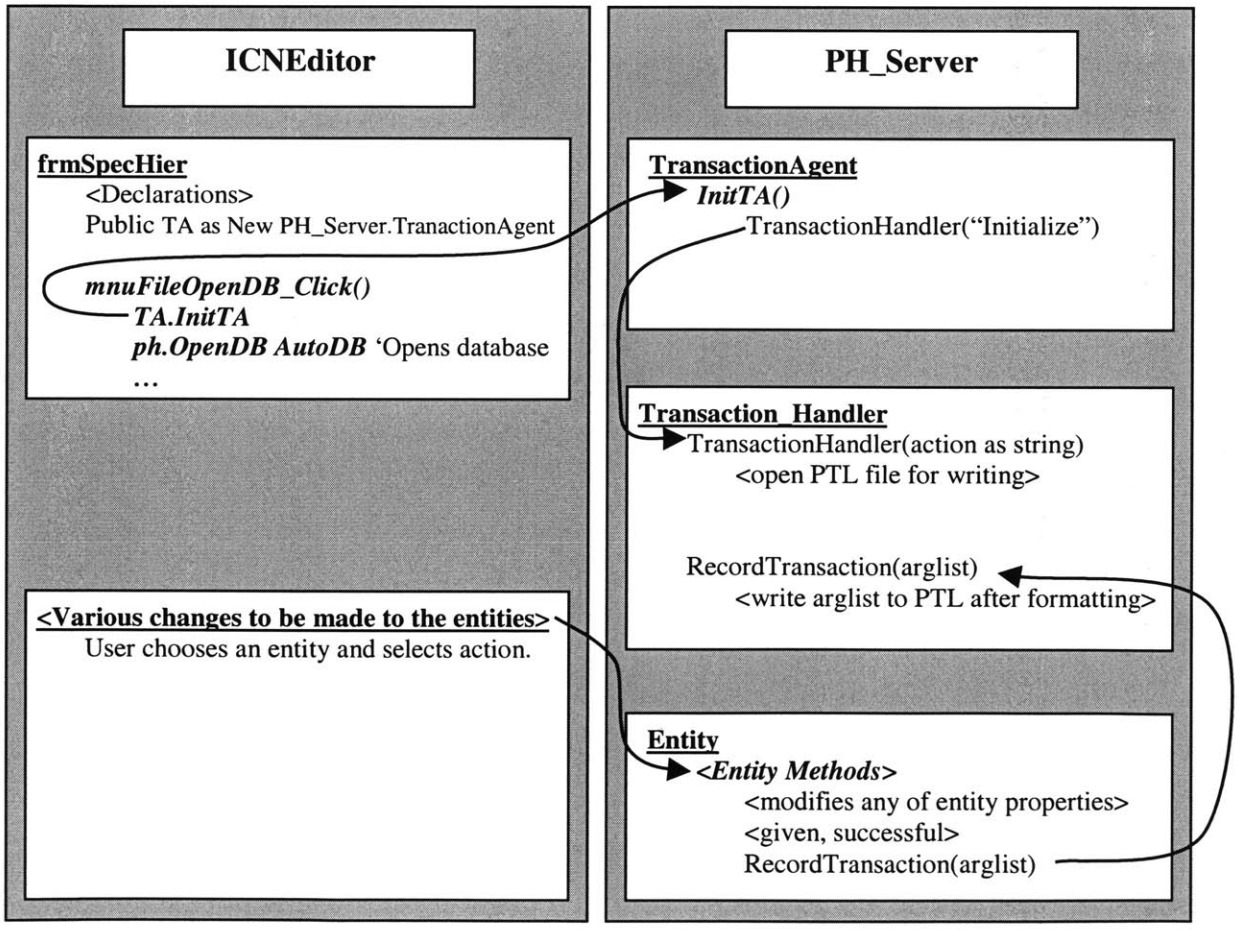

Figure 14: Relevant Command Hierarchy for Recording a PTL file

Agent TA is initialized by calling TA.InitTA, and prompting the user to choose a PTL file for recording. This TransactionAgent method calls TransactionHandler("Initialize"), which sets up filehandles and global variables including boolean `Recording`.

With initialization complete, the user proceeds to make changes using the PH_TNG client. When methods in an entity selected by the user are used, additions in the entity's methods code leads to calls to RecordTransaction(arglist), where arglist would be the relevant information pertaining to the entity method being recorded.

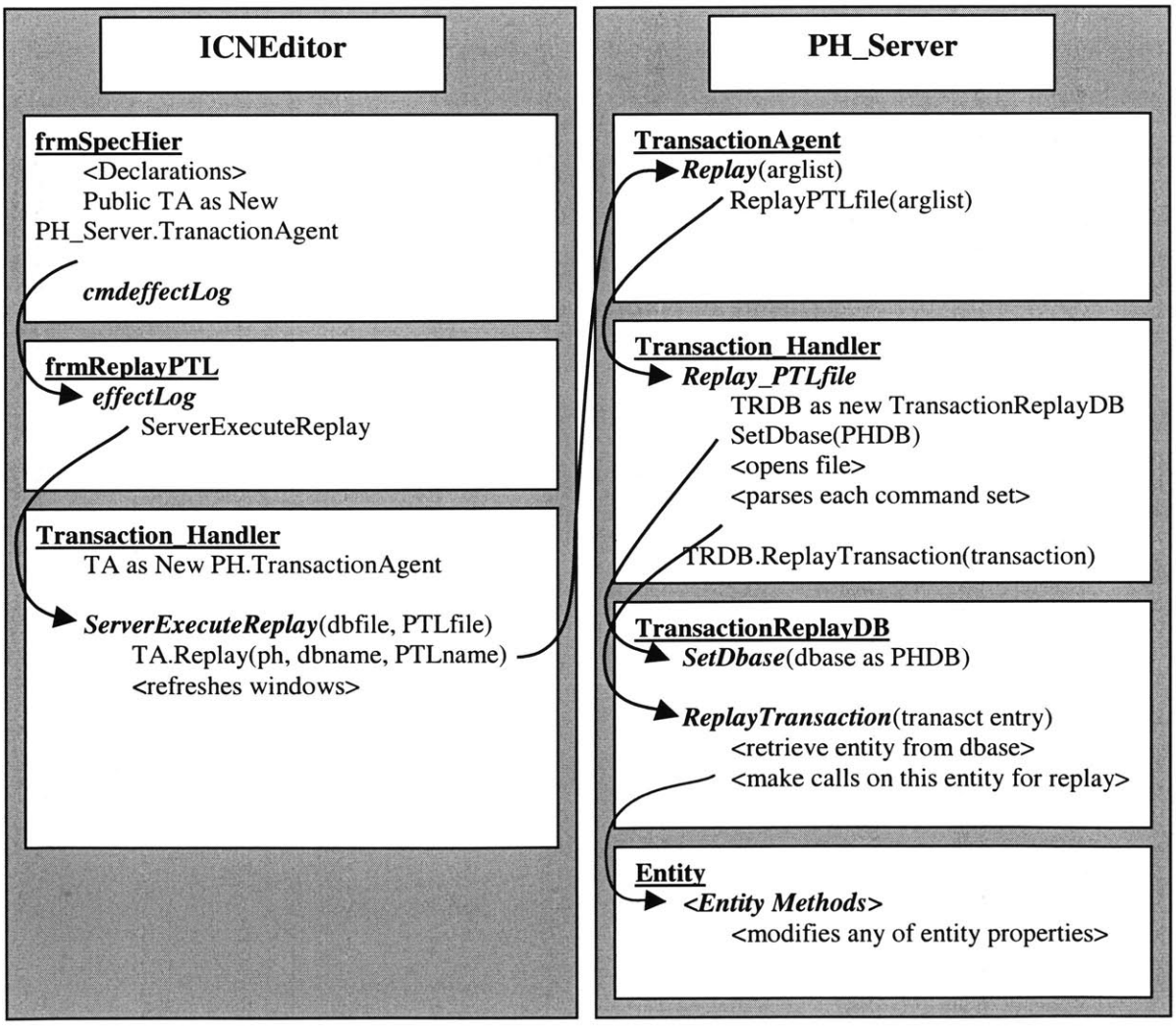

Figure 15: Relevant Command Hierarchy for Replaying a PTL file

5.3.4 Transaction Playback Command Hierarchy

Figure 15 illustrates the command path for transaction playback from a PTL file, assuming that a PH database has already been opened and transaction recording has been turned off. When the user triggers the cmdeffectLog event, the Transaction Recorder Replay menu (frmReplayPTL) is displayed. Once the user selects a source PTL file, the form calls ServerExecuteReplay(database filename, PTL filename) in the Transaction_Handler Module. This function in turn makes a call to the server-side object TA, a server-side TransactionAgent. The agent in the server passes this information to the ReplayPTLfile function in the server-side Transaction_Handler module.

ReplayPTLfile first checks the validity of the file to confirm it is a Process Handbook transaction log. Then it initializes the TransactionReplayDB object with its SetDbase method, passing in a pointer of the currently opened database. Replay_PTLfile then opens the PTL file and begins parsing each transaction entry. The entries are passed to repeated calls to ReplayTransaction(transaction entry). This is done until all the transaction entries have been parsed and executed by TransactionReplayDB.

Chapter 6:
Evaluation of Achievement of Goals

Recording and playback system successful for large part of PH functions

The newly implemented Process Handbook Transaction Recorder and Playback System integrated with the existing Process Handbook, has demonstrated the feasibility our approach as defined in Chapter 3. This system allows user groups and sponsors of the Center for Coordination Science to routinely gain updates of process and organization models created by CCS scientists, without disrupting their own current systems Process Handbook databases.

As of this writing, of the sixteen transactions based on entities that modify the Process Handbook database, fourteen have been successfully recorded and played back. Limitations of the existing Process Handbook application on which this system was implemented, has resulted in this temporarily limited success. However, work continues as this researcher and CCS scientists are addressing this unresolved issue.

The following sections describe the success in achieving our goals, including each of the goals listed in chapter 3.

6.1 Updating Client Process Handbook Databases

Over half of the changes that can be made by the PH_Client can now be recorded and played back to other databases with this Transaction Recorder and Playback System. The implementation of this transaction recorder over a "diff" utility was decided because of its non-intrusiveness to the existing Process Handbook application. Other reasons for this design were addressed in Chapter 4. In addition, the implementation of this transaction recorder makes it easily possible to implement a "diff" utility synchronization system.

With the transaction re-player in this system, Process Handbook users can update these changes to their database and see from the PTL file what these changes will be before they make the modification. The CSV format of the PTL file allows users to use spreadsheet applications to study these transactions in the PTL files. PTL files can also be copies and spread to many users at once, which makes multiple-point updating possible despite differences among the substrate databases.

6.2 Database Integrity

The design of the Transaction Recorder and Playback System has made it possible for maintaining database integrity of substrate databases after updating by not permitting the transaction recorder to record transactions that are not successful. In addition, the system notifies the user if a transaction was not successfully recorded for any reason. This allows the user to undo or cancel the operation.

The database integrity issues involved need only be related to playback and recording because the implementation of the transaction recorder was made at the highest level possible in the PH server. As a result we only need to ensure that entity methods are called correctly with the proper parameters. Lower-level database integrity issues need not be addressed since that has been done so during the implementation of the PH server. Thus, to maintain database integrity and synchronicity, we need only assure that each transactions is recorded successfully if and only if the transaction is executed successfully.

6.3 Maintainability and Maintaining Abstraction

Abstraction was maintained in the design and implementation of the transaction recorder in relation to the existing Process Handbook application as well as the internally. No changes were made to existing modules in terms of function and architecture. Each new function of the transaction recorder and player were defined in modules or class modules, which in turn used existing PH_server functions or object methods.

If any changes are ever made in the future to the PH_server, as long as the input requirements of the new functions do not change, our abstraction ensures that no changes will need to be made in the transaction recorder.

The non-intrusive integration of the transaction recorder and the existing Process Handbook application was also possible with our maintenance of abstraction. There is no possibility that the transaction recorder and playback system might corrupt the functions of the existing Process Handbook.

Within the transaction recorder portion of the application, maintainability is easy, as abstraction has been maintained among the modules doing playback, recording, and file access. This allows for the possibility to change recording method, PTL file format, or any other function without modifying other modules.

6.4 Forward Compatibility and Easy Expansion of Transaction Recorder

The maintenance of abstraction in the design of the Transaction Recorder ensures forward compatibility of the system. Each of the transactions recorded and also made by the playback use only function calls that are already exist and used by the Process Handbook Client. The PH server can thus be modified extensively, and as long as the PH client does not to be re-implemented, neither will the Transaction Recorder.

Possible useful expansions of the Transaction Recorder's functionality are proposed in the following chapter. These proposed changes are easily possible since these expansions were in mind during the transaction recorder's design. The information saved in PTL files are more extensive than necessary to ensure proper playback. The additional information saved allows future versions of the application to check these bits of information, for example.

Another example is the `Recording` global Boolean in the transaction recorder. As discussed in chapter 4, the current version of the system does not allow the playback of

PTL files to be recorded in another transaction log to minimize complexity during the transaction recorder's design. To enable this feature, the simple changes that need be made are an additional file handle for recording, as well as flipping a Boolean.

6.5 Limitations

Although a large majority part of the goals set for success were reached, a comprehensive and fully tested transaction recorder and transaction player was not possible due to several limitations.

One of these limitations was the existence of unresolved issues in the current version of the Process Handbook application. For example, some of the transactions targeted for recording could not be accessed through the user interface.

Modifications to the Process Handbook code has only recently been made in attempt to enable testing . Few of the transactions that can be recorded, as a result, could not be tested. Currently, the transactions not recorded are namely SetAttributeValue and SetAttributeObject. Work continues in addressing this issue.

An additional limitation included lack of detailed technical documentation of the Process Handbook application. For example, a previous thesis that described the architecture and specs of the current PH object server was indeed outdated. This problem was remedied by the expertise of CCS research scientist John Quimby, that was often available for issues relating to the object server.

In addition, this implementation of the Process Handbook Transaction Recorder is not comprehensive, as it was based on recording and replaying database-modifying methods of the object Entity. However, changes can also be made to the database through methods of the object ObjAttribute. Work needs to be continued in this section for a comprehensive database synchronization system. Since the transaction recording framework is already in place, the modifications necessary will be minor.

Summary

The Process Handbook Transaction Recorder and Playback System has now demonstrated the feasibility for user groups and sponsors of CCS to routinely gain database updates through Process Handbook Transaction Logs. These logs can contain information of new process and organization models created by CCS scientists, and may update client databases without disrupting their Process Handbook databases. Limitations and issues still exist, however, although testing and implementation of the system has been taken to the farthest extent possible with the current Process Handbook object API.

Chapter 7:
Future Work

7.1 Ease of Expansion

The Process Handbook Transaction Recorder and Playback System has been designed and implemented with forward compatibility and expandability in mind. The modularity and maintainability of the system has been discussed in chapter 4. Other useful additional functionality of the system is described in this chapter.

7.2 "Diff Utility"

The design of a Transaction Recorder and Playback system was preferred over the "Take two databases, and update both"-type "Diff Utility" because the Transaction Recorder allows multiple databases to replay the transactions from a single database. They would also have a step-by-step view of the new database changes. In addition, a Transaction Recorder implementation ensured that there are no problems relating to database update conflicts.

A "Diff" utility however can be implemented to make use of the transaction log player's capabilities. For example, to use the Transaction Recorder and Playback system an application (Diff-Utility-PTL generator) can be designed to generate a PTL file to be played back in either of the PH databases being compared and synced in that manner. This implementation can be made relatively simpler since the PTL file could simply record all the steps made while the "Diff-PTL" generator edited the target database to become identical to the source database.

7.3 Transaction Recorder and Replayer Function Expansion

As suggested in chapters 4 and 5, additional information has been stored in PTL files

relating to the transaction than those that are required to replay the transactions. Entity and contact names are an example of this.

Future versions of the transaction re-player can use this additional information to allow the user to select the changes that should be replayed based on contacts. For example, a user can choose to replay transactions made only by John, but not Mike.

Another functionality is selective recording and playback, based on the transaction type, or time frame. This implementation would allow a user to only record transactions that add entities to the database, or transactions that are not "Delete Decomp." The user could also dictate that only transactions made between "June 4th" and "July 7" be played back. This latter functionality would require that the PTL record transaction time, which is not in the implementation described by this thesis. Other possibilities include user comments, that allow a user to mention why a change has been made. With this functionality, other users can see why a specific relation has been replaced by a specialization.

There are many more additional functionality that may be added to the Process Handbook Transaction Recorder and Re-player. Many of these additions can be made relatively easily due to the modular and simple architecture of the newly integrated system.

Chapter 8:
Conclusions

The implementation of a transaction recorder on the Process Handbook has demonstrated the feasibility of allowing the Center for Coordination Science to update user Process Handbook databases through PTL files. Recording and playback testing indicates that the transaction recorder is largely successful in updating changes of a Process Handbook database to another database.

The architecture of the transaction recorder is non-intrusive to the existing Process Handbook, both in the user interface as well as architecture. The system can be easily expanded to provide additional functionality in transaction recording and playback options as well.

The new system provides a more comprehensive way of transferring PH database processes than the existing PIF application, as it allows the user to see the changes that will be made through the PTL file. With easy expansion, the user will also be allowed selectively replay transactions via choosing properties to include or exclude; contact, for example.

The Transaction Recorder allows for easier communication of ideas among users of the Process Handbook by through transaction playback. In addition, the system provides snapshots of PH databases in time with the PTL file. The new functions made possible with the Process Handbook Transaction Recorder and Playback System will make the Process Handbook an even more robust tool for analyzing processes by facilitating communication of ideas among its users.

References

Ahmed, Zia (1998) An Integrated Dependency Editor for the Process Handbook. Unpublished M.Eng. Thesis, Department of Electrical Engineering and Computer Science, MIT

Ahmed, Erfanuddin. (1995). *A Data Abstraction with Inheritance in the Process Handbook.* Unpublished M.S. thesis, Department of Electrical Engineering and Computer Science, MIT.

Dellarocas, C. (1996). *A Coordination Perspective on Software Architecture: Towards a Design Handbook for Integrating Software Components.* Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, MIT.

Elley, Yassir (1996). A Flexible Process Editor for the Process Handbook. Unpublished M.S. thesis, Department of Electrical Engineering and Computer Science, MIT.

Farooq, Umar. (1997) An Object Server for the Process Handbook. Unpublished M.S. thesis, Department of Electrical Engineering and Computer Science, MIT.

Lee, Jintae. (1994) The PIF Process Interchange Format and Framework. MIT Center for Coordination Science Working Paper 180.

Malone, T.W., Crowston, K., Lee, J., Pentland, B., Dellarocas, C., Wyner, G., Quimby, J., Osborne, C. and Bernstein, A. (1999). *Tools for inventing organizations: Towards a handbook of organizational processes* (Revised 10/98 Working Paper 198), Center for Coordination Science, MIT.

Spencer, K.L. and Miller, K. (1996). *Client/Server Programming with Microsoft Visual Basic.* Microsoft Press, Redmond, Washington.

Tanenbaum, A. (1992). *Modern Operating Systems.* Prentice Hall.

Yuen, Calvin M. (1997). *A Discretionary Access Control Policy for the Process Handbook.* Unpublished M.S. thesis, Department of Electrical Engineering and Computer Science, MIT.

(1995). *Microsoft Visual Basic: Building Client/Server Applications with Visual Basic.* Microsoft Corporation.

(1995). *Micorsoft Visual Basic: Professional Features.* Microsoft Corporation.

Appendix A: Object API Documentation

New Modules in Process Handbook the Next Generation Editor

- Form: frmTransactionLog

  This form indicates to the user if a Process Handbook Transaction Log (PTL file) has been detected for the database being opened. If detected the user to chooses if he wants to record transactions or not. If not detected, the user can choose to record with a new PTL file or not.

- Form: frmReplayPTL

  This form is displayed when a user chooses to play back a transaction log. It displays the current database opened, and the PTL file to be played back once the user chose it.

- Module: Transaction_Handler

  The Transaction_Handler module contains functions that set parameters to initialize the recording process and functions that complete the recording process.

  Function ReplayTransactionLog(DBfilename, PTLfilename As String)

  - Highest level function for replaying transaction. Function called by frmReplayPTL once user sets preferences.

  Function ServerExecuteReplay(dbfile, PTLfile As String)

  - This function serves to form an abstraction for calling the Transaction Agent object's function for replaying a PTL file
  - Calls functions for refreshing opened windows after replay is successful

  Function SetTransactionLog(PTLfilename As String)

  - This function serves to form an abstraction for calling the Transaction Agent object's function for setting the current transaction log filename.

  Function TransactLogExists(DBfilename As String)

  - Checks to see if a transaction log filename is on record for the database name being checked. Returns Null if no PTL on

record for database in argument

Most of the code in the transaction recorder exist in the PH_Server since that is where all the activity directly relating to recording are. New Modules in the Process Handbook Object Server.

- Module: Transaction_Handler

  The server-side Transaction_Handler module contains lower-level functions that initialize the recording process and functions that concludes the recording process for a session. It also contains functions to parse PTL files to be replayed. Functions for recording transactions are also included.

  Function RecordTransaction
  (TransactType As String, ParamArray TransactProperty())

  - Requires: TransactType to be one that can be handled by this function. In the arguments for the TransactType must also satisfy requirements.

  Function ReplayPTLfile(dbase As PHDB, PTLfilename As String)

  - Opens file <PTLfilename> and repeatedly calls TransactionPlayback_DB.ReplayTransaction() as it iterates through each transaction entry.

  Public Function SetTransactionLog(PTLfilename As String)

  - Abstraction for setting PTL filename.

  Function TransactionHandler
  (Action As String, Optional DBfilename As String, Optional PTLfilename As String)

  - Function initializes system for transaction recording if Action = "initialize" and completes recording cycle if Action = "save".

  Static Function TransactionProperty
  (Action As String, Optional Key As String, Optional Value As String) As String

  - This function records and retrieves information that is created at lower levels, which are needed for recording transactions.

- Class Module: TransactionPlayback_DB

  class that controls an opened database
  Initialized by a pointer to the opened PH database
  Retrieves the entities on which the recall of methods for transaction is made

Function ReplayTransaction
(TransactType As String, ParamArray TransactProperty())

- Retrieves entities from initialized PH database and calls entity method as described in transaction entry(TransactType + TransactProperty()). Returns true if replay successful, and false otherwise.

Public Function SetDbase(dbase As PHDB)

- Initializes pointer for TransactionReplayDB.

- Class Module: TransactionReader

Object created to simplify transferring information from PTL files or to PTL files. This object intakes each PTL transaction entry's parameters for access for transaction playback.

Public Function SetFoxVal(a As Integer, Value)

Public Function ResetVals()

- Class Module: TransactionAgent

Serves to form an abstraction and communicate between PH_TNG editor client and PH_Server.

Public Sub AssocPTL(TA_BDfile)

- Abstraction layer call.

Public Sub InitTA()

- Abstraction layer call.

Function Replay(dbase As PHDB, DBfilename, PTLfilename As String) As Boolean

- If replay successful, true is returned. False otherwise.

Public Function TransactLogExists(filename as string)

- Checks if a transaction log exists for filename. Returns transaction log filename if found, empty string otherwise.

Appendix B:

Transaction Entry Key Summary

This appendix aids the user in reading Process Handbook Transaction Logs (PTL files). Each transaction log entry relating to database modifications contains a *TransactType* property followed by an array of *TransactProperty* variables. The functions of each of these variables are summarized in the function below. Since development in the transaction recorder continues, this list is not comprehensive in describing all transaction entry types.

TransactType	***AddNewSpec***	***RemoveFromSpec***
TransactProperty 0	Name of CreatingSpec	TR_KidPIDIF
TransactProperty 1	CreatignSpec PIFID	TR_KidName
TransactProperty 2	NewSpecName	TR_ParentPIFID
TransactProperty 3	TP_PIFID	TR_ParentName
TransactProperty 4	Contact	Contact
TransactProperty 5		

TransactType	***AddNewDecomp***	***RemoveFromDecomp***
TransactProperty 0	Parent_Name	Self_EntPIFID
TransactProperty 1	Parent_PIFID	Self_EntName
TransactProperty 2	NewDecompName	Parent_EntPIFID
TransactProperty 3	NewDecompPIFID	Parent_EntName
TransactProperty 4	Contact	TR_MySlot
TransactProperty 5		

TransactType	***DeleteAttribute***	***ReplaceInDecomp***
TransactProperty 0	TR_entPIFID	TR_selfPIFID
TransactProperty 1	TR_entName	TR_selfName
TransactProperty 2	TR_attrName	TR_ChildEntPIFID
TransactProperty 3	TR_ContextArr	TR_ChildEntName
TransactProperty 4	TR_ChangedArr	TR_NewChildEntPIFID
TransactProperty 5		TR_NewChildEntName
TransactProperty 6		TR_ContRels'
TransactProperty 7		TRV_ChanArr
TransactProperty 8		TR_Changed

TransactType	*AddNewAttribute*	*AddNewNavRelation*
TransactProperty 0	TR_entPIFID	TR_entPIFID
TransactProperty 1	TR_entName	TR_entName
TransactProperty 2	TR_attrName	TR_NewEndEntPIFID
TransactProperty 3	TR_ContextArr	TR_NewEndEntName
TransactProperty 4	TR_ChangedArr	
TransactProperty 5	TR_attrValue	
TransactProperty 6	TR_attrObject	
TransactProperty 7	TR_Inherit_type	
TransactProperty 8	TR_system_attribute	

TransactType	*AddExceptionHandler*	*CreateConnector*
TransactProperty 0	TR_entPIFID	TR_selfPIFID
TransactProperty 1	TR_entName	TR_selfName
TransactProperty 2	TR_Relation_type	TR_EntStartPortPIFID
TransactProperty 3	TR_NewEndEntPIFID	TR_EntStartPortName
TransactProperty 4	TR_NewEndEntName	TR_EntEndPortPIFID
TransactProperty 5		TR_EntEndPortName
TransactProperty 6		TR_Changed
TransactProperty 7		TR_ContRels
TransactProperty 8		TR_ChanArr
TransactProperty 9		TR_StartPortSlots
TransactProperty 10		TR_EndPortSlots

TransactType	*AdoptSpec*	*AddException*
TransactProperty 0		TR_entPIFID
TransactProperty 1		TR_entName
TransactProperty 2		TR_NewEntPIFID
TransactProperty 3		TR_NewEntName
TransactProperty 4		
TransactProperty 5		
TransactProperty 6		