# Collaborative Design using Long Transactions and 'Change Merge'

*Robert Aish*

## Introduction

If our goal is implement collaborative engineering across temporal, spatial and discipline dimensions, then it is suggested that we first have to address the necessary pre-requisites, which include both the deployment of "enterprise computing" and an understanding of the computing concepts on which such enterprise systems are based. This paper will consider the following computing concepts and the related concepts in the world of design computing, and discuss how these concepts have been realised in Bentley Systems' ProjectBank collaborative engineering data repository:

| **Computing Concept** | **Related Design Concept** |
|---|---|
| Normalisation | Model v. Report (or Drawing) |
| Transaction | Consistency of Design |
| Long Transaction | Parallelisation of Design |
| Change Merge | Coordination (synchronisation) |
| Revisions | Coordination (synchronisation) |

While we are most probably familiar with the applications of existing datadase concepts (such as Normalisation and Transaction Management) to the design process, the intent of this paper to focus the contribution which concepts such as Long Transactions and a 'Change Merge' can make to design.

## LONG TRANSACTIONS AND THE PARALLELISATION OF DESIGN

The idea of a transaction is pretty fundamental to maintaining the consistency of design data. However, the transaction only relates to an individual user. If we have multiple users (such as a project team) then there is the conflicting need to share data with one's colleagues but also to work at an individual level on a sub-set of the project data, possibly over an extended period of time.

As we know, it takes time to create or modify a design. During this gestation period, various design alternatives may be explored. During any one of these design cycles the design may be incomplete. The individual practitioner may not want to publish his work to his colleagues in the design team until his work is complete. In a 'shared' short transaction system, there is the possibility of all users seeing the changes of all other users immediately these occur. This is unrealistic and most probably undesirable. The user who generated a particular change does not want to publish incomplete intermediate solutions, nor does he wish to be bombarded with other users' incomplete intermediate solutions. The idea of a permanently shared repository with constantly updating short transactions is generally not thought to be appropriate to the work of design teams.

The idea of the long transaction is to enable users to work on a copy of all or part of a model for an extended timescale. The long transaction begins with the selection of the model or sub model to be worked on. The long transaction may include a whole series of short transactions, but these are completely private to that user. The long transaction finishes when the user commits his changed model back to the central shared data repository.

One approach is to adopt a pessimistic strategy where, for any shared data, there may be many readers but only one writer. The problem here is that the first user to 'grab' a particular data set (or file) for modification and update, essentially locks out the remaining users. Instead, we have adopted an optimistic strategy which allow the same data to be simultaneous modified by more than one user. There are three reasons for this:

First, not to allow an optimistic strategy would create a complete blockage to parallel working. Second, we are assuming that management will usually coordinate members of a design team so as not to allow users to make conflicting changes to the same data items. Third, we have created a technology which helps to resolve conflicting changes, if these should occur. This is not a closed system with hard coded rules. This technology enables future application developers to define their own rules for identifying and resolving conflicting changes, since what constitute a conflicting change depends on the specific context and application semantics.

Long transactions extend the concept of the transaction to a multi-user team and enables the parallelisation of design (which is fundamental for collaborative engineering).

## 'Change-Merge' and Coordination of Parallel Design Processes

As we have seen, long transactions allow multiple users to start parallel design sessions. At the end of these sessions, there is a need to gather these parallel 'strands' back into a single unified and resolved design statement.

One of the underlying themes which we are focussing on, is to maintain the consistency of design data. It is important to remember that in order to maintain this consistency, it is only possible to finish a long transaction (by committing the data

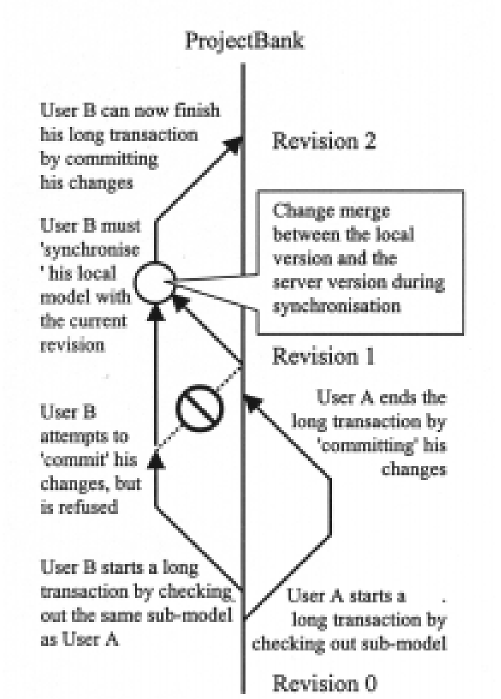

ProjectBank
User B can now finish his long transaction by committing his changes
Revision 2
User B must 'synchronise ' his local model with the current revision
Change merge between the local version and the server version during synchronisation
Revision 1
User B attempts to 'commit' his changes, but is refused
User A ends the long transaction by 'committing' his changes
User B starts a long transaction by checking out the same sub-model as User A
User A starts a long transaction by checking out sub-model
Revision 0

Figure 1:
Project Bank workflow

back to the data repository) if the differences between the local (workstation model) and the repository (server model) are due only to the changes that the user has made to his local model and that the data in the repository is unchanged since the start of the transaction.

Let us imagine that we have a ProjectBank repository with data at revision 0, and two users, A and B (Fig. 1). User A starts a long transaction, then user B starts another long transaction. At the end of A's long transaction he is able to commit his changes back to the repository, because the repository is in the same state (revision 0) as when A started his long transaction. The only changes that are relevant have occurred on A's workstation.

When user A commits his changes to the ProjectBank repository a new revision (1) is created. The difference between revisions 0 and revision 1 represents only the changes made by user A. By committing his changes, user A has essentially published these to his co-workers.

User B is notified that there are changes on the ProjectBank repository that he does not have on his workstation.

Now user B wants to commit his changes. But the repository is at revision 1, and is no longer in the same state as it was when B started his long transaction (which was at revision 0).

For a whole host of reasons, B cannot continue with his 'commit'. Instead he must "synchronise" his version of the data with the current state of the repository (including the recent changes committed by A).

The synchronisation allow User B to see (and possibly respond to) A's changes. However another reason for synchronisation is to prepare B's model to be 'committed' to the ProjectBank repository.

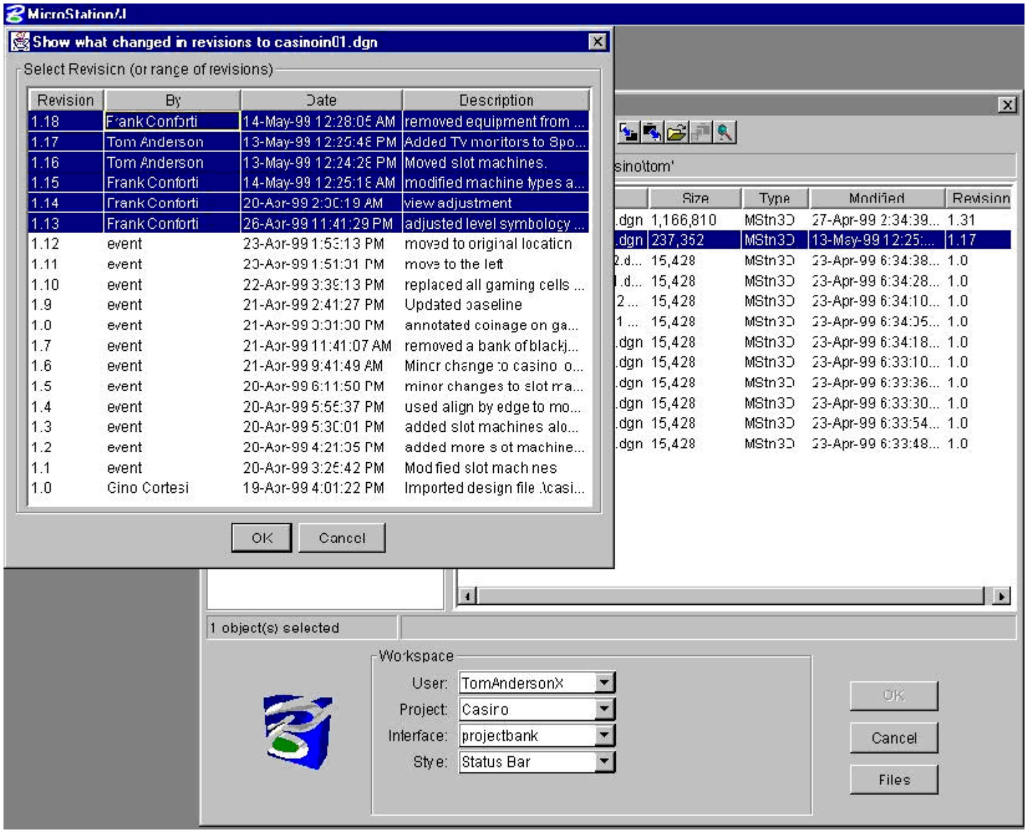
The pre-condition for a 'commit' is that the only difference between the ProjectBank repository and the model on User B's workstation are the changes made by B. So having synchronised with the current state of the repository, B is now free to commit his changes, resulting in a new revision on the ProjectBank repository (2).

So far we have described the activities of User A and B as 'changes' without going into the specifics. Essentially there are two types of changes: non-conflicting and conflicting. A non-conflicting change might occur when two users change some unrelated or incidental attribute of an objects, for example, one user changes the location of an item, while another user changes the item's colour. A conflicting change might occur when two users each make different changes to the same attribute of an objects or an inconsistent change to two (or more) related attributes of an objects, for example, one user changes the length of a beam and another user changes its depth. What constitutes a conflicting or non-conflicting change depends on the application semantics and the related validation logic.

In creating an application schema, we require the software developer to implement a 'changeMerge' method. This method can encode the rules which allow alternative versions of an object to be queried to determine if they are essentially compatible (and therefore can be automatically merged) or are incompatible. If the later is the case, then both versions are presented to the user who can choose which version should be accepted.

We can see that that the change-merge process is not (and necessarily cannot be) completely automatic, particularly when conflicting changes are encountered. Resolution of conflicting changes requires human intervention. Therefore this is a process which must be carried out, not just with the user's participation, but under his control (on the client application). It cannot be carried out on the server side (within the repository). Therefore the user must update (or synchronise) his version of the model, by down loading the current state of the model from the repository and resolving any conflicting changes locally.

To help in this process the user has access to a


MicroStation/J

Show what changed in revisions to casinoin01.dgn

Select Revision (or range of revisions)

| Revision | By | Date | Description |
|---|---|---|---|
| 1.18 | Frank Conforti | 14-May-99 12:28:05 AM | removed equipment from ... |
| 1.17 | Tom Anderson | 13-May-99 12:25:48 PM | Added Tv monitors to Spo... |
| 1.16 | Tom Anderson | 13-May-99 12:24:28 PM | Moved slot machines. |
| 1.15 | Frank Conforti | 14-May-99 12:25:18 AM | modified machine types a... |
| 1.14 | Frank Conforti | 20-Apr-99 2:30:19 AM | view adjustment |
| 1.13 | Frank Conforti | 26-Apr-99 11:41:29 PM | adjusted level symbology ... |
| 1.12 | event | 23-Apr-99 1:53:13 PM | moved to original location |
| 1.11 | event | 23-Apr-99 1:51:31 PM | move to the left |
| 1.10 | event | 22-Apr-99 3:39:13 PM | replaced all gaming cells ... |
| 1.9 | event | 21-Apr-99 2:41:27 PM | Updated baseline |
| 1.0 | event | 21-Apr-99 0:01:30 PM | annotated coinage on ga... |
| 1.7 | event | 21-Apr-99 11:41:07 AM | removed a bank of blackj... |
| 1.6 | event | 21-Apr-99 9:41:49 AM | Minor change to casino o... |
| 1.5 | event | 20-Apr-99 6:11:50 PM | minor changes to slot ma... |
| 1.4 | event | 20-Apr-99 5:55:37 PM | used align by edge to mo... |
| 1.3 | event | 20-Apr-99 5:30:01 PM | added slot machines alo... |
| 1.2 | event | 20-Apr-99 4:21:35 PM | added more slot machine... |
| 1.1 | event | 20-Apr-99 3:25:42 PM | Modified slot machines |
| 1.0 | Gino Cortesi | 19-Apr-99 4:01:22 PM | Imported design file .\casi... |

sino\tom'

| | Size | Type | Modified | Revision |
|---|---|---|---|---|
| .dgn | 1,166,810 | MStn3D | 27-Apr-99 2:34:39... | 1.31 |
| .dgn | 237,352 | MStn3D | 13-May-99 12:25:... | 1.17 |
| 2.d... | 15,428 | MStn3D | 23-Apr-99 6:34:38... | 1.0 |
| 1.d... | 15,428 | MStn3D | 23-Apr-99 6:34:28... | 1.0 |
| 2 ... | 15,428 | MStn3D | 23-Apr-99 6:34:10... | 1.0 |
| 1 ... | 15,428 | MStn3D | 23-Apr-99 6:34:05... | 1.0 |
| .dgn | 15,428 | MStn3D | 23-Apr-99 6:34:18... | 1.0 |
| .dgn | 15,428 | MStn3D | 23-Apr-99 6:33:10... | 1.0 |
| .dgn | 15,428 | MStn3D | 23-Apr-99 6:33:36... | 1.0 |
| .dgn | 15,428 | MStn3D | 23-Apr-99 6:33:30... | 1.0 |
| .dgn | 15,428 | MStn3D | 23-Apr-99 6:33:54... | 1.0 |
| .dgn | 15,428 | MStn3D | 23-Apr-99 6:33:48... | 1.0 |

1 object(s) selected

Workspace

User: TomAndersonX

Project: Casino

Interface: projectbank

Style: Status Bar

Figure 2:
ProjectBank 'Element History' function allows the user to review changes to a specific item or component or group, on a change by change basis and to precisely identify "who did what, when". It provides a complete 'audit trail' of the design process.

Revision Explorer, which identifies unchanged elements (in grey), added elements (in green), deleted elements (in red), changed elements, in their pre-changed state (in light blue) and in their post-changed state (in dark blue). The user is then in a position to commit his changes, because at this moment the only difference between the version of the model on the repository and his (client) version are his uncommitted changes.

## Revisions and Design History

Every time a user 'commits' to the ProjectBank repository a new revision is created. This is not a complete copy of the model, but rather the changes (or 'delta') from the previous revision. There are two advantages here: First, the storage is much more compact than the equivalent set of files (which would have laboriously duplicated all the unchanged data as well). Second, ProjectBank explicitly records all changes to all items at the 'component' level, not at the file level. This means that the user can use the 'Element History' function (Fig. 2) to review changes to a specific item or component or group, on a change by change basis. In addition the user can compare the complete design model or project at different moments in time, on a component by component basis. As we can see ProjectBank is NOT a document management systems.

This functionality opens up important management scenarios. For example, an item, which had been deleted in a previous revision, is not permanently lost, because it is likely to be available in the preceding revision. In fact, it can be effectively undeleted, by being brought forward from the revision preceding its deletion to the current version of the model. The generality of this is that the user can construct a new version of the model, by selectively including items from any previous revision. When 'committed' to ProjectBank, this becomes the latest revision, but all the previous revision are still intact.

The ability to 'play with' design history in this way helps managers to monitor the evolution of design, to identify and value the contribution of individual users, to combine features from alternative design and to re-use previously discarded concepts.

## Conclusions

The fundamental advantage that ProjectBank offers is the opportunities for greater consistency and control of information. In particular the 'annotation' of transactions naturally provides an audit trail. The explicit parallelisation of design opens new management possibilities, with the opportunities for 'ownership' of this management function. It is going to be interesting to observe how this type of collaborative engineering system influence the design process.

*Robert Aish M.Des.(RCA), PhD.*
*Director of Research, Bentley Systems*
*robert.aish@bentley.com*