Cooperative Data Manipulation in a Low-Connectivity Environment

R.H.W. Graves
Chief Technical Officer, Davion Systems Ltd
Ottawa, Ontario, Canada
www.davion.com

Abstract – A method is presented whereby a workgroup of any size can cooperate on the creation and maintenance of a one-to-many hierarchical database in the absence of a connecting network. Sections of the data structure are separated out as independent subordinate files ('subfiles'), together with all the necessary supporting data. Simultaneously, a set of parameters is created, known as a subfile lookup table, containing the information required to re-integrate the subfile back into the parent file. The data structure of a subfile may be expanded in isolation before being imported back into its parent file. Any subfile may in turn separate out one or more parts of itself as lower-level subfiles, and this process may be continued indefinitely, to as many levels as are required. The method by which subfiles are transmitted to and from their parent files is immaterial. If a network is used for transmission, computers need only be connected to it for the brief time needed to transmit files.

1 INTRODUCTION[1]

Unauthorized access to data systems is a constant threat to most organizations. The root of the problem lies in the existence of the network connecting the computers. If that network is connected to the outside world, i.e. is not merely an isolated intranet, then sooner or later, no matter what defences are employed, it is possible that a sufficiently funded, motivated and competent interloper will find a means of entry.

Since networks themselves are the weak points in data systems, we asked ourselves whether it would be possible for a geographically dispersed workgroup to cooperate on a common project without constant access to the same, or indeed any, network. The term 'geographically dispersed' indicates a workgroup not located in the same room or the same building, but spread out over an arbitrary set of physical locations.

To introduce this subject, consider how a group of individuals would typically cooperate on a data development project such as an engineering design exercise. The work is normally split into a number of sections. Each section is worked upon by one or more individuals, and these sections are then recombined into the final product.

Splitting up and subsequently recombining sections of work which are not tightly integrated together is not a major problem. To take a simple case, a document such as a report can be divided into several chapters, each chapter can be written by a different individual, and the entire document can then be assembled using a word processing program. Difficulties only arise when the work is based upon a tightly integrated structure and/or includes a common database to which individuals have the ability to make additions and changes.

One solution to this data management problem is to maintain a central or master file containing the structural and other common data, and forbid all changes to this data outside the master file. Sections of the work which are split off (subordinate files, or 'subfiles') would then contain a copy of this data, but would be unable to make changes to it. However, this would introduce an unwanted rigidity into the system with constant requests from subfile owners for data changes and additions, and the necessity of downloading them in a timely fashion from the master file to the subfiles. The process described in this paper permits additions and modifications to structural and other common data to be made within subfiles, together with a method whereby these additions and modifications can subsequently be re-integrated into the master file.

2 GENERAL APPROACH

Cooperative processing activities involving temporarily disconnected users have been studied in some detail. Typical processing models include the cooperative activity (CoAct) model [1], and its development into forms suitable for disconnected users [2]. Various other forms of data collection and processing in the presence of intermittently-connected networks have been proposed, such as Spray and Wait [3] and Intermittently Connected Embedded Database [4]. Volatile networks, i.e. networks that do not necessarily retain fixed connectivities, have been studied in some depth, such as TelegraphCQ [5].

All of these models assume that the basic structure of the database pre-exists, and any transactions that take

[1] Presented at the IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2014), Toronto, Canada, May 4-7, 2014

place are confined to inserting data into pre-defined slots, or modifying existing data. The purpose of the present model is to provide a means whereby a hierarchical database can be created *ab initio* by a number of disconnected users, for such uses as the development of an engineering design or the development of costing estimates for a large project. The term 'disconnected' is taken here to mean users who only intermittently access a network, or have no access at all but rely on other means of communication.

The approach taken assumes that the work product can be described in terms of a one-to-many hierarchical structure. The process described in this paper allows any node of the structure to be 'hived off' into an independent subfile and given to another person or organization for completion. At the same time a set of parameters is created, known as a subfile lookup table, containing the information required to re-integrate the subfile back into its parent file. This paper will describe a typical data structure and set of database processes required to support this method of network-independent workgroup cooperation.

2.1 Multi-Level Subfile Creation

Any subfile may hive off one or more parts of itself, so the whole process can be regarded as one of successive delegation of parts of the work. The resultant constellation of subfiles will constitute another one-to-many hierarchical structure (Fig. 1), independent of the work product structure.

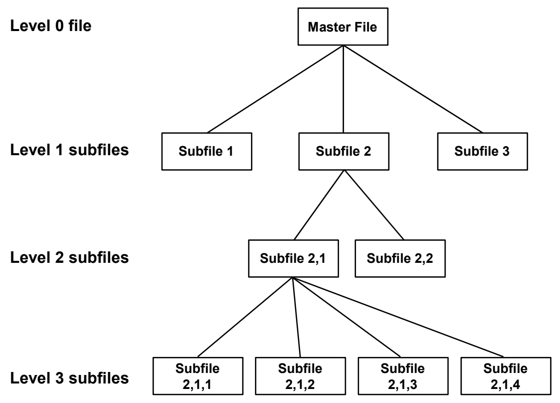

Fig. 1. Typical hierarchical subfile structure. Subfile numbers use comma separators to avoid confusion with node numbers.

2.2 Multi-Level Imports

Subfiles are normally imported, i.e. re-integrated, into their immediate parents. However, it is useful to be able to import a subfile directly into a more distant ancestor. Suppose, for example, that new data is developed in a level 3 subfile (Fig. 1) which is required urgently in the top-level master file. The present model allows such multi-level imports, provided that all the intermediate single-level, child-to-parent imports have already occurred.

2.3 Subfile Transmission

The method by which a subfile is transmitted to and from its parent file is immaterial. Note that if a network is used for transmission, computers only need to be connected to the network for the brief time needed to transmit the relevant files.

3 DATA STRUCTURES

The data within any one file, whether master file or subfile, necessary to support these processes will include:

- Structural data
- Node-specific data
- Common data
- Subfile lookup tables

3.1 Structural Data

Structural data defines and controls the database hierarchical structure. In any one file, structural data is contained within a s*tructure table*. Each record within the structure table represents a single node of the database hierarchical structure.

System Numbers

Each structure table record is assigned a unique *system number.* Each record contains a link to the record of the parent of the node that it represents, and links to the records of any children of that node. Links are specified in terms of system numbers. While subsequent additions, movements and deletions can alter the position of nodes within the hierarchical structure, system numbers, once assigned, never change within a structure table.

Node Numbers

Node numbers, such as 1.2.3, are simply addresses which indicate the current position of a node in the hierarchical structure, and as such are subject to change. Node numbers are not a necessary part of structure table records, since the position of an item in its parent's child list defines the last component of the corresponding node number. The node number relating to any record may therefore be inferred by tracing its parent links all the way up to the topmost record in the structure.

3.2 Node-Specific Data

Each node can be regarded as a container for data. Data associated with individual nodes is known as *node-specific data* and is situated in a separate table or set of tables or other data structure, depending on the quantity and type of data to be stored, known generically as the *node data table*. If data exists for a particular node in the node data table, the address for that data will be stored in the corresponding record in the structure table.

3.3 Common Data
Common data consists of non-structural data which can apply to any node. For example, in an engineering design application, the tensile strength of a particular steel alloy would be common data since it might be used in several different nodes. Common data is maintained in a table or set of tables or other data structure, depending on the quantity and type of data to be stored, known generically as the *common data table*.

3.4 Subfile Lookup Tables
A subfile lookup table, or subfile LUT, is a collection of data that is used to control the importation of a subfile into its parent or ancestor. A unique subfile LUT will exist for every subfile for every import transition, such as subfile-to-parent, or subfile-to-ancestor. Subfile-to-parent LUTs are created when the subfile itself is created, subfile-to-ancestor LUTs are created as and when required. Each subfile LUT will contain the following data:

- Subfile number, defining the subfile to which this subfile LUT relates (see Fig. 1);
- System number map, which maps system numbers in the subfile to the corresponding numbers in the parent file;
- Common data maps (one for each distinct data type), mapping field numbers in the subfile to the corresponding numbers in the parent file;
- Destination level, defining the target file (parent or more distant ancestor) to which this subfile LUT relates.

A copy of each subfile LUT is maintained on the subfile itself and on its target, i.e. import destination, file.

4 DATABASE PROCESSES
Development of a work product using this method of network-independent workgroup cooperation involves six distinct activities:

1. *Master file establishment*: creating a master file representing the totality of the work to be performed, albeit initially in very abbreviated form;
2. *Exporting*: splitting the work into discrete packets by hiving off subfiles from the master file, and if necessary, hiving off further subfiles from those subfiles, to as great a depth as required;
3. *Fleshing out*: entering data into individual subfiles, which activity may be performed in isolation by separate individuals or workgroups;
4. *Importing*: recombining data into a coherent whole by importing subfiles back into their parent files;
5. *Data dissemination*: (optional) dissemination of non-structural common data between subfiles by downloading an updated set of common data to each subfile as it is imported back into its parent.
6. *Subfile updating*: transmitting an updated version of the subfile back to its owner, if any changes have occurred which need to be registered in the subfile.

4.1 Master File Establishment
The master file represents the totality of the work to be performed, and will eventually contain the data for the completed project.

Master files can be created *ab initio*, in which case they will simply contain a structure table with a zero-level node in it. Alternatively, a new master file can be created from an existing master file if a similar project is to be repeated. In such case all node-specific data and subfile LUTs are deleted. The structure table may optionally be deleted as well, or it may be retained and modified for use in the new file. All the non-structural common data in the old file is retained.

Each master file is given a unique recognition code, such as a random alphanumeric string. This code is copied to all subfiles derived directly or indirectly from that master file, thus enabling them to be recognized as part of that master file family.

4.2 Exporting
Exporting is simply a method of delegating sections of the work into successively smaller parcels.

For example, as shown in Fig. 2, if the 2.0 node of the master file is exported as a subfile, then the section of the work represented by that node has, in effect, been delegated to another person or organization. The recipient of this level 1 subfile is then at liberty to develop the hierarchical structure descending from the 2.0 node, and delegate in turn any of the new nodes by exporting level 2 subfiles in a similar process. The decision to export a new subfile can be taken at the local parent file level, and does not need to be planned in advance at the master file level.

4.2.1 Subfile Creation
A new subfile is created by making a copy of the parent file and then deleting all node records in its structure table except for the selected node and any descendants it may already have. For reasons of operational efficiency, all node records remaining in the new subfile are assigned new, consecutive system numbers beginning at 1. The common data table is normally copied as is to the new subfile, although facilities are provided for removing any sensitive or unwanted data. All node data table records in the new subfile are deleted, save for any that are associated with the selected node and its descendants. A subfile LUT for the subsequent importation of the subfile back into its parent is created and stored on both the subfile and the parent file.

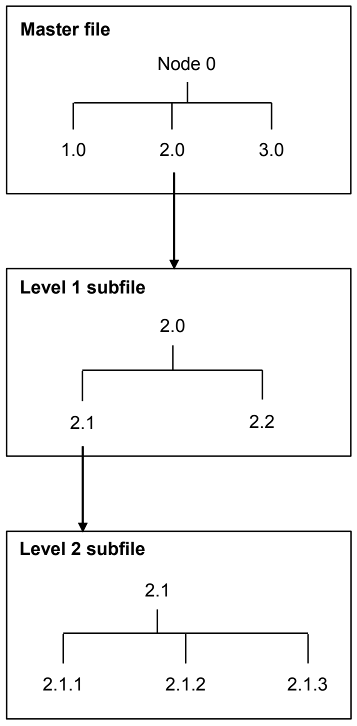

Fig. 2. Multiple subfile levels

4.3 Fleshing Out

The fleshing out process consists of entering or modifying data in individual subfiles, which activity may be performed in isolation by separate individuals or workgroups. Subfile owners may add descendant nodes within their subfiles, insert node-specific data, and add or modify common data. They may also export lower-level subfiles for fleshing out by other individuals or workgroups.

4.4 Importing

Importing consists of the integration of data in a subfile into its parent or ancestor file. The import process copies all nodes in the subfile into the parent or ancestor file, together with any node-specific data, and any new or modified common data.

The import process requires the following steps. The term *source file* refers to the file which is being imported, while the term *target file* refers to the file into which the source file is being imported:

- *Checking for importability* - checking that the source file is legally importable into the target file;
- *Checking for acceptability of common data* - checking that any differences between source file common data and its counterparts in the target file are reconcilable, and performing reconciliation as necessary;
- *Updating the system number map* - updating the system number map in the subfile LUT to account for any changes to both source file and target file structure since the source file was created or last imported;
- *Importing node structural records* - importing all node records in the source file structure table into the target file structure table;
- *Importing common data* - importing the common data from the source file common data table into the target file common data table;
- *Updating subfile LUTs* - updating the subfile LUT for this import process to accommodate any new nodes and any new common data fields, and importing any lower-level LUTs;
- *Importing node-specific data* - importing all node-specific data records in the source file node data table into the target file node data table.

4.4.1 Multi-Level Imports

The import process can proceed only if maps exist defining relationships between source and target node system numbers, and between source and target common data field numbers. If the source file is an immediate child of the target file, such maps will exist in the subfile LUT that was created when the source file was first exported, and subsequently updated. If the source is a more distant descendant of the target, then the requisite maps must be synthesized from a sequence of parent/child maps spanning the range from the current source to the current target, by tracing system numbers and field numbers up the sequence from source to target. Thus, in order for a multi-level import to take place, all the single-level imports spanning that range must already have occurred, so that the single-level maps for each step will exist.

4.4.2 Prior Importation

In a multi-level system of subfiles it is possible for a node to be imported into a parent file by more than one route, as shown in Fig. 3.

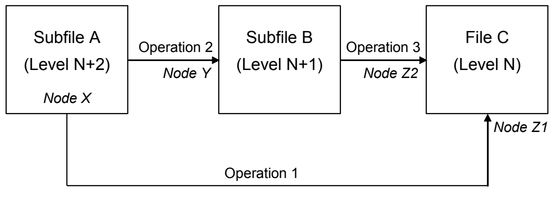

Fig. 3 Prior importation of a node from a low-level subfile into its ancestor, followed by stepwise importation via its immediate parent, resulting in two different system numbers in the ancestor file

A node having system number X in the low-level subfile A is imported directly into file C via a multi-level import (operation 1), where it is assigned system number Z1. The same node is then imported at a later date from subfile A into subfile B (operation 2), where it is assigned system number Y, and from there into file C (operation 3), where it is assigned system number Z2, since there is no obvious connection at this point with its earlier import as Z1.

In order to determine that Z1 and Z2 are the same, it is necessary to check the multi-level A-to-C subfile LUT which maps file A system numbers to file C numbers. In more general terms, when any import operation takes place, a search is made in the target file for any subfile LUTs containing precursor system number maps of the form M-to-T where T is the target file and M is a descendant of the current source file. In this case, when the B-to-C import occurs, the A-to-C data map would be identified as a precursor and hence all new node system numbers in file B (i.e. not referenced in the existing B-to-C system number map) are checked in the most recent A-to-B data map to see if they originated in file A. If a node is found to have originated in file A and is also referenced in the A-to-C data map, the established target system number in this map will be used for that node in the B-to-C import process rather than assigning it a new system number.

This process necessitates that when any import occurs, all subfile LUTs in the source file for all transitions into that file are copied into the target file for possible later use.

Prior importation also applies to common data fields, and is treated in a similar fashion.

4.5 Data Dissemination

Common data from a constellation of subfiles may be disseminated to each individual subfile by downloading new common data items from the target file to each source file as it is imported.

The import process will result in a parent file accumulating all the common data originating in all of its descendant subfiles. If a subfile is being imported and does not contain the complete set of common data in the target file, this data set may optionally be downloaded to that subfile after its importation is complete. Common data can therefore be disseminated across a constellation of subfiles by a process which can be described as 'discrete diffusion', in the sense that updates take place as and when subfiles are returned to their parents or ancestors for importation. A necessary part of this process is that subfiles updated in this way are then returned to their owners.

4.6 Subfile Updating

At the end of the importation process the operator will be informed if any changes have been made to the source file, either by updating the subfile LUT, or by a changed value for one or more common data items, or by dissemination of common data. If any changes have been made it will be the operator's responsibility to transmit the updated source file back to its owner, by any convenient means.

If the updated source file is not transmitted back to its owner, or if it is transmitted but its owner neglects to replace the original file with the updated file, no lasting harm will have been done, but the data renewal processes described above will simply reoccur if the source file is imported again for update purposes.

If the subfile LUT was updated in the import process, but the new version is not written onto the original subfile for either of the reasons mentioned above, problems can arise during subsequent importations because nodes or fields which had already been imported will not be recognized as having been imported. For this reason, a copy of the LUT is maintained on both the parent file and the subfile, and the latest version of the two is always used to control the import process.

5 CONCLUSION

It has been shown that it is possible for a widely dispersed work group to cooperate in the creation of a hierarchical database in which the computers involved are only intermittently connected to a network, or indeed not connected to a network at all. The model described herein is a simple one, but is believed to cover all the essential points needed for such cooperative activity.

REFERENCES

1. "Towards a Cooperative Transaction Model - The Cooperative Activity Model", M. Rusinkiewicz, W. Klas, T. Tesch, J. Wasch and P. Muth, Proceedings of the 21st VLDB Conference, Zurich, 1995
2. "Enabling Cooperation among Disconnected Mobile Users", J. Klingemann, T. Tesch and J. Wasch, Proceedings of the Second IFCIS Conference on Cooperative Information Systems, Kiawah Island, South Carolina, June 24-27, 1997
3. "Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks", T. Spyropoulos, K. Psounis, C. Raghavendra, SIGCOMM '05 Workshops, Philadelphia, Aug 23-26, 2005
4. "ICEDB: Intermittently-Connected Continuous Query Processing", Y. Zhang, B. Hull, H. Balakrishnan, S. Madden, IEEE 23rd International Conference on Data Engineering, Proc ICDE 2007, pp. 166-175
5. "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World", Sirish Chandrasekaran et el., Proceedings of the 2003 CIDR Conference.