Versioning and Concurrency Control in a Distributed Design Environment

Atsushi Takahara

LSI Design Systems Laboratory NTT LSI Laboratories
3-1 Morinosato-Wakamiya, Atsugi-shi, Kanagawa-ken, 243-01 JAPAN

Abstract
This paper describes a design data management method in a distributed design environment. This method is based on the detection of the difference between the original design data and modified design data. One of the advantages of this method is that the precise difference can be identified among versions. Exploiting this feature, the history of modifications of design data can be traced and the required storage space can be reduced to less than that with previous methods. Another advantage is that efficient concurrency control is achieved by detecting the difference between design data.

1 Introduction
In the CAD Framework, Design Data Management is one of the key mechanisms for integrating CAD tools. Currently, design activities are being carried out in multi-user and multi-tasking environments as well as in distributed environments. In a distributed environment, Design Data Storage Management [7] is a crucial problem because design data have various relations and these design data are shared by several users or several tools at the same time. To support the management of various design data, design data must be well organized. Currently, UNIX file systems are used to store design data, but it is difficult for several tools or users to share one UNIX file at the same time. Also, various relations in design data are poorly managed. Database systems are alternatives for managing design data. The data structure of a Relational Database system is different from the design data structure used in CAD tools. For example, the table structure of a Relational Database system has insufficient modeling ability for use in a CAD environment. In recent years, Object Oriented Database (OODB) systems [3] have been attracting attention. OODB systems support a richer modeling capability than a table structure and a better programming interface than conventional query languages, but they lack the efficiency required to manage a large amount of data. To ensure greater efficiency, CAD specific data management systems have been proposed [2, 4, 5, 6].

A design data management system must be capable of efficiently handling a large amount of data. The volume of data is huge because many versions are created and maintained as design data throughout the evolution of a design. In those different versions, it is rare that all the data are different. Therefore, it is desirable to detect what is a difference between versions for further reference. To save storage space, it would also be helpful if only the differential parts are stored as a new version. In Version Server [2], the configurations of versions are managed well using a tree structure, but there is no way to detect the differences between versions.

Another required feature is "Concurrency Control". Concurrency Control handles two cases: One is the independent access to design data. For instance, two different tools use a module at the same time but there are no mutual relations between those tools. The other is concurrent accesses from tightly related tools to the same data. For instance, when both a schematic editor and a simulator use a module at the same time, the simulator can run soon after the design data is modified by the editor. In the former case, a locking mechanism can be used, but the current locking method is not suitable for managing VLSI Design Data. This is because a CAD tool tends to lock a design data for a long period of time, which forces other tools to wait. Wolf et al. proposed a concurrency control method in which the locking mechanism is combined with a versioning method to improve the performance of concurrency control [8]. However, the locking method has an overhead and is visible to users when a CAD tool accesses data. If possible, such a concurrency control should be invisible to users. In the latter case, in Octane [5] (an enhanced version of OCT [1]), the original data is copied to a "workspace" for each tool when the design data is used at the same time. In this method, it is difficult to exchange modifications among tools because the copied data is independent in each tool.

In this paper, we propose a new versioning and concurrency control method which is used to implement our Distributed Design Data Management System (D3MS). Features of D3MS are 1) A mechanism that detects the precise difference between design data, 2) An efficient versioning mechanism in which only the differences between the previous version and the current version are stored as a new version, 3) An efficient concurrent access control mechanism based on a capability to detect differences between design data rather than using the locking method.
2 Overview of D3MS

D3MS is a Server-Client style design data management system (Fig.1). To support the handling of concurrent accesses, a Server-Client scheme is needed to arbitrate the accesses and control versions of design data in a distributed environment. A server is a process which handles persistent design data and arbitrates requests from many CAD tools. A client is a CAD tool which uses D3MS database access routines to access design data in a server when it needs the design data.

A server consists of three modules: a Server Communication Module (SCM), a SNBT (Server Name Binding Table, introduced in section 3.1) Handling Module (SHM) and a Cache Module (CM). The SCM handles requests from clients and sends the design data to clients. The SHM maintains versions and rebuilds the current version from the differential versions. The CM has the design data cache for persistent data to enhance access performance.

A client also consists of three modules: a Client Communication Module (CCM), a NBT (Name Binding Table, introduced in section 3.1) Handling Module (NHM) and a CAD Application Module. The CCM communicates with servers and other clients. The NHM maintains the client data which are loaded from a server or created by a client.

3 Functions of D3MS

In this section, we first describe the structure of design data (Section 3.1) and explain how versioning and concurrency control are supported in D3MS (Section 3.2 and 3.3).

3.1 The Structure of Design Data

To detect differences in design data, we focus on the relationship between a runtime data and a persistent data because a client modifies a runtime data, while this data is stored as a persistent data in a server. The key to expressing this relationship is the appropriate handling of the pointer structure of data because the pointer structure is frequently used in design data. In our method, we provide a table called the "Name Binding Table" (NBT) (Fig.2(a)). This table indicates the relation between the pointer used in a tool and the "Node name" used as a global notation. This table is also used to detect modifications.

The structure of runtime design data is a set of "Node"s where each "Node" has a unique name. A "Node" consists of a set of "Value"s and a set of "Pointer"s. Figure 2(b) shows an example of a Runtime Design Data Structure for expressing a netlist. Logic gate G1 is expressed by the "Node" whose "Node Name" is Node1. The gate name and its function are described as "Value". The connections of gates are expressed as "Pointer"s.

The persistent design data are sequential files, and previously defined "Node"s are saved in a file as an array of nodes like that in Fig.2(d). "Value"s and "Pointer"s of a node are sequentially mapped on the persistent data. In this persistent data structure, an integer value represents a "Node" location in the file.

In the Server-Client model, the design data need to be queried by an identifier that is unique in the distributed environment. This query is easily performed using the NBT. In a server, the design data are managed using a Server Name Binding Table (SNBT) which indicates the relation between a node name and the location of a "Node" in the persistent data (Fig.2(c)). When a client loads a "Node" from persistent data, it gives a node name to a server. The server looks up the SNBT to load a "Node" from persistent design data then a "Node" is sent to the client.

3.2 Differential Versioning

In the versioning of D3MS, the differences from a prior version are only saved to create a new version. The updated data are appended to the original data when they are stored in a server. Thus, the previous data are still kept in persistent data after the updated data are stored. Each version is memorized as a SNBT which depicts the difference between previous and current versions. We call this versioning method "Differential Versioning" (DV).

DV has three steps.

1. When a tool modifies its own data, this modification is memorized in the NBT.
2. When a tool saves its data, it sends the data of modified nodes to a server by referring to its NBT.
3. In a server, the data of modified nodes are appended to a persistent data and tuples of modified nodes and their position in a persistent data are saved in a SNBT. This SNBT memorizes a created version.

The three versions are depicted in Fig.3. V1 shows that the net with node name “D” is added to the Original. In this case, the node “D” is appended to the Original persistent data. The {D, 4} is saved as the SNBT which expresses a V1 and only modified data of “D” is added to original data. Here, the tuple of {D, 4} means that the node data of “D” is located at the 4th position in the persistent data. In V2, which is produced from V1, the gate with node name “A” is modified. At this moment, the new node “A” is appended to the persistent data and the tuple of {A, 5} is saved as a SNBT. In the persistent data, both the previous node “A” and the new node “A” are preserved.

Here, the persistent data of V2 has three versions (Original, V1, V2). When a previous version is referred to after finishing the modification of V2, the Original SNBT and V1’s differential SNBT are used. For instance, all the data of V1 are recreated by loading a persistent data in accordance with the original SNBT and V1’s SNBT.

DV enables (1) detection of the differences in versions and (2) reduction in the volume of versioning data. The differences between versions can be detected when they are referred to later and the volume of these different versions of data is kept small to facilitate the handling of design data and save disk space.

3.3 Concurrency Control

In a distributed environment, several CAD tools use the same persistent data concurrently. In D3MS, each tool has its own workspace data as runtime data. Here, a crucial problem arises when a tool saves its workspace in a distributed environment. If one tool modifies and saves its workspace in persistent data while other tools are still using them, it makes the persistent data inconsistent.

In D3MS, DV is also used to avoid these inconsistencies. Figure 4 shows how our method works. Tool-A and Tool-B are accessing some persistent data at the same time. Node “A” stands for net N of the Netlist A. Here, Tool-B changes the connection of net “N” from gate G1 to gate G2. In this case, the “Pointer” which stands for gate G1 is changed in Tool-B. This causes Node “A” to be modified in Tool-B. On the other hand, in Tool-A, net N is still connected to gate G1. At this moment, Tool-B saves its design data by DV. That is, Tool-B makes a new version and Tool-A refers to a previous version. Although Tool-A and Tool-B refer to different versions, both the persistent data of “A” and the persistent data of “new A” exist in one persistent data. Even if both tools finish their processing, the persistent data holds both data. Such handling cannot be achieved using the conventional “locking method”.

If tools have mutual relations, a method for Consistency Maintenance among accessing tools is required. In Fig.4, Data “A” is therefore inconsistent. To make this situation consistent, Tool-B should notify Tool-A of the changes in “A”. There are two ways to accomplish this:

1. Notification of a tuple of the NBT and its “Node” data. In Fig.4, Tool-A is notified of both
the node data of "new A" and a tuple of NBT for that data.

2. Notification of only a tuple of the NBT. Here, only a tuple of NBT for the "new A" is sent to Tool-A. Tool-A memorizes this tuple. When Tool-A accesses the "new A", Tool-A loads the data from the persistent data in accordance with the tuple of the NBT sent from Tool-B.

If the first approach is used, the data is updated immediately. If the second is used, only the tuple of NBT is modified and the data is loaded from the persistent data in a server later. If Tool-A keeps the original data, Tool-A ignores this update information from Tool-B to keep Tool-A's own data. In the second case, the update information is selectively used by the tool which receives notification of modification.

D³MS provides a more efficient concurrent access mechanism than is possible with the locking mechanism. With this method, several tools can share the same design data independently without affecting each other because one tool does not wait until the access of another tool terminates.

4 Implementation and Evaluation

We have experimentally implemented D³MS on UNIX based workstations (SUN SparcStation). The program is written in C++.

Figure 5(a) shows the effect of differential versioning when this method is applied to practical netlist data. In DV, the volume of data is much smaller than when DV is not used. This is because different versions of data are shared in the Differential Versioning method. Figure 5(b) compares the access speed with and without using the D³MS concurrent access method when one tool loads the netlist data which are modified by another. In D³MS, only the differences are transferred between tools. The smaller the modification ratio is, the more this concurrency control method accelerates reconstructing the netlist modified by another tool.

5 Conclusions

D³MS provides a method for efficiently sharing and exchanging data among different tools for both independently accessing tools and related ones. In "Differential Versioning", the required storage space is minimized because only differential data are saved as a new version. The modification history of design data can be easily traced to investigate differential versions. This is a desirable property in an environment where many versions have to be manipulated[2]. This versioning approach is also used for concurrency control. D³MS enables tools to access the same persistent data without those tools affecting each other. If tightly related tools are accessing persistent data at the same time, D³MS provides a method for exchanging only the modification between those tools. Thus, D³MS can provide an efficient distributed design environment.

References