Design And Cooperation Management Through Design Steps and Object Characteristics

Maria Aparecida Castro Livi & Cirano Iochpe

CPGCC-UFRGS
Caixa Postal 15064
91501 Porto Alegre RS
BRAZIL

e-mail: iochpe@inf.ufrgs.br

ABSTRACT: Conventional database management systems (DBMS) do not cope well with the requirement of cooperative work in the design environment (DE). They implement conventional, short transactions which run in isolation preventing designers from exchanging non-committed object updates. Cooperation models are being proposed which aim at supporting DE's dynamic properties (i.e. its processing model) especially with respect to both object-transfer operations and the definition of application-oriented correctness criteria for the database (DB).

This work gives an overview of the design management and cooperation model for CAD database systems being developed at the CPGCC-UFRGS and presents a mechanism which realizes it. The model supports object definition by means of application-oriented object characteristics (properties). Characteristics are integrated into objects through the execution of design steps. Design steps are related to one another forming partial orders which represent the design process. Relying on sets of object characteristics and partial orders of design steps, the DBMS's Design and Cooperation Manager (DCM) guarantees database correctness by controling the design process as well as object-transfer operations.

1. Introduction

Conventional DBMS implement the transaction paradigm [2] to support the application's dynamic properties. Application work must be modeled as (short) transactions which follow the ACID principles of atomicity, consistency, isolation, and durability [3]. In this context, database correctness can be guaranteed on the basis of serializable schedules of concurrent transactions [1]. Concurrency control techniques that rely on serializability need not take application semantics into consideration.

Processing models of design applications (e.g. VLSI design, software engineering) pose new requirements on the DBMS's transaction manager. Design work can last very long (e.g. days, months). From the application point of view, it is impossible to model it as an atomic transaction that must be completely undone in case of error or failure. Furthermore, the design work is a cooperative activity ([5], [6], [8]). Designers exchange information about their work in order to achieve common goals. Consequently, the requirement of isolation does not apply to design transactions.

The STAR environment for VLSI design [10] being developed at the CPGCC-UFRGS integrates a (long) design transaction manager [9] and a design and cooperation manager ([4], [6]). The latter implements an extension of the cooperation model proposed in [5]. It relies on sets of application-oriented object characteristics as well as partial orders of design steps to control the correctness of the design process.

After giving an overview of the design management and cooperation model (DMCM) proposed in [6], this paper presents the design and cooperation manager being implemented as part of the STAR project. The rest of the text is organized as follows. Section 2 discusses DMCM's main features. Section 3 presents the design and cooperation manager including its architecture and the information it works upon. Section 4 concludes the paper and points out to future work.

2. A Design Management and Cooperation Model

The model being developed at the CPGCC-UFRGS supports two views of the evolutionary character of design applications: object construction and design process development. Design objects are specified to the system by means of their (future) features which should be realized during the design process. The latter is modeled as a partial order of design steps. Each design step manipulates a set of objects and implements some of their (expected) features. Design steps incrementally change object states until objects reach their final states, that is, states in which objects incorporate all features declared in their specifications.

2.1 The Underlying Computer System Configuration

The model supposes a distributed computer system with a server-workstation architecture. Server nodes store application objects (e.g. designs, libraries, processing tools, and administration data). Designers work on private workstations connected to the server nodes through a communication subsystem (e.g. a LAN). Therefore, design work actually takes place at the workstation sites.

2.2 Transaction Management and Database Architecture

According to the hierarchical structure of design projects, the transaction model supports the specification of transaction hierarchies. A project transaction (TApj) represents the whole effort of a group of designers working on a common design. A project database (DBpj) is associated to every TApj at its start. When a project needs to access an application object that is not locally stored in its DBpj, the TApj issues a checkout operation in the public database (DBpu). Concurrent project transactions are isolated from one another through a, possibly non-strict, two-phase locking mechanism.

Each designer develops her/his work from inside a design transaction (TAd). Every TAd is related to a private database (DBpr). Since both design and project transactions are of long duration, they can be subdivided into sets of (possibly concurrent) ACID transactions.

Design transactions of the same project are considered to be nested transactions of the respective TApj. They cooperate with one another to carry out the design project. DMCM controls the information flow among design transactions.

The transaction and database hierarchies described above are mapped onto the computer network as follows. Project transactions execute at server nodes. Similarly, project databses and the DBpu are located at those processing nodes. Design transactions execute at workstation sites. Private databases are located there, too.

2.3 Object Specification and Object Design Evolution

The specification of a design object includes a set of features that represents the set of characteristics which must be present in the object at the end of the design process. Object features express design decisions about form, performance, functions, and correctness criteria. They can also represent the ocurrency of either a test or a simulation of an object. Moreover, the object specification can either establish or estimate values for the features (e.g. the area of a chip must end up between 2 and 3 cm^2). An Object specification can be updated at any time.

New features can be added to as well as existing features can be deleted from it. Designers can also change expected feature values at any time.

Object design evolution is captured by a partial order of object versions. Each version expresses one of the states which were already reached by the object due to the design process. Each state is represented by a specific set of (realized) object characteristics. Therefore, versions of a same object differ from one another either in the set of features they incorporate or in the values they set to them.

Object versions can assume three different development states:

- **in-work**: The version is kept in the private database of any of its creators and is not eligible for transfer operations. Therefore it can be deleted at any time without affecting other designers.
- **stable**: The version already incorporates a significant subset of its expected characteristics. Therefore, it can be lent other users to help them by their own designs. Stabel versions are kept in the project database.
- **consolidated**: The version presents all of its expected characteristics and its creators release it to public use (i.e. use in any project). It is transferred to the DBpu and cannot be deleted anymore.

Object specifications as well as object versions are given unique identifiers (surrogates). Therefore, there are three distinct ways to identify either a specific object version or a set of versions: through the unique identifier of the specific version; giving the id of an object specification together with a set of object characteristics (either related to specific values or not); giving only the id of an object specification to automatically retrieve its most actual version.

The use of object characteristics in the identification of versions enables designers to identify sets of versions that have the same subset of features or even object versions which have not yet been created. Both advantages can be explored by the cooperation manager to control the correctness of object-transfer operations (see below). Though, there are some disavantages in the use of object characteristics to identify versions. System implementation can be more complex and query processing can last longer.

2.4 Project Specification and Design Process Evolution

Usually, projects in areas such as VLSI design, software engineering, and mechanical parts design are very complex and tend to be recursively subdivided into subprojects until each design (sub)task achieves a complexity level that can be handled by a small group of specialists. DMCM views a design project as a hierarchy of design (sub)tasks (i.e. subprojects), each one of which develops one or more (sub)objects. The latter are integrated to form the final product of the design project (e.g. the design of a cpu could include the design of a main memory component, the one of a bank of registers, that of an ALU, and the bus design).

Each (sub)project is modeled by means of a set of object specifications and a project specification. The object specifications describe the tasks of the project. The project specification defines the way it should be carried out. A project specification consists of a partial order of design steps (DS). Each DS expresses a set of design activities which have a common meaning (e.g. form a specific design phase) for the application. During its execution, a design step can implement new object characteristics, test already existing ones, and inform designers about object design evolution.

An DS is responsible for the execution of a set of application-oriented design tools (e.g. editors, compilers, and simulators), each one of which accesses existing object versions and can generate new ones. The model supports human intervention to be represented as an application tool inside the step.

DMCM allows designers to define any execution order for the tools inside an DS. In addition, designers must point out the subset of object versions to be created inside a design step that

must be kept in the database at the end of its execution. The design and cooperation manager must pass this information on to the DBMS's version manager.

At DS definition time, designers must declare specific object versions as either input to or output of design tools. Relying on this information, DCM can infer the exact point in time when a specific version will be needed for cooperation. It can also infer when it will be produced as well as in which DS it will occur.

Not only its relative position in the partial order of design steps determines when an DS should be started. Designers can associate a start predicate to every design step. Such a predicate can relate the execution of its associated step to the realization of any object characteristic as well as to the evaluation of any query against the database. DCM can control design process evolution by keeping information about the directed graph of design steps as well as evaluating the start predicates of steps which are ready to execute.

DMCM allows step specifications to be updated at any time. Moreover, the model enables designers to suspend, resume, and abort step executions. Any of these operations on one DS can force DCM to abort or compensate for those design steps which depend upon that one and either executed already or are still executing.

2.5 Object-Transfer Operations

Cooperation among designers of a same project is supported by the design management and cooperation model on the basis of both object characteristics and object-transfer operations.

As mentioned above (see section 2.3), any designer can access object versions stored in the public database. Object versions in a DBpj can be accessed only by designers working on the respective project. Finally, versions stored in a private DB can be accessed by its owner only.

Since DMCM allows the design of an object to be carried out by more than one designer, the transfer of an object version between the private databases of two of its creators has not the same semantics as a transfer operation which lends a copy of an object version to a non-creator designer. While the prior operation prevents the first creator to contiue working with the version, the latter prevents the user of the version to update it, while the creator can continue designing the object (i.e. producing new versions of it). Therefore, transfer operations between creators transfer object versions from one DBpr to another. On the other hand, object-transfer operations between creator and user send copies of object versions from project databases to private ones.

Actually, the model supports two kinds of cooperation: between designers (i.e. creators of a same object) and between projects (creator-user relationship). The prior takes place inside the same project specification (i.e. partial order of design steps), every time a following design step is to be executed under the responsibility of another designer (possibly at another workstation). In this case, there is only one copy of an object version that is transferred from one database to the other.

Cooperation between projects implies the existence of, at least, two copies of a same version: one stored in a DBpj related to its creators and the others being used by non-creator designers in their own DBpr. DCM can detect when this kind of cooperation must occur by controlling the execution state of existing partial orders of design steps. Moreover, it can also notify creators of pending object-transfer operations so that they can eventually change the implementation order of some object characteristics.

Since new object versions can be created in parallel to the use of some older version of the same object, DCM must keep track of project cooperation. New versions should eventually substitute older ones at user sites. Object versions either composed by or whose design relies upon borrowed versions cannot be consolidated in the DBpu before those borrowed versions are. In addition, DCM must identify those design steps (of any existing project specification) which relied on any object version created inside a DS that has been aborted or compensated for. Responsible designers must decide whether affected design steps should be completely aborted or not.

The end of a cooperation at the project level is usually decided by designers responsible for the project specification that borrowed the object version(s). On the other hand, cooperation can either remain until the end of the whole project or be cancelled due to either DS abortion or version deletion.

3 A Mechanism for Design and Cooperation Management

This section discusses a mechanism which implements the design management and cooperation model described above. At present, a prototype of it is being developed as part of the STAR environment. The design and cooperation manager is being modeled on top of the KRISYS (Knowledge Representation System) [7]. Its routines are being written in LISP and C. We expect to have a running version of the DCM by the end of 1992.

To correctly support design process evolution, DCM must maintain information about object as well as project specifications. Since this information must be stored in the database but, at least in the present version of the mechanism, is not available to the application tools through the usual database interface, it is kept in a special repository called design and cooperation database (DBdc).

DBdc's conceptual schema is shown in Figure 1 as an E-R diagram where entities are drawn as rectangles and relationships as circles. The schema is known to the designers who create and update it during the whole design process. For this purpose, DCM supports operations to insert, update, delete, and query all information kept in the DBdc. DCM, too, updates the design and cooperation DB. It must keep track of state transitions on objects and steps as well as control both cooperation between designers and projects.

DCM works close together with two other STAR components: the transaction manager (which actually is being implemented as part of KRISYS) and the version manager. Partial orders of design steps are mapped onto (long) design transactions and the execution of application tools inside steps occur as short, ACID transactions. Only down to DCM's level, design steps are seen as application-oriented units of work. Object versions generated inside design steps are passed on to the version manager that builds (version) derivation graphs for each object specification at all levels of the database hierarchy.

Designers use DCM's services for two purposes: create as well as update DBdc and start as well as control the design process. In the following, on the basis of its data structures (see Figure 1) we discuss the main services to be implemented by the mechanism.

3.1 Defining Projects and Objects

DCM supports the partioning of projects into subprojects. This information is kept by the **projects** entity and the **subproject** relationship. Every project is related to a **design-specification** as well as a set of **object-specifications**.

Objects can be related to sub-objects through the **sub-object** relationship. Moreover, each object specification **incorporates** a set of **object-characteristics** and can be **represented-by** many different **versions**. DCM keeps information about version derivation by means of two relationships (to speed up retrieval): **derived-from** and **derives**.

Each design specification is related to a set of **design-steps**. The relationships **predecessor** and **successor** express the partial order of design steps. Each DS can be related to versions in three ways. It can **implement**, or be **waiting** for, or **borrow** object versions. While the first relationship is established at DS definition time, the other two ones are inserted into DBdc during the design process. Steps are related also to application tools (by the **activate** relationship).

Tools either **realize** or **test** (i.e. verify/simulate) object characteristics while creating new object versions. Versions can both **present** characteristics and set up values to them.

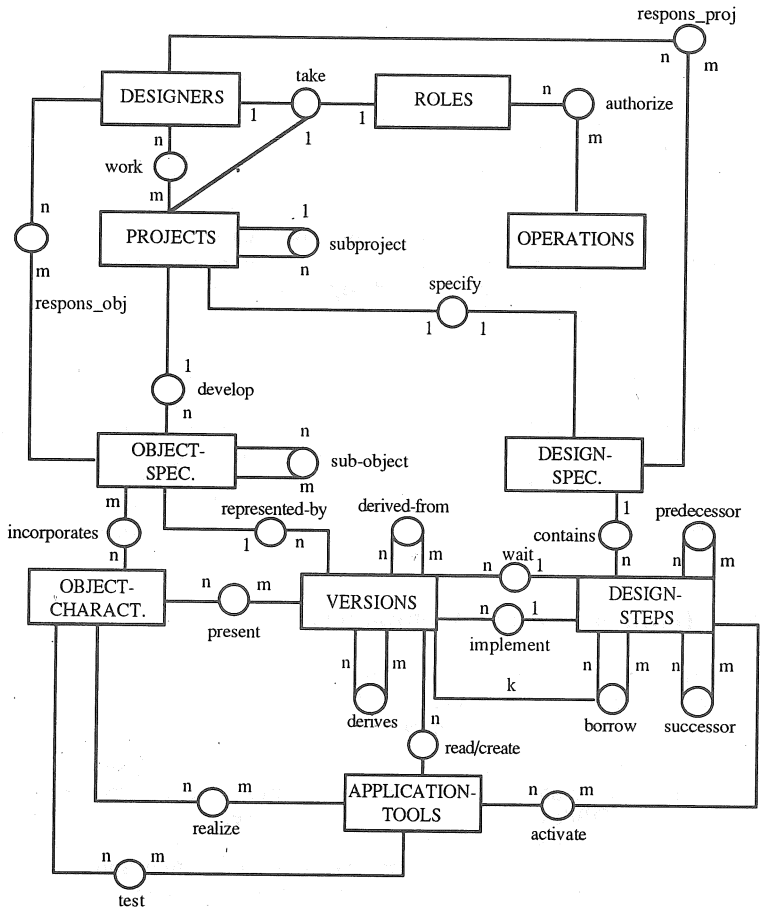

Figure 1: DBdc's conceptual schema

3.2 Giving the Designer a Role

A designer can **work** in one or more projects but in each one of them she/he can **take** only one **role**. The latter **authorizes** the subset of DCM **operations** which can be executed by designers related to it. Examples of operations are the creation of new projects, the insertion/deletion of an object characteristic, and the execution of a design step.

Depending on her/his role in the project, a designer can be responsible for the specification of one or more objects (**respons-obj**) as well as the design specification (i.e. definition of design steps and the relationships where they play a role). In addition, designers responsible for a design specification are the only ones who can start as well as control execution of its design steps.

3.3 Executing Partial Orders of Design Steps

By starting the execution of a design specification, the responsible designer may choose from one of two execution modes: automatic or monitored. The prior implies that DCM does not have to ask for permission as to start and commit the execution of design steps. It keeps

information about step dependencies and starts predicate evaluation to decide which DS (or set of design steps) can be started at each point in time. Monitored execution forces DCM to ask the responsible designer for permission every time a DS can be started. The latter form of execution enables designers to share the responsibility over the design process. Different steps can be executed at different workstation sites. Automatic execution, on the other hand, keeps on starting design steps at the site where the execution of the partial order began. Only in case of object fault in the actual DBpr, DCM halts step execution and asks the responsible designer for a decision.

During the execution of a partial order, DCM updates DBdc when necessary. Thus, if a design step inside which new object versions were generated commits, the design and cooperation manager updates the **version** as well as the **design-steps** entities and starts looking for pendent object-transfer operations which can use the just created versions.

If either an object or a design specification is updated during the execution of related partial orders of design steps, DCM investigates the consequences of such an act. For instance, in case a relationship between an object characteristic (Ci) and an object specification (Sj) is removed from **incorp**orates, DCM searches the tool which should realize Ci in Sj, identifies the versions of Sj which should have that characteristic and the design steps that either created or must create such versions. From this point on, DCM can identify what must be modified in the partial order (i.e. steps which must be aborted or compensated for) and which other partial orders will be affected by such modifications (i.e. the ones that contain steps which borrowed object versions from steps being aborted or compensated for).

3.4 DCM's Implementation Architecture

Two software modules implement STAR's design and cooperation manager: the project manager (PM) and the cooperation manager (CM). PM supports the managerial aspects of a design project while CM supports the cooperation among designers.

The project manager inserts and maintains all information kept in the design and cooperation database but those data related to cooperation activities. It guarantees design process correctness by enforcing DBdc consistency, controlling the execution of partial order of design steps, evaluating start predicates, identifying possible commit points for design specifications, mapping partial orders onto (long) design transactions, submitting application tools for execution as short, ACID transactions, and interacting with STAR's version manager to declare as well as access and delete object versions in the design database hierarchy.

The cooperation manager controls cooperation activities among designers by keeping track of object-transfer operations. It searches for versions being requested by design steps and verifies the possibility of lending them to the steps. CM maintains all information concerning borrowed object versions (i.e. creators, users, involved databases, predecessors, successors, creator step, user steps, etc.). Relying on these data, it can, for instance, identify who must be informed in case of stabilization of new versions of the same object. Furthermore, CM can also inform object creators of most needed object characteristics in order to speed up the cooperation process. Finally, the cooperation manager must make sure that object versions which depend on other versions (e.g. **object-subobject** relationship) are not consolidated before the latter are.

4 Conclusions and Future Work

In this paper, we gave an overview of a design management and cooperation model and presented a mechanism to implement it. The model supposes a distributed computer system with a server-workstation architecture. It also assumes the existence of an underlying transaction manager that supports hierarchies of (long-duration) nested transactions. At least the top-most transactions should be allowed to exchange non-committed results.

The design management and cooperation model supports the specification and evolution of design projects. Object specification is done by means of application-oriented object

characteristics while design process specification is based on partial orders of design steps. The model also supports cooperation among designers who develop either common objects or sub-objects related to one another.

The design and cooperation mechanism which realizes the model relies on a so-called design and cooperation databse to control design process evolution. DBdc stores all relevant information about the design. The mechanism which is being implemented as two related software modules is responsible for DBdc's creation and maintenance. It also controls the correctness of the design process relying on partial orders of design steps and object-transfer operations.

A prototype of the design and cooperation mechanism is being implemented at the CPGCC-UFRGS. The DBdc and some functions of both the project and cooperation managers will be implemented as an application of the KRISYS (Knowledge Representation System). Other modules of the managers will be implemented in LISP and C. We expect to have a running version of the prototype by the end of 1992.

5 References

[1] P. A. Bernstein, V. Hadzilacos, N. Goodman: Concurrency Control and Recovery in Database Systems. Addison-Wesley Series in Computer Science, Addison-Wesley Publishing Co., 1987

[2] K. P. Eswaran et al. The Notions of Consistency and Predicate Locks in a Database System. Communications of the ACM, New York, v. 19, n. 11, p. 624-633, Nov. 1976

[3] T. Härder, A. Reuter: Principles of Transaction-Oriented Database Recovery. ACM Computing Surveys, v. 15, n. 4, p. 287-317, Dec. 1983

[4] C. Iochpe, M. A. C. Livi: Cooperation Support in a CAD Environment. 6th Brazilian Symposium on Database Systems, Manaus, May 1990 (in Portuguese)

[5] W. Käfer: A Framework for Version-Based Cooperation Control. Research Report, Universität Kaiserslautern, 1990.

[6] M. A. C. Livi: Requirements Analysis and the Proposal of a Cooperation Model for the AMPLO Environment. Research Report, CPGCC-UFRGS, 1991 (in Portuguese)

[7] N. M. Mattos: KRISYS - A Multi-Layered Prototype KBMS Supporting Knowledge Independence. Research Report, Universität Kaiserslautern, 1988

[8] M. H. Nodine, A. H. Skarra, S. B. Zdonik: Synchronization and Recovery in Cooperative Transactions. 4th Int. Workshop on Persistent Object Systems Design, Implementation, and Use, 1990

[9] F. de F. Rezende: A Transaction Model Supporting the World Concept of KRISYS. Master Dissertation, CPGCC-UFRGS, 1992 (in Portuguese)

[10] F. R. Wagner et al. STAR: An Environment for the Integration of Digital System Design Tools. 6th Congress of the Brazilian Microelectronics Society, Belo Horizonte, 1991 (in Portuguese)