Controlling Cooperation Through Design-Object Specification
- a Database-oriented Approach -

Christoph Hübel, Wolfgang Käfer, Bernd Sutter
University Kaiserslautern
W 6750 Kaiserslautern, Germany
{huebel,kaefer,sutter}@informatik.uni-kl.de

Abstract
We propose an operational framework for supporting cooperative design processes carried out by a group of designers. Besides the cooperative refinement of the design-object specification, the introduced cooperation protocol allows cooperative design by system-controlled interchange of design object versions. The versions reflect the process of step-wise improving the design object so as to meet finally its specification. We introduce a data integrated design environment based on a nonstandard database system in order to manage the whole cooperation process, i.e., the design objects, the versions, and the design activities.

1. Cooperative design scenario
The design of complex technical objects is commonly performed by cooperating designers. These work together, since they jointly aspire to a shared goal. In order to support this cooperative process in a way controllable by computers, and to make it more visible and reviewable to others, a cooperation model is required, that provides adequate cooperation protocols and primitives.

The complex technical object to be designed, i.e. the design object (DO), is hierarchically divided into subobjects (sub-DO) that have to be refined at different levels of abstraction. These levels are often called domains of DO's representation. As the objects, the entire design process is partitioned into a hierarchy of design activities (DA). Each DA takes over the task of partial refinement and corresponds to a part of the DO in single domain, i.e., the goal of such an activity is specified by a set of properties that have to be satisfied by the DO considered at the end of the DA. DAs are assigned to particular designers who, in turn, are responsible for the ongoing refinement of related DOs. DAs may exploit results of other DAs, thereby becoming dependent from each other. However, they can be performed in parallel if subsequent DAs can start work with uncompleted, unproved and unreliable results delivered by DAs previously started. Obviously, such a situation constitutes the typical one in most cooperative design environments, since it is caused by iteration, step-wise improvement, and feedback loops.

From a more abstract point of view, the interdependencies among such DAs are effected by the overall shared goal to be reached, e.g., sub-DOs are designed to fit together and to be used for the composition of an aggregated DO. Furthermore, these interdependencies occurred due to the parallel execution reflect the conviction that a particular shared goal can be achieved better and in shorter time when DAs are cooperatively performed. The subject of cooperation among DAs and, therefore, among designers is determined by the subject of design at a particular level of the DO's aggregation hierarchy within a single domain. Since this corresponds to the results of a DA, the act of design cooperation can be seen as the coordinated exchange of preliminary results of DAs. Such a result is called version of a corresponding DO (design-object version, DOV). In order to quantify the degree of preliminarity of DOVs and to control their exchange, it becomes necessary to specify the quality of an object in a formal manner. In a design environment, such kind of quality may appropriately expressed by so-called features which describe various characteristics of the DO by value intervals, test programs, etc. The set of needful features is called DA specification. Since a DOV gradually fulfils this set of features, we can distinguish various quality stages. Satisfying the complete DA specification, a DOV is said to be final. We call a DOV private if its features are not yet evaluated. A private DOV is only visible to the creating designer. Otherwise, i.e. a DOV is neither final nor private, it is called tentative what means that it provides just a subset of the DA specification and therefore it is still subject to improvements. Nevertheless, a tentative version is visible (in a controlled manner) to other designers.

The main intention of any mechanism for cooperation control lies in supporting a group of designers to reach their shared goal by guiding them to gradually improve the DOVs that they have to create within related DAs [1]. By the first measure to meet this intention, dependencies among DAs as well as their relationships with the DOs
must be explicitly represented and must be exploited for notification and change propagation purposes. That is, the designer of a DO would like to be informed at least when relevant properties of an used sub-DO change. Fig. 1 shows an example of a DA hierarchy and a DO hierarchy chosen from the area of VLSI cell design. The cell under design, e.g., a VLSI processor, is divided into several sub-DOS, e.g., CPU, memory cache, bus. The DA that aims at the design of the overall processor is partitioned into the DAs overall processor (creation and management of sub-DA, release of final DOV), assembly (for assembling and testing the integration of the other sub-DOS), CPU, memory cache, and bus. As suggested in Fig. 1 the DA overall processor has invoked the other DAs. Obviously, the assembly DA uses preliminary DOVs of the CPU, memory cache, and bus before it can start work. In turn, it affects the specifications of the corresponding DAs, e.g., in order to coordinate the shape size that might be used by these DAs to carry out the cells desired. Though they do not use related DOS, the CPU, memory cache, and bus DA mutually affect their DA specifications so as to harmonize, e.g., time as well as place conditions. That is, by the adaptation of DAs' specifications these DAs affect the ongoing improvement of related DAs so as to reach the specified features attached to their own.

Computer Aided Design (CAD) systems, in particular, frameworks for such systems [2, 3, 4] should support cooperative design as mentioned above. In this paper, we present an operational framework for supporting cooperative design within a data-integrated design environment that is based on the idea of integrating DAs by means of advanced database systems (DBS). Thus, in chapter 2 we point out our view on how DBS functionality might be exploited for carrying out design cooperation, and we illustrate the role of such a cooperation component in a data-oriented CAD-framework architecture. Starting on top of a nonstandard DBS, we obtain the capability to qualify the subject of cooperation, and to exploit the evaluated quality for handling dependencies among cooperating DAs. Therefore, we introduce information structures that are required for representing concrete cooperation scenarios, and we refine several operations that serve as cooperation primitives. The dynamic characteristics of these primitives are illustrated using a state/transition graph and are further clarified by exemplifying a simple cooperative design process.

2. Incorporating a cooperation component into a database-oriented design environment

Our considerations start from the idea to exploit the advantages of DBS in the development of design environments. DBS are able to handle bulk data efficiently and allows for multi-user access. Especially, advanced DBS, also called non-standard database management systems (NDBS), show some degree of object orientation and, therefore, they might be tailored for maintenance of complex design objects [5]. In addition, DB-research efforts aim to enhance such NDBS by means for several basic services (e.g., cooperation control, version management, or design management) as required in design applications.

Most investigations concerning the integration of cooperation mechanisms into database systems [6, 7] start from the well known transaction concept [8] in which each transaction (as the unit of database processing) obeys the ACID-principle. ACID stands for atomicity, consistency, isolation, and durability of transactions. Especially, atomicity and isolation guarantee the processing integrity in a multi-user environment, because they enforce serializability of concurrent transactions. This task is achieved by system encoded mechanisms for synchronization (locking). In case of processing conflicts, i.e., transactions need access to the same data, the system automatically reacts by blocking, or backing out. Since isolation contradicts the cooperation requirements (exchange of unreliable data among uncommitted transactions), several approaches allow the invalidation of the ACID characteristics [9, 10] so as to get the ability of cooperation but giving up the clear processing semantics defined by the ACID principle.

Our own approach differs from those in that we do not concentrate on isolation but start from the consistency point of view. Knowing that transactions cover DAs with a well specified goal, we explicitly define the meaning of processing integrity for a couple of DAs with respect to the shared goal they have to reach [11, 12]. That is, processing integrity is defined to be satisfied when the versions, generated by the DAs involved, show at least no contradiction to the shared goal. In order to force this, we exploit protocol
mechanisms that allow user-defined reactions when the system recognizes that any cooperating DA generates an improved version on the way to its final result.

Before presenting the cooperation model in more detail, we sketch our architecture concept for a database-oriented design environment in order to illustrate the embedding of the cooperation component. Being influenced by a workstation/server distribution, the working place of a designer is associated with a workstation-site, whereas more general services of the design environment are located at the central server site [13]. The system architecture is roughly divided into three layers (cf Fig. 2).

- The NDBS kernel layer provides at its interface an application independent data model that shows at least a high degree of structural object-orientation and an expressive (mostly descriptive) query language which allows the selection and the manipulation of complex objects.

- The interface of the design application oriented layer is tailored to the special needs of design applications. Therefore, this system layer is composed of a number of basic services providing appropriate functionality to design applications. However, in this paper we only describe those services which are necessary for the comprehension of our approach to control cooperation.

The design manager offers general operations for design-process handling like generation/termination of DAs, specification of features, and definition of reactions. Maintenance of design cooperation is subject of the cooperation manager. It provides a couple of primitives that should allow designers to cooperate with each other under system control. Thus, it is subject of the cooperation manager to guarantee concurrency control among the designers with respect to the underlying cooperation model. Since we want to integrate the different managers in a homogeneous manner, all information structures concerning the DAs, the current state of the cooperation process, the DOS, DOVs etc. have to be represented in an integrated manner.

- The design environment layer is the top most layer in our system architecture and offers the DOS and the operations the designers need to work with. In particular, this interface allows the designer to deal with DOS, i.e. to manipulate them by using design tools, and to activate the operations provided by the underlying components so as to cooperate with each other.

The actual system environment influencing our investigation about cooperation control is given by the NDBS prototype PRIMA [5] and the experimental VLSI-design system PLAYOUT [14].

3. Information structures and operations concerning the cooperation model

This chapter aims to work out the information structures and operations of the mentioned basic components as far as necessary to explain the cooperation model. Fig. 3 illustrates the information structures in an entity/relationship diagram-like representation form. We use a special symbol for the illustration of versioned entity types, i.e. of entities which exist in versions. However, the data model of the underlying NDBS system (which will not be introduced in this paper) copes with versions in a homogeneous manner. The semantics of the schema will be explained via the operations working on it.

Operations of the design manager

- generating / terminating a DA

Firstly, a new instance of the entity type design_activity is created and it may be enhanced by an informal description about what this activity is for. However, the design specification is given by the linkage of features via design_specification to the DA at hand. Furthermore, the creator may associate a number of design tools, which have to be employed during the execution of the DA (i.e., no other design tools are allowed). Lastly, we have to charge a designer with the DA.

As mentioned in the first chapter, complex DAs are decomposed into a number of sub-DAs. This decomposition is represented by the relationship design_hierarchy. The creator of DAs determines the group of cooperating designers (cooperate_with) that work together, and therefore, are able to affect the goal and the way in which it is achieved.

---

1. References in the text to entity or relationship types depicted in Fig. 3 are written in italic style.
A DA can only be terminated, if all sub-DAs are terminated and the creator of the DA at hand accepts the termination. All tentative versions of the DA will be deleted. Furthermore, a DA may be deleted by its creator. generating a feature

The features a DO has to accomplish, we describe in a formal manner, e.g., the functional design objectives and environmental design restrictions, or organizational aspects concerning the design state such as "test program A" or "release procedure X", etc. Thus, the features linked via design specification with DAs are called the specification of the DA.

However, these features may be used to describe the quality (i.e. the completeness or correctness) of the DOVs w.r.t. the design specification. Therefore, we introduce the relationship provides linking those features of the design specification to a DOV which are valid for this DOV. For example, we link the feature "test program A" (via provides) to those DOVs which have passed test program A successfully. In addition, the expressiveness of features may be enhanced by introducing feature values. For example, a particular version of a cell has a particular size, e.g. 85 \( \mu \text{m}^2 \). Therefore, in addition to the linkage of the feature "size within 80 to 95 \( \mu \text{m}^2 \) via provides to the DOV, we relate the precise value of the feature via has_value to the DOV and the feature.

Summarizing, a feature is characterized by four tokens: evaluation procedure, predicate, value, and validity. The evaluation procedure may be a simple attribute selection on the related DOV or a complex procedure computing the actual feature value from the related DOV (e.g., the computation of the cell size). The predicate puts constraints on the result of the evaluation procedure, i.e., it restricts the valid range of the feature value. The feature is called valid if its value holds for the given constraint.

- defining an action

The entity type action defines the reactions being necessary in a DA according to feature modifications (i.e., feature value modifications or changes in the provides relationship) of used DOVs. The execution of a design tool or just the notification of the responsible designer about the changes are examples of actions. The activates relationship connects the actions to those features they depend on, whereas the defined_for relationship connects them to a particular DA. This takes into consideration that different DAs may apply different actions for the same changes of features.

Operations of the cooperation manager

- propagating a version

During a DA a designer may produce an arbitrary number of private versions. In order to make a version visible to other designers, the designer has to propagate it, i.e., the version becomes tentative. During the propagation, the cooperation component performs two main steps. It has to evaluate the valid features and their values for the given DOV. The feature values have to be stored and the provides and has_value relationships have to be installed.

The actions associated with the features are invoked. Furthermore, the cooperation component determines those tentative DOVs with minor quality of the same DO in order to replace them by the new one. Some of these tentative versions may be used by cooperating designers. Since the cooperating designers share an overall (shared) goal, they will be interested in the progress of the propagating DA and therefore, they are informed and have to use the new DOV. However, the users are given some time to perform necessary adaptations. When it is no longer used, the tentative DOV will become a private DOV again. Thus, the creating designer may start (an alternative) design upon this private version, but it is no longer visible to other designers.

- requiring/using a version

During the execution of a DA, a designer needs information about the design of cooperating designers, i.e., she/he wants to get tentative (or perhaps final) DOVs of the related DAs. Therefore, the designer specifies the required features of a DO. If there exists a DOV satisfying these features, the designer may use or incorporate it into the own DO. Thus, the relationship uses allows for the correlation among a DA and the incorporated DOVs. Otherwise, i.e. no existing DOV fits, the designer will be informed when an appropriate DOV is propagated.

- propose - agree - disagree (about features)

These are the basic operations to allow negotiation about (the features of) the design specification among two
or more designers. One of them may propose a new or refined feature (of course, under consideration of the original design specification, i.e., only 'new' features and feature refinements are allowed), and the other designers may agree or disagree with this proposal.

- impossible - modify (features)

During the design, the designer may recognize that one or more features of the design specification are impossible to carry out, or, on the other hand, have to be modified due to changes in the superior DA. Consequently, the cooperation component offers the operations impossible and modify to cope with these cooperation problems.

4. The cooperation process

In this chapter, we sketch out the principle course of cooperation processes in terms of our cooperation model. The basic means of the representation is the state/transition graph depicted in Fig. 4. The nodes represent the states of DAs whereas the edges represent valid transitions. The transitions are either initiated by performing operations within the considered DA or by effects of operations initiated by other DAs.

After the generation of the DA, the responsible designer is informed about his new task and therefore, she/he begins studying the DA's specification. Then there are two ways to cooperate with other designers.

- Cooperation via design specification

In case the design specification is somehow imprecise or inaccurate, the designer proposes a refined design specification to cooperating designers starting a negotiation process and entering the state NEGOTIATE. During this negotiation process, the designer may propose further refinements of the design specification or she/he may agree or disagree to proposals of cooperating designers. In the former case, the designer can start work with the refined design specification (entering the state ACTIVE). In the latter, the negotiation process goes on.

- Cooperation via versions

In case the design specification is fairly clear, the designer starts work (state ACTIVE), i.e., she/he generates some private DOVs thereby requiring and incorporating DOVs of cooperating DAs. When the design has gained a certain stage, the designer propagates an initial DOV. Subsequently, she/he propagates refined DOVs, thereby turning all previously propagated DOVs with minor quality into state private. Designers that use older DOVs are informed about the new DOV and its features.

The two ways of cooperation mentioned above can be related to the well-known general phases of design processes:

- Design: The design task is accomplished in state ACTIVE by performing design tools, generating new DAs along with the related features and actions (if necessary), and by propagating/requiring DOVs (cooperation via versions).
- Reviewing: During the design, the designer may want to review her/his own design specification or the design specification of cooperating DAs (cooperation via design specifications). Perhaps, this is due to the purpose of harmonizing the design specifications (propose/modify) or due to the designer determines that some part of the design specification is impossible to carry out (in all cases entering the state NEGOTIATE).

- Completion: After propagating at least one version which satisfies the design specification, the designer want to complete the DA by entering state TERMINATE (terminate). In order to do so, all pending propagate actions and negotiations have to be finished first. Furthermore, the responsible designer of the DA, during which the DA at hand was generated, has to accept. Otherwise, this designer can modify the design specification of the actual DA changing its state back to ACTIVE.

**Example**

Looking on our example depicted in Fig. 1, we can imagine the following cooperative design process. Suppose, the designer assigned to the memory cache DA can start work some time earlier than his colleagues. Thus, she/he generates and propagates a final DOV and terminates. Since, at this time nobody can be really sure, whether the DOV will fit with the other components, the termination is suspended, i.e. the responsible designer of the overall processor DA does not agree (at this time).

Let's suppose, the cpu DA and the bus DA start work. Since, their DA specifications are rather rough, they firstly try to refine them. Thus, the cpu designer proposes a clock cycle time of 24 MHZ and a transmission rate as specified by the memory cache DA. In order to meet this transmission rate the bus designers proposes 27 MHZ and about
10% more size than expected (i.e., the task cannot be done with the actual size restriction). The cpu designer agrees, but the demand for the size violates the restriction given by the overall cpu designer. Thus, this demand is passed to her/him. She/he recognizes that the memory cache has surplusous space. Therefore, the designer modifies both size specifications of the bus DA and the memory cache DA (shifting the memory cache DA back to state ACTIVE).

After a while, the cpu DA and the bus DA propagate their first DOVs allowing the assembly DA to start work. The assembly designer recognizes that she/he needs more space in order to carry out her/his task. Fortunately, the cpu designer can reduce the space requirements of the cpu. Thus, they propose a refined specification of their DAs again involving the overall processor designer. This time, the designer just agrees.

After some time, one DA after another propagates a final DOV and enters the state TERMINATE. Granting (agree) the termination of the cpu DA, the bus DA, the memory cache DA, and the assembly DA the overall processor designer finishes these DAs and finally the whole design.

5. Conclusions and outlook

Maintaining cooperative design requires mechanisms for explicitly describing the processing units as well as the subject of cooperation. Both issues are addressed by our framework for cooperation support: the design process is carried out by a hierarchy of DAs and cooperation among them is subject to qualified versions of preliminary design objects. Managing DO versions and DA hierarchies in an NDBS-oriented environment, we have introduced a set of operations which enriches the well-known transaction concept, and which gives it more adequacy for cooperative NDBS applications. In comparison with the traditional transactions the DAs, as the units of cooperative processing, show three major characteristics that make them suitable for design process support. First of all, in opposite to the global view of consistency as it is addressed in the conventional transaction concept, i.e., where consistency is defined with respect to the overall database, we have allowed defining consistency in a design-activity specific way via goal specification. Additionally, this kind of consistency is not statically defined but it is the matter of negotiation. Secondly, in contrast to the lock-mechanisms that are commonly used for isolation purposes, i.e., for reducing visibility of data, we utilize the quality of DOV to increase visibility in an explicit as well as a controllable manner. Thirdly, traditional transactions are automatically blocked or backed out when a conflict occurs. In the cooperation scenario, a conflict means that some desired features of a DA cannot be carried out by the responsible designer. In this case, our cooperation model allows for constructive (failure) reactions via negotiation (disagree/impossible) among all DAs involved.

So far, we have worked out the operational framework for supporting cooperative design. Our current investigations focus, firstly, on carrying out the cooperation primitives as explained above based on our NDBS prototype, and, secondly, on evaluating their feasibility within our experimental VLSI design-environment.

6. Literature