On Structuring Primitives and Communication Primitives for Design Environments

Norbert Ritter, Bernhard Mitschang, Michael Gesmann, Andreas Grasnickel, Theo Härder, Clarence Huff, Christoph Hübel, Wolfgang Käfer, Harald Schöning, Bernd Sutter

Department of Computer Science
University Kaiserslautern
P.O.Box 3049
6750 Kaiserslautern, Germany

e-mail:{ritter, mitsch, ..., sutter}@informatik.uni-kl.de

1. Introduction & Motivation

The evolution of CAD systems [HNST90] can be described in several stages which reflect an increasing effort for system integration. It starts from a file-and-translator approach evolving to a data-integrated tool environment, and finally reaching the stage of a data-integrated design environment for CAD (sometimes also called CAD Framework). In the following we will detail some aspects of these stages.

The first step toward CAD Frameworks is characterized by a set of design tools which either communicate through **data/file translation** or use a **common intermediate format** [EDIF, STEP] to simplify data exchange. Each tool has to handle everything on its own, i.e. data management and data access, version control, communication, and all cases of failure handling. For the CAD system user, i.e. the designer, there is no system support w.r.t. design management, design methodology, etc.; he has to keep all these design-specific issues and strategies in his mind.

While the CAD tools themselves are essential to the design process, the management of design data and its presentation to the tools in a useful and efficient form has become a major issue. With the success of database management systems (DBMS) in commercial and business environments one tried to employ these systems also for design data management. Quite soon, however, it became clear that data definition and data manipulation capabilities of conventional DBMSs are not powerful enough to satisfy the requirements set by engineering design data [Si80, Hä89]. This deficiency triggered considerable research leading to the development of advanced DBMSs [CACM91], which cover the modeling and management issues either by extensions to the relational model or by some kind of object orientation.

Employing a DBMS leads to a *data-integrated tool environment*, where **the responsibility for data or even version management is taken away from the tools and shifted to the integrated data repository.** Usually the design/version data is extracted from the database (Checkout operation) and loaded into main memory close to the tool application. After the tool completes its work on this data set, the changes are propagated back to the database (Checkin operation). Thus, tool data management is considerably simplified, while the designer's task stays unchanged: still, it is on him to keep track of all design-specific issues and to control the progress of design.

It is the next generation of CAD environments, the so-called *data-integrated design environments*, which concentrate on simplifying also the work of the designer, i.e. providing services that offer **support for design management**, design methodology, etc. Hence, work can be shifted from the designer to the system. For example, it is now possible to specify a 'recipe' that models a certain design methodology. In order to do this, adequate structuring primitives and communication primitives that help in organizing communica-

tion, cooperation, and managing the design process have to be developed and provided by the design environment. Of course, there are other important services, e.g. tool integration and user modeling that can simplify the CAD system administrators's task.

The major issue of our work is the conceptualization of the different kinds of activities having to be carried out during the overall design process w.r.t. their different requirements. We have to distinguish between at least two kinds of activities reflecting specific levels of design decisions. At a higher level the administrative part of design work has to be supported, focusing on the description and placing of design orders and the synthesis of partial results. A lower level has to support the organization and execution of complex objects processing, i.e. tool executions. In the course of this paper we will concretize adequate concepts for modeling those activities by examining their properties such as their notion of consistency, their interactions, their internal structures as well as the failure handling mechanisms including conflict resolution.

Sections 2 and 3 introduce structuring primitives and communication primitives that abstract from concrete design tools, thus providing generic facilities. We will analyze their properties and usages as a basis for modeling and managing the design process. In Section 4 some aspects of transaction processing and failure handling in our data-integrated design environment will be sketched. The last section will give a conclusion and an outlook to further work.

2. Structuring Primitives

Obviously, the primitives to be supported by a data-integrated design environment are derived from the comprehension of the design process being characterized by goal orientation, hierarchical refinement, stepwise improvement as well as team orientation and cooperation. The concepts discussed in this section reflect the first three of these characteristics while the last two are the intension for providing communication primitives which will be discussed in the next section.

Design Activities

The overall goal of the design process is to come up with a ***design object*** (DO) meeting all requirements specified. In general, DOs are composed of several subordinate design objects, thus spanning a ***DO hierarchy***, which provides a natural basis for further structuring of the design process. According to the decomposition of the DOs, the design process is partitioned into a hierarchy of ***design activities*** (DA). The task of a DA is the derivation of a DO obeying the ***design specification*** given by a set of required ***features*** [Kä91].

Fig. 1 depicts a simplified DA hierarchy with the responsibilities (shown as arrows) of the included DAs for parts of the related DO hierarchy. The operation *Init_Design* allows for the initiation of a design process by the creation of the top-level DA (DA1 in our example). It requires a schematic description of the according DO (here DO1) and the design specification describing the goal of the overall design process initiated. In the simplest case a feature in the design specification can force the value of an elementary object's property to belong to a certain range of the underlying domain. A more complicated feature can express the need that the object under design has to pass a test tool successfully. In addition to DO description and design specification a designer (or a group of designers) has to be assigned to the DA, who has to control respectively to carry out the work. Due to simplicity of our explanations in this paper we will consider the DAs as the active units of the design process abstracting from designers work. During its efforts to reach the design goal a DA may delegate parts of its own design order. This has to be done by creating a sub-DA. The operation *Create_Sub_DA* requires a schematic description of the DO and a design specification as input parameters, too. The sub-DAs' specification constitutes a subgoal of the super-DA's design goal and the DO of the sub-DA has to be a part of the super-DA's DO. The execution of the *Create_Sub_DA* operation implicitly establishes an occurrence of the relationship type which we call delegation. The delegation is an

multi-level relationship type spanning the DA hierarchy. In our example (Fig. 1) DA1 has created the sub-DAs DA2 and DA3 with the order to design the parts DO2 and DO3 of the aggregate DO1. In the same way, DA2 has delegated parts of its work. We see, that the super-DAs in our example splitted their design order completely and delegated the sub orders to their sub-DAs. Their own work remains the control of the design work in the subordinate part of the DA hierarchy and the integration of the results delivered by the sub-DAs. The facilities for controlling their sub-DAs' work will be explained in section 3.

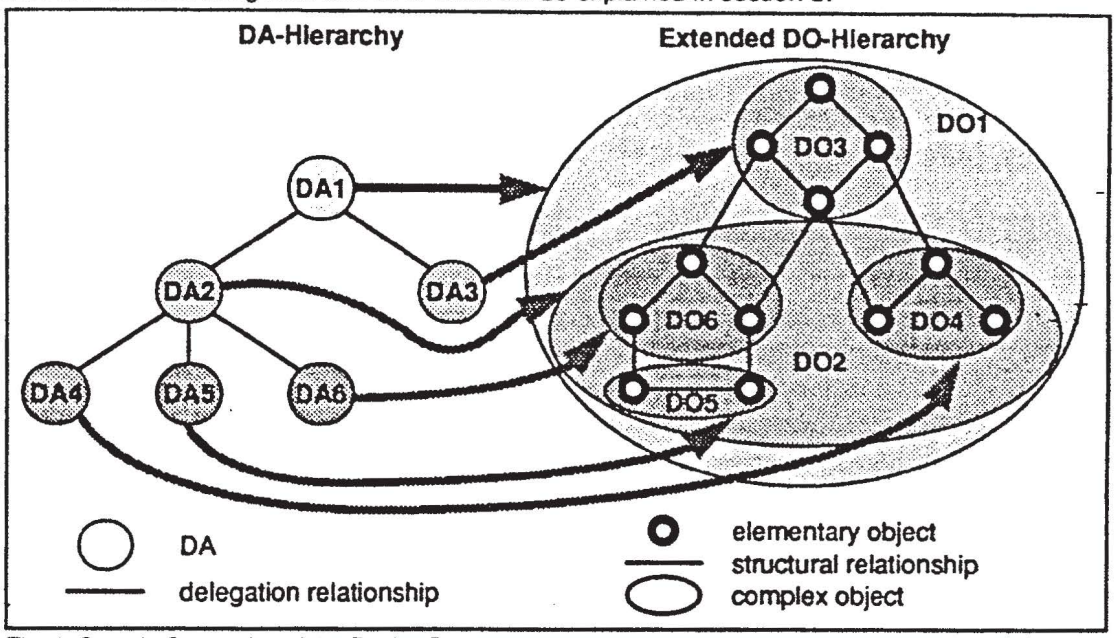

Fig. 1: Sample Structuring of the Design Process

Design Operations

Mostly, design tools are used to accomplish the design task attached to a DA. In order to abstract from a specific design tool, we call the action performed ***design operation*** (DOP). DOPs are used to achieve step-wise improvement of (preliminary) results, which are represented as ***design object versions*** (DOV). A DOP reads several DOVs and writes a newly created one. In this way a derivation graph arises organizing the DOVs created within the scope of the DA [KS92]. For example, DA4 in Fig. 1 manages a derivation graph organizing the versions of DO4. Based on the number of fulfilled features (w.r.t. the design specification of the DA) a DOV can be assigned a certain level of design 'quality'. We distinguish them as non-qualified, partly-qualified and fully-qualified DOVs. The quality state can be determined by the *Evaluate* operation. This operation also allows for the recognition of the final stage of the DA which is reached with the derivation of at least one fully-qualified DOV. DOVs are managed by our integrated data repository using checkin/checkout operations. The DOVs organized in the derivation graph of a super-DA are a synthesis of DOVs delivered by sub-DAs and the results of not delegated parts of the design work of the super-DA. Note, that further work on a DOV constituting a synthesis of fully-qualified sub-DA DOVs may be necessary, because in most cases the assembly of the fully-qualified DOVs of the sub-DAs does not automatically constitute a fully qualified DOV of the super-DA.

Fig. 2 depicts a sample series of DOP executions within a DA. This DA execution plan may include sequential as well as parallel tracks as shown by the arrangement of DOP executions in the picture. The plan is determined either a priori by appropriate definition facilities or in an ad-hoc manner by a designer, who interacts with the DA. Fig. 2 shows a parallel execution of DOP1 and DOP2, followed by DOP3 execution, and finished by the determination (*Evaluate* operation) of the quality state of the DOV derived by DOP3.

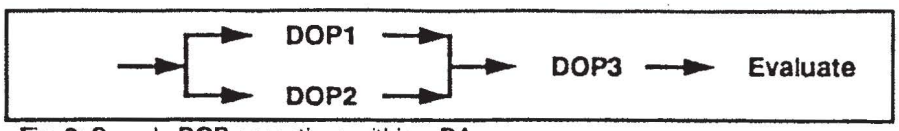

Fig. 2: Sample DOP executions within a DA

The mentioned concepts of DAs, delegation relationships and DOPs allow for structuring the design process. How to exploit the inherent integrity constraints of special relationship types between DAs to enforce a controlled communication and cooperation is the topic of the following section.

3. Communication Primitives

From an abstract point of view, design proceeds in a cooperative manner reflecting the conviction that a particular goal can be achieved better and in shorter time when the DAs of a DA hierarchy work together (may be in parallel). Communication between DAs might be even necessary, because the initial specifications given to DAs are in most cases neither fixed nor complete enough such that their results (i.e. the DOVs created) automatically fit together constituting a solution for the higher goal given by the design specification of the super-DA. Communication and cooperation among DAs result in interdependencies that can be modeled by specific relationship types showing up as additional edges in the DA hierarchy graph (cf. Fig. 3). In the following, we introduce these relationship types sketching their semantics and inherent integrity constraints by which the observance of special communication protocols can be forced.

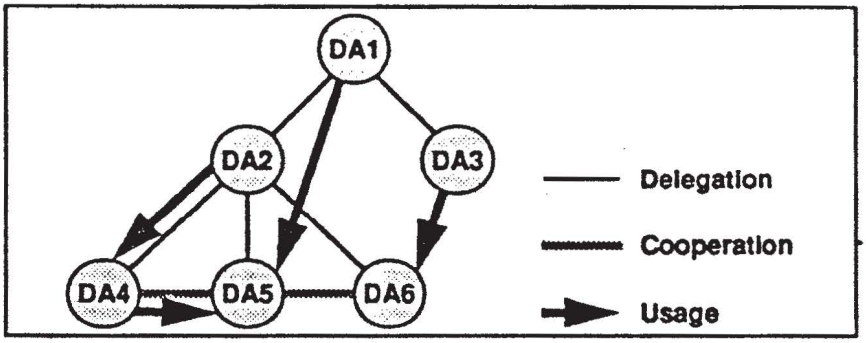

Fig. 3: Communication and Cooperation Relationships attached to a DA Hierarchy

The delegation relationship is fundamental for modeling cooperative design processes as shown in the previous section. We take up this relationship type again because it constitutes not only a way of ordering but also a way of communication between a super-DA and a created sub-DA. Both are DAs with the characteristics described above. Remember, a DA may create an arbitrary number of sub-DAs (operation *Create_Sub_DA*), as long as it is appropriate to reach its own design goal. When creating a sub-DA, the super-DA has to formulate the goal specification for the sub-DA, whose successful termination is the precondition for the termination of the super-DA. The super-DA keeps all the rights of the creator, i.e. it is able to terminate a sub-DA or to modify its specification (operations *Terminate_Sub_DA* and *Modify_Sub_DA_Specification*). Such reformulations are typical in design applications. The sub-DA, on the other side, is only allowed to concretize its own specification by addition of new features or by further restriction of existing features. As soon as a sub-DA completes its work by reaching one or more fully-qualified DOVs, it has to send a message to the super-DA showing up a final state w.r.t the actual specification (operation *Sub_DA_Ready_To_Commit*). The sub-DA may not terminate without the agreement of the super-DA for the following reasons. It may be possible that the super-DA wants to modify the sub-DA's specification in such a way that it would be appropriate for the sub-DA to keep the prevailing results (derivation graph) as a basis for deriving new DOVs on the way to reach the new goal. If the modification of the sub-DA's spec-

lification is not the intention of the super-DA the sub-DA has to be terminated. In this case, the fully-qualified DOVs devolve to the scope of the super-DA. A further operation is *Sub_DA_Impossible_Specification* which informs a super-DA that the executing sub-DA will not be able to fulfill the requirements of its specification and forces a reaction of the super-DA, e.g. termination of the sub-DA or modification of its design specification.

Modifications of a DA specification can also be the result of (cooperative) negotiations between DAs. This leads us to the second relevant relationship type which we call ***cooperation***. We only allow cooperative relations between the sub-DAs of the same super-DA, because (see above) the sub-DAs contribute to the accomplishment of only their super-DA's design goal. The operation *Create_Cooperation_Relationship* connects two DAs via cooperation relationship. Using this operation a DA may establish communication with a DA of the same super-DA. The subject of this cooperation are the sub-DAs' specifications which may be changed due to negotiation. During this negotiation process one side may propose further refinements of the design specification and the other side may agree to or disagree with these proposals (operations *Propose*, *Agree/Disagree*). Recall, that cooperation is sometimes necessary for sub-DAs (of the same super-DA), because in many cases their initial specifications are not constrained in such a way that their results constitute a solution for the super-DA. If two cooperating sub-DAs are not able to reach an agreement, the super-DA has to be informed (operation *Sub_DAs_Specification_Conflict*) and has to resolve this conflict. A detailed discussion of this cooperation model is described in [HKS92].

Besides the cooperation via design specification, the coordinated exchange of preliminary results of DAs is necessary. We model this data exchange by the relationship type ***usage***. A requesting DA may ask a supporting DA which must not be a predecessor in the DA hierarchy for a DOV with a certain feature set satisfied (operation *Require*). This set of features defines the quality needed for the DOV in order to fit into the design of the requesting DA. Precondition for requiring a DOV is that the requesting DA knows about the design specification of the supporting DA and the possibility that the requested information may help the requesting DA to reach its goal. From the view of the supporting DA the delivered DOV must not be a fully-qualified one. DAs which are not connected by a usage relationship may not exchange data. A DOV becomes only visible along usage relationships if it was propagated by the according DA (operation *Propagate*). All propagated DOVs have a certain quality state determined by the operation *Evaluate*. This allows a DA control over which of its DOVs become visible to other DAs.

The usage relationship type leads us to the task of the **cooperation manager** as an important component in the architecture of a design environment, which is responsible for the correct realization of the design methodology. It is on it to manage usage relationships and to inform requesting DAs whenever DOVs with the required quality state or even a higher quality state are propagated by supporting DAs. Furthermore, it has to manage the DA hierarchy and to enforce the communication protocols and the integrity constraints according to the relationships set.

4. Failure Handling

It should be noted here, that our approach in contrast to other investigations concerning communication and cooperation primitives, does not agree with the original concept of transactions. The original concept implies that transactions are atomic, can be processed quickly and alter a small amount of data (ACID-paradigm, Atomicity, Consistency, Isolation, Durability). At the DA-hierarchy level, the ACID concept is not adequate in the design environment. Atomicity and Isolation are in some sense contrary to the requirements of design applications, e.g. long duration of processing and teamwork (i.e. communication and cooperation). Design Activities may be recovered using the powerful operations of the version manager [KS92]. This allows users the capability to have some control in the recovery process in the case of a system crash. Not only traditional

failure classes but also application dependent errors (i.e. a DOP execution did not lead to the expected quality state) may be adjusted by choosing arbitrary nodes (DOVs) in the derivation graph for starting point of new derivations. DOPs are able to be isolated and have three states: not started, active and complete. The internal structure of DOPs are saved using checkpoints which safeguards against a system crash.

5. Conclusion and Outlook

By means of the concepts mentioned in this paper, we are able to express certain design methodologies. While *DA*s, the *delegation* relationships and *DOP*s mainly serve for the structuring of the design process in the form of a task hierarchy, the relationship types *cooperation* and *usage* define ways for controlled communication between designers (i.e. among DAs). Though DAs may become dependent from each other, they can be performed in parallel. Within a DA, there are facilities available to explicitly control the organization and processing of DOPs, i.e. tool executions.

Now we are able to summarize the characteristics of the different design activities as mentioned in the motivation of this paper. DOPs are atomic, proceed isolated and the DOVs created are stored persistently in the integrated data repository. DOPs have an internal structure given by adequate save/restore mechanisms. In contrast to the DOPs DAs are neither atomic nor isolated, because a controlled cooperation and communication is supported. A DA may be internally structured by a DOP execution plan. The DOVs created in the scope of a DA are connected via derivation relationships documenting the flow of design. Furthermore, application specific quality states can be assigned to DOVs. This is the basis for adequate data exchange between DAs. The explicit way of handling design specifications constitutes a new notion of consistency in the area of database systems for advanced applications.

Due to space limitations we could not give a detailed description of the architecture of our data-integrated design environment and a validation of the described concepts by showing application examples. Up to now, we have investigated the areas of software engineering and vlsi design. The topics of our future work are the facilities for definition and evaluation of design specifications and the definition of DOP execution plans.

6. Literature

CACM91 Cattell, R. (ed.): Next Generation Database Systems, in: Special Section of Communications of the ACM, Vol. 34, No.10, 1991.

EDIF EDIF, Electronic Design Interchange Format Version 200, Electronics Industries Association, Washington, August 1987.

EI92 Elmagarmid, A. (ed.): Transaction Models for Advanced Database Applications, Morgan Kaufmann, San Mateo, Calif., 1992.

Hä89 Härder, T.: Non-Standard DBMS for Support of Emerging Applications - Requirement Analysis and Architectural Concepts, (Eds: Shriver, B.D.), Proc. of the 22nd Hawaii Int. Conf. on System Sciences (HICSS-22), Kailua-Kona, Hawaii, Volume II, Jan. 1989, pp. 549-548.

HKS92 Hübel, C., Käfer, W., Sutter, B.: Controlling Cooperation Through Design-Object Specification - a Database-oriented Approach, in: Proc. of the European Design Automation Conference, Brussel, Belgium, March 1992.

HNST90 Harrison, D., Newton, R., Spickelmier, R., Barnes, T.: Electronic CAD Framework, in: Proceedings of the IEEE, Vol. 78, No. 2, 1990, pp. 393-417.

Kä91 Käfer, W.: A Framework for Version-based Cooperation Control, Proc. of the 2nd Symposium on Database Systems for Advanced Applications (DASFAA), Tokyo, Japan, April 1991.

KS92 Käfer, W., Schöning, H.: Mapping a Version Model to a Complex Object Data Model, Proc. of the 8th Int. Conf. on Data Engineering, Tempe, Arizona, 1992.

Si80 Sidle, T.W.: Weakness of Commercial Database Management Systems in Engineering Applications, Proc. of the 17th Design Automation Conf., Minneapolis, 1980, pp. 57-61.

STEP STEP Preliminary Design, ISO TC184/SC4/WG1 N208, Mai 1988.