Design Management within a Design Environment

Petra Drescher, Julia Miller, Gerhard Schulz

Cadlab
Cooperation Nixdorf Computer AG and University GH Paderborn
Bahnhofstr. 32
D-4790 Paderborn
Federal Republic of Germany

ABSTRACT

Management of the diverse design tools and design data used during the design process necessitates the use of a design environment with integrated design management capabilities. In this paper, we discuss design management requirements and present an approach for their realization within an integrated design environment. Further, we show the Cadlab Workstation Design Environment as validation of the realization approach.

INTRODUCTION

Historically, CAD tools have been used to automate the more difficult or time consuming parts of the design process. In recent years, a greater number of CAD tools of ever increasing sophistication have become available, making automation of almost every phase of the design process possible. Advances in design automation have made the realization of extremely complex designs feasible. They have also created a management nightmare. Keeping track of the designers working on a given project, the design tools and libraries approved for use within the project, and the vast amount of data generated during the design process has become virtually impossible without the aid of automated management techniques.

One recent approach is to build commands directly on the operating system which manage some specific part of the design process /MSS87/. This approach is helpful but it cannot be seen as a long term solution. The long term solution is an integrated CAD system with built in design management capabilities.

The goal of this paper is to show the feasibility and usefulness of a design management environment with respect to the design process and tool integration. We outline design management requirements, define the general model for the realization of an integrated design, and show how this model is used to realize the design management requirements. This work is based on the practical experience of realizing the Cadlab Workstation (CWS) Design Environment /GKK87/. In conclusion, we will discuss the current status of this work and our future plans.

DESIGN MANAGEMENT WITHIN AN ENVIRONMENT

A design environment consists of a single design management sub-environment and a set of application sub-environments. In many ways, the design manager is just one of the component sub-environments (or design tools) which make up the design environment. As such, it imposes the same requirements on the design environment as any other integrated design tool /MGS89/. It assumes that the design environment provides for:

- Tool Integration - methodology which supports integration of design tools at different integration levels.
- User Interaction - single easily understood user interaction methodology for all integrated tools.
- Environment Extensibility - methodology which guarantees easy extension of the environment capabilities.

On the other hand, the design environment contains a single definition of design management concepts with which all other design tools must be integrated. As such, design management itself is a further requirement on the environment. The design environment must provide for:

- Design Management - A uniform management facility for managing design data and design tools in a multiuser, heterogeneous design environment.

In the remainder of this section, we will attempt to briefly discuss the specific design management requirements for a first generation design environment. We believe that capabilities for design environment modeling, change control, and design management query are required.

Design Environment Modeling

Design environment modeling provides a means for emulating an organization's design environment. It must provide for working environments, design elements, client roles, and design teams.

Working Environments denote semantic partitioning of information within the design environment. We suggest at least 3 types of working environments:
A design element is a product of engineering labor such as chip, cell, program, subroutine, or fabrication process. Each of these products can contain other designs which are themselves products of engineering labor. Furthermore, design elements are implicitly managed within the environment unless the client indicates that they should be explicitly managed. This concept of design elements can be modeled within the environment with less the client indicates that they should be explicitly managed.

A client is anyone or anything which performs an action in the environment and which is subject of access control (e.g., users, tools). A client working within the environment takes on certain roles. Design environment modeling must allow the definition of multiple client roles. The semantic of a client role is expressed by the design tasks that clients who take on this role may perform on certain objects.

Design environment modeling must support the grouping of clients working on a project or sub-project (n project leaders and m designers).

Modeling of working environments, design elements, client roles and design teams within the environment will, of course, vary from organization to organization and also within any given organization over time. The environment must allow modification of existing definitions and introduction of additional definitions to support tailoring of the environment to the specific needs of the individual organizations using it.

In describing the design environment modeling concepts it is not sufficient to describe the environment partitions, design elements, client roles, and design teams. The design actions supported for defining and manipulating the design environment model of a particular organization must also be defined. There are three classes of design actions required for design environment modeling:

- access control of design elements and operations which make up the working environment.
- creation/deletion of clients, working environments, design elements, and design teams
- configuration of working environments

Change Control

Efficient design management requires change control to record all changes made to design objects and to relate the changed objects to the original objects. This task is complex and covers a wide range of capabilities. A 1st generation design environment must concentrate on realizing basic change control capabilities and insuring that they are realized in such a way as to facilitate future extensions. We therefore constrain the change control to version handling, configuration, and session logging.

Versions are recorded design points for a design element within the design process. Two or more versions may be created from an existing version, therefore version trees must also be handled within the design management environment. Version handling capabilities /FaG89, KSW86, MSS87/ imply mechanisms for:

- Generation/Deletion - automatic generation of design versions in conjunction with the invocation of a design activity. Automatic and manual deletion of non-used or out of date versions should be supported.
- Storage/Retrieval - the ability to put versions in long term storage and to subsequently retrieve them.
- Version Selection - a method for selecting a specific version from the version tree.

Configuration definition will vary depending on the object class but in general it describes a grouping of components at a point of time /EvF86/. At least two types of configuration must be supported within the environment: environment configurations and design configurations. Environment configuration is the set of selection defaults, display defaults, and technology parameters used within an organization or project. Design configuration is the synchronized collection of versions of hierarchically related design elements /KAC86/. Configuration capabilities imply mechanisms for:

- Binding - means for resolving non-explicit references within a design hierarchy.
- Generation/Deletion - automatic generation of design configuration in conjunction with the invocation of a design activity. Automatic and manual deletion of non-used or out of date configurations should be supported.
- Design Activity Footprinting - a record of the configurations used as input to a design tool and those produced as output.
- Configuration Query - a means for allowing the client to examine the subcomponents of a configuration and their use.

**Design session logs** must enable the audit of a design session. Therefore mechanisms must be provided for:

- Design Session Protocols - record of the design transactions completed during a designing session
- Error Handling Protocols - record of the errors which occurred during the design session, their cause, and if applicable, the steps taken to recover.

**Design Management Query**

Design Management Query is an important feature for retrieving information about the particular instantiation of the environment. The environment should supply:

- **Browse** - navigate through the design elements which make up the environment.
- **Select** - display the design actions which may be performed on a specific design element.
- **Info** - display of all appropriate information about the design element.
- **Help** - display of available actions and their invocation.

We have attempted to define the set of design management capabilities required within a first generation design environment /Bau87, DoD86, MSS87, WoL88/. We believe that the listed capabilities will solve the major immediate problems in data and design management. In addition, they provide a basis for extending the environment in a consistent manner.

**GENERAL MODEL FOR REALIZATION**

The realization of any sub-environment within the complete design environment is a three step process /MSR89/. With respect to realization, the design management sub-environment is like any other sub-environment. It is realized by:

- **defining** a model for the design elements and actions which describe the desired design management concepts,
- **generating** the design elements and realizing the design actions which are defined in the design management model, and
- **binding** these elements and actions together with the other design tools and basic environment components to form an instance of the environment.

A general model which is naturally applicable to all steps of the integration process should be used as the basis of the environment realization. This general model must support the modeling of all required design management concepts. Further, the general model must facilitate satisfaction of the general requirements for a design environment (i.e., tool integration, user interaction, and environment extensibility).

The object-oriented approach /HDO87, Ith88, Rob81/ is naturally applicable to all steps of the realization process /HDO87, Mar87/, supports the modeling of required design management concepts, and can be used to satisfy the general requirements of a design environment. It is the state-of-the-art for software environments /Nej89, BST86/ and is being used in research and development by many CAD projects. We have used a model based on the object-oriented approach in the realization of the Cadlab Design Environment.

**REALIZING DESIGN MANAGEMENT REQUIREMENTS**

In this section, we will look at how a general model based on the object-oriented approach satisfies the above mentioned criteria for a general model by showing the realization of design management requirements based on the general model.

**Defining the Design Management Model**

The first step in the realization process is to define a domain model for design management within the design environment. A domain model "is an information model which captures the essential semantics of the domain information within an engineering environment" /DoD88b/.

We have chosen to use a domain model and modeling notation similar to that used within the American EIS project /DoD88b/. The modeling notation consists of 2 formalizations. The first is a graphical description of objects and the relationships between them. A graphical description supplies an overview of a particular concept. The second is a set of object description templates. Object templates are used in the implementation and binding steps.

The graphical representation contains:

- objects with attributes
  
  | object name |
  | attribute name |

- relationship between objects

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>zero, one or many</td>
</tr>
<tr>
<td>#</td>
<td>zero or one</td>
</tr>
<tr>
<td>$</td>
<td>one or many</td>
</tr>
<tr>
<td>$</td>
<td>a specific quantity</td>
</tr>
</tbody>
</table>
An object description template formally describes the object as it will be realized within the environment. It describes:

- type name: object type name
- attributes: object characteristics
- Purpose: short object description
- inherited from: superclass of object type
- inherited by: subclass of object type
- object representation: graphical template
- methods: actions which may be performed on the object
  - accessible by: clients who may perform the action
  - method representation: graphical template

Not all of the listed descriptions must occur in a specific object description template.

Clearly, a complete model of the required DM capabilities is outside the scope of this paper. However, as a validation, we offer a model of 2 specific design management requirements. First, we model access control and second versioning.

Access control is concerned with restricting access to data, tools or other environment components, depending upon the type of operation attempted, specific objects being accessed, and the particular phase of the design life-cycle.

Access to an object or method may be granted or denied a client for many reasons. There are for example, basic access levels which are understood within the environment (e.g., read, write, execute). The basic access level granted a client group determines their access to objects and the application of methods. Within the environment many such access control policies are defined. It is through policies that the environment determines which client may apply which methods to which objects. Figure 2 clarifies the interrelation of methods to be applied to objects by clients.

An important part of access control is a client model, ensuring that clients may only receive services for requests for which they have been previously authorized. A client working within the environment takes on a certain role. Depending on his role, he may perform certain actions on certain objects. Methods are also classified within the environment.

One of the more important classes of objects within the environment is managedObject. Figure 3 provides a description of managedObjects.

The AdministrativeDataObject (ADD) is a kind of managedObject. ADDs maintain information for engineering management and control. Client, method, and DesignDataObject are clearly among the kinds of ADDs.

DesignDataObjects (DDO) contain some part of a design which is itself a part of a project. (If we considered mapping EDIF /EDIFS/ to this model, we could model "EDIF-views" to be DDOs).
One of the most requested requirements is version handling. In general version handling is concerned with:

- tracking the evolution of an object
- modification of an object resulting in a derivation
- initial (or root) states which are neither a version nor a derivation

Versions may have different states

- initial (or root) version
- derived version, obtained by several modification steps
- replaced version, i.e. the modification history is unknown

The graphical representation of version object is shown in figure 4.

![Graphical representation of a version object](image)

Figure 4: graphical representation of a version object

It is not sufficient to describe the objects of an environment but also the methods applicable to the objects. As previously mentioned, mechanisms or services are required to work with versions (e.g., generation/deletion of versions and selection of versions). These mechanisms are described in the object description template. The following object description template is an example of a description for a version object.

type name: version object
attributes: versionIdentifier;KindOfState; etc.
purpose: record of an object state within the evolution of an object
methods: CheckinVersion(versionOID:ObjectID; etc.); Description: generates a derived version accessible by: designer;design tool
GenerateVersion(versionOID:ObjectID; etc.); Description: creates an initial version accessible by: design tool

Delete Version(versionOID:ObjectID; etc.); Description: deletes a version accessible by: designer;design tool
QueryVersion(versionOID:ObjectID; etc.); Description: returns the KindOfState accessible by: designer;design tool etc.

The model definition phase is probably the most important and perhaps the most difficult phase of the realization process. It is in this phase that all possibilities for using the sub-environment within the integrated design environment are determined. Once the model definition phase is completed, the implementation phase begins.

Implementation Phase
The Implementation phase entails the implementation and test of the methods on the specific objects which compose the sub-environment. The implementation phase is independent of the domain model being implemented. During the implementation phase an object type description is created for each object type in the sub-environment model. Within the CWS Design Environment, an object type description is composed of three pieces of information:

- a description of the data which may be contained within an object of this type.
- a description of the methods applicable to an object of this type.
- a description of the graphical representation of an object of this type and its methods.

The CWS Design Environment includes an object-oriented integration system (OIS) for registering object types and handling them in a uniform way within the environment. It also includes utilities for testing individual methods on individual object descriptions. After the individual methods and objects are tested, the object types of the sub-environment are bound together with all other relevant object types to generate an instance of the environment.

Binding Phase
Binding entails the registration of object types within the OIS, compilation of the resulting OIS, and link of all relevant object code and libraries. This phase is dependent on the implementation of the underlying model. The result of this phase is an executable environment.

CONCLUSION, STATUS, AND FUTURE WORK
We have discussed design management requirements for a 1st generation design manager within an integrated design environment. In our opinion, this includes the general design environment requirements of tool integration, user interaction, environment extensibility, and design management. Where,
the specific design management requirements include environment modeling, change control, and design management query.

We have proposed a general model which satisfies these requirements and have shown how it is used to realize required design management capabilities within an integrated design environment. The model is based on the object-oriented approach. The object-oriented approach is naturally applicable to all steps of the realization process, supports the modeling of required design management concepts, and can be used to satisfy the general requirements of a design environment.

The presented approach has been used to realize design management capabilities within the prototype Cadlab Workstation Design Environment /GKK87/. An initial implementation of most of the required design management capabilities within the design environment has been completed.

From our initial work, we intend to build a complete model for required design management capabilities and implement a set of general design management services. These design management services will be available on the NMP-Cadlab common framework /HSG89/. The model and services will primarily concentrate on hierarchical design, version/configuration handling, and process flow control.

ACKNOWLEDGEMENTS

The work presented in this paper has been partially funded by ESPRIT, IDPS project 2270. The implementation of the concepts presented in this paper were realized by the actual development of the Cadlab Work Station Design Environment. The authors wish to thank all the people involved in the realization of the CWS, especially the user interface group and the data handling group. Further, we would like specifically to thank Klaus Gröning, Ulli Blancke, Jürgen Strauss, Charles White, and Thomas Kathöfer for the helpful discussions about the concepts presented in this paper.

REFERENCES


[DoD86] The Department of Defense "Requirements for Engineering Information Systems", July 1986

[DoD88b] The Department of Defense, Workshop Baltimore, 14-17 November 1988


[Ith88] "ITHACA", Project Description (September 1988), 1-57.


[Rob81] D. Robson "Object-Oriented Software Systems", Byte,(1981), 74-86