A Flexible Access Control Mechanism for CAD Frameworks

Alfred van der Hoeven Olav ten Bosch
Rene van Leuken Pieter van der Wolf

Delft University of Technology
DIMES Design and Test Centre
Delft, The Netherlands

The possibility to control access to data and tools is vital in many organizations where design activities take place. However, in other organizations a stringent access control mechanism would seriously hamper the cooperation among designers. In this paper we present a configurable and unobtrusive access control mechanism for CAD frameworks that is flexible enough to support a wide range of access control policies. Furthermore, we present the realization of the access control mechanism in the context of the Nelsis CAD Framework.

1. Introduction

A CAD framework [1][2][3] is a software infrastructure that provides a common operating environment for CAD tools. It provides designers with a number of services that help them to manage their design data and to control the design process. These services include concurrency control, version management, design methodology management and browsing facilities. Another important framework service is access control. Access control in a CAD framework is used by designers and framework managers to specify and enforce access permissions to resources that the CAD framework knows about. A *resource* can be an entire class of objects or it can be a specific object that is managed by the framework. Examples of *class resources* are *the class of design projects* and *the class of design objects*. Examples of *object resources* are a *specific design project* or *a specific design object within a specific design project*. In this paper, we discuss the requirements for an effective access control mechanism and we derive an information model that describes the structural semantics of the access control related information maintained by the CAD framework. Subsequently we show how this information model is used to realize the access control mechanism in CAD frameworks in general and more specifically in the Nelsis CAD framework [4].

In this paper, we assume frameworks that have been designed according to the widely accepted CAD system architecture of Figure 1. The access control mechanism is offered as one of the kernel framework services. We also assume that a CAD framework allows the logical distribution of design data and design activities via *design projects*. A design project is a local environment in which design activities take place on locally defined design objects, possibly in accordance with a locally defined design flow.

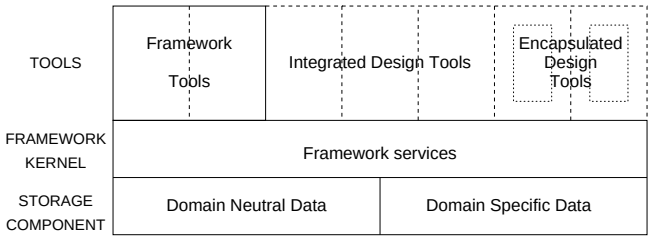

Figure 1. CAD system architecture.

2. Access control requirements

Framework users expect different things from an access control mechanism at different times and in different environments. In some places, for instance in a university research environment, people do not want to be bothered by an access control mechanism. In other places, for instance in a design company environment with strict design policies, people want to use the access control mechanism to limit access to sensitive design projects and the design objects therein, and to reserve certain design tasks, such as sign-off tasks, to distinguished users. Therefore, it is imperative that the access control mechanism is flexible and configurable so that it can cater to everyone's needs.

Furthermore, if people have the proper access permissions, the access control mechanism should not be noticeable. The mechanism must do its work unobtrusively. For instance, it should not enforce a separate system login mechanism on frameworks that do not already have this and there should not be any significant time penalty in the verification of access

This research is supported in part by the commission of the EU under project 7364 (JESSI-Common-Frame).

permissions. Naturally, if people do not have proper access permissions, they are to be notified discretely.

An important and obvious property of an access control mechanism is that it is secure. It must not be breakable by ordinary users of the framework and the framework's underlying operating system. On the other hand, the access control mechanism of the underlying operating system may not be in conflict with that of the CAD framework. Thus, if access to a design object has been granted by the framework, it should not be denied by the underlying operating system.

The access control mechanism manages information that is specific to its own operation. This information also requires access control, to be performed by the mechanism itself.

It must be straightforward for framework users to inspect and modify, if permitted, resources and access permissions via an interactive access control browser. The mechanism should allow for the easy modification of the access permissions of a designer, for instance when he leaves a company or when his function within a company changes.

3. Access control in other environments

A well-known example of an access control system is the file access control mechanism of Unix. For each file or directory in a file system, the user community is divided into (1) the *user* who owns the file, (2) a *group* of persons related to the user and (3) the other people in the community. Read, write and execute permissions for each file or directory can be defined for each of the above mentioned categories. One might argue that, given the implementation of a CAD framework in a Unix operating environment, the files and directories used in that implementation could also be used to realize the access control mechanism. This approach is followed in OCT [5]. Although appealing because of its simplicity, this approach exhibits two major disadvantages with respect to CAD frameworks. Firstly, not all resources managed by the framework can be mapped onto files or directories of the Unix file system. This applies especially to the class resources managed by CAD frameworks, such as *the **class** of design objects that are not yours*. Secondly, the access permissions of files and directories in the Unix file system do not allow for the distinction of more than one group of users per file or directory. Hence it is too inflexible.

In some versions of Unix, notably HP-UX [6], it is possible to specify an access control list (acl) for each of the files and directories in the file system. In these Unix versions, the second disadvantage mentioned above no longer applies.

The Cadence framework [7] provides an access control mechanism that is similar to that of Unix. The framework distinguishes between *library* objects, *cell* objects, *view* objects and *cellview* objects, each of which can be protected via access permissions. For each of these objects an *owner*, *working group* and *user group* needs to be defined by the creator of the object. The other users belong to the *public*. The designer may specify *read*, *edit* and *delete* permissions for each of the 4 mentioned user categories. The disadvantages of the Unix access control system also apply to the Cadence access control system. It restricts itself to the access control management of individual objects in the Cadence system and does not allow for the access control management of CAD framework class resources.

In the Jessi Common Framework [8] (JCF), the access permissions to data and tools are organized in *user roles*. JCF provides four user roles whose associated capabilities cannot be modified: *FrameworkAdministrator*, *MethodologyManager*, *ProjectManager* and *Design-Engineer*. Former roles include the capabilities of latter ones. Each user registered by the framework has one specific role but can be part of a number of teams. If a team has been assigned to a project, all members of that team have access to the project with the privileges as specified by their roles. Users have to log into the framework to obtain access to its databases. The JCF access control system follows an approach that is different from the Cadence approach. Instead of managing the access permissions of the specific objects in the framework, JCF makes it possible to manage access to more abstract framework class resources. The disadvantages of this approach are:

- The user roles are fixed. No roles can be added nor can their list of associated privileges be modified.
- It is not possible for a user to play a certain role in one team and a different role in another team.
- The access rights of a team that has been assigned to a project cannot be restricted. It is for instance not possible to give an existing team, containing members with write permissions, read-only permissions to a library project whose contents is being developed or maintained by another team.

MPCAL [9] is a multi-person calendar system with a sophisticated access control system. For each calendar in the MPCAL system, the owner can define a set of users and a set of roles and specify which users may play which roles. A role consists of a name and a set of rules describing access permissions for the operations (e.g. *make appointment* or *show schedule*) supported by MPCAL. For each calendar, the owner can thus determine who is allowed access to which calendar operations in a flexible manner. Although the complexity of a CAD framework is different from that of a calendar system, the ideas behind MPCAL's access control system are generally applicable. A disadvantage of this approach is that for each calendar, the roles are defined separately. This makes it difficult to attach an interpretation to roles used by the calendar system since roles with the same name may represent different access permissions in different calendars.

All of the above described CAD framework access control mechanisms fulfill one or more of the requirements that we stated earlier in this paper. However all of them also fail to satisfy one or more of the specified

requirements.

4. The information model

In this section we describe an access control mechanism that satisfies the specified requirements. We adopt a formal data modeling technique to represent the relevant entity types and their relationships. The modeling technique used in the figures displayed below is OTO-D (Object Type Oriented Datamodel) [10]. In this semantic data model, the boxes represent the object types (classes) and the lines connecting the centers of these boxes denote attribute (part-of) relationships.

4.1 Teams and members

In most design environments, designers are organized in teams. Designers usually are a member of more than one team and perform a certain role in each team. The role they perform may differ between teams. For instance, in one team a designer may play the role of team leader, whereas in another team he may play the role of observer. This is modeled in the dataschema of Figure 2.

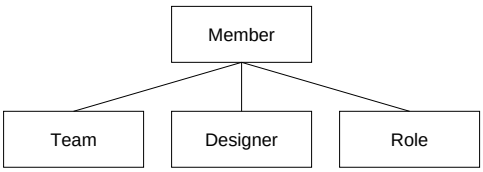

Figure 2. Dataschema modeling teams and members.

This dataschema specifies that a `Designer` can be a `Member` of a `Team` and play a specific `Role` in that `Team`. `Roles` are used to specify the resources a designer or team has access to and in which manner. For a further explanation of the type `Role`, see section 4.2. In design frameworks that do not have a separate framework login system, the login system of the underlying operating system is used to determine the identity of the designer. Observe that in the JCF framework, a `Designer` instead of a `Member` has a `Role`. Also note that, in case of Unix as the underlying operating system, teams may be configured to coincide with Unix groups. Figure 3 gives an example of two teams and their members. The first column of the table contains the `Member` object identifiers.

4.2 Roles and privileges

Since we do not target the access control mechanism for any specific CAD framework nor for any specific application of the framework, we cannot predict what resources are required and how they are organized. Therefore, the access control mechanism should allow the specification of resources and privileges providing access to resources and the assignment of privileges to roles. This is modeled in the dataschema of Figure 4.

In this dataschema, a `Privilege` is the right to access its `Resource` in the manner specified by its `AccessType`. Examples of `AccessType` are *create* and *delete*. Examples of `Resource` are *the class of design objects* and *the class of projects*. A `Permission` denotes that the associated `Role` has permission to execute the associated `Privilege`. The instances of `Role` and `Privilege` thus form an access control matrix whose entries are defined by `Permission`. We use `Roles` for the organization of privileges because it relieves a team manager from having to create and delete numerous `Permissions` when the position of a designer in a team changes. Instead, he can just assign him another role. Also, it is often the case that several members of a team have the same privileges and this can be modeled via roles. `Roles` can be organized hierarchically via the `RoleHierarchy` object type. The `Parent-Role` of a `RoleHierarchy` also has the `Permissions` of the `Child-Role` of that `RoleHierarchy`. This enables the easy specification of roles, which is especially useful in the definition of team roles (see section 4.3). A `Member` of a `Team` has access to resources via his `Role` attribute within the operating scope of his `Team`. For an example listing of role hierarchies and their role attributes, see figure 5.

Member	Team	Designer	Role
MB1	Nelsis	Ank Russo	secretary
MB2	JCF	Ank Russo	secretary
MB3	Nelsis	Alfred van der Hoeven	engineer
MB4	JCF	Alfred van der Hoeven	framework manager
MB5	Nelsis	Peter van der Wekken	engineer
MB6	Nelsis	Peter van Putte	engineer
MB7	JCF	Peter van Putte	engineer
MB8	Nelsis	Pieter van der Wolf	engineer
MB9	JCF	Pieter van der Wolf	framework manager
MB10	JCF	Olav ten Bosch	engineer
MB11	JCF	Rene van Leuken	team manager
MB12	Nelsis	Rene van Leuken	team manager
MB13	Nelsis	Edwin Essenius	engineer
MB14	JCF	Kees Schot	engineer
MB15	Nelsis	Wim Tiwon	project support

Figure 3. An example listing of team members according to the schema of Figure 2.

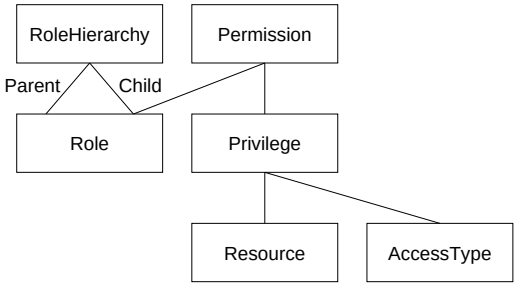

Figure 4. Dataschema modeling roles and privileges.

Resources can be defined within the scope of a design environment (framework-wide) or within the scope of a design project (project-wide). Framework-wide resources are usually class resources. Examples are *the class of design objects* and *the class of projects*. Project-wide resources are usually specific objects. Examples are *a specific design object within a specific project* or *a specific*

Role-Hierarchy	Parent-Role	Child-Role
RH1	framework manager	engineer
RH2	framework manager	project support
RH3	project owner	project observer
RH4	team manager	engineer
RH5	project owner	framework manager

Figure 5. An example listing of role hierarchies according to the schema of Figure 4.

design task within a specific project. We thus need a mechanism to specify resources and privileges and verify the permissions to use them at the framework level as well as at the level of a project. For now, we will restrict ourselves to resources specified at the framework level and defer the topic of project-wide resources and privileges to section 4.6.

4.3 Projects and project partners

In the introduction, we stated that design information is organized in design projects. A project can be a library containing design components or a project may contain a top level design description consisting of design components from a library (projects may also be used in other ways). Since projects may be used differently by different teams, these teams may require different access permissions to the same project. Team 1 may for instance be a library development team and thus needs *write* permissions for its library projects. Team 2 may be a chip development team using the libraries of team 1 and thus only needs *read* access to these libraries, independent of the roles of its members. This is modeled in the dataschema of Figure 6.

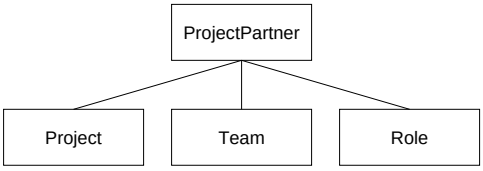

Figure 6. Dataschema modeling projects and project partners.

In this dataschema, a `Team` can be a partner in a `Project` with a specific `Role` via the object type `ProjectPartner`. Members of that team do not have access to resources in the project other than those specified by the `ProjectPartner`'s `Role` attribute. As specified in Figure 4, the role of a project partner may be composed of child-roles. Observe that the JCF framework follows a similar approach here, but differs in that a `ProjectPartner` does not have a `Role` attribute. JCF hence lacks the possibility to restrict the access permissions of a team per project.

4.4 The overall access control schema

Figure 7 displays the overall dataschema of the access control mechanism, showing the integration of projects, teams and roles.

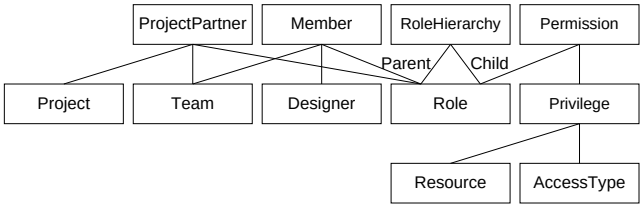

Figure 7. The overall access control dataschema.

4.5 Framework-wide resources

The framework-wide resources are divided into two groups. The first group contains the resources that are defined and verified within the scope of a framework instance. This group includes various resources that deal with the access control mechanism itself. The second group contains the resources that are defined within the scope of a framework instance but whose use is implemented and verified within the scope of a specific project. An example of this group is *the class of design objects that are not yours*. The use of these resources is thus restricted via the role of the project partners. A list of possible privileges is given in the table of figure 8.

	Privilege	Resource	AccessType
	PR1	Team	create
	PR2	Team	modify
	PR3	Team	delete
	PR4	Role	create
	PR5	Role	modify
	PR6	Role	delete
	PR7	Privilege	create
	PR8	Privilege	delete
	PR9	Project	create
*	PR10	Project	delete
*	PR11	Project	access
*	PR12	Team-Project	add
*	PR14	Team-Project	delete
*	PR15	Design Object	create
*	PR16	Design Object	delete
*	PR17	D.O. not yours	delete
*	PR18	D.O. not yours	read

Figure 8. List of privileges specified at the framework level. The marked privileges (PR10-PR18) are verified at the project level.

Note that many of these privileges are vulnerable to the *bootstrapping* problem which is that as long as no one has the privilege to use a certain resource then no one has permission to give others the privilege to use that resource. Take for instance the resource *Team* with access type *modify* which allows a team member to add or remove members to/from his team. A newly created team will initially be empty, and thus no one will have the privilege to add members to that team. Therefore, for a number of privileges we need to adopt the policy that:

> *If no one has been explicitly granted the privilege to use a resource within the appropriate scope (project, team or both), then everybody may use that resource.*

Thus in the case of an empty team, everybody may add designers to that team. An additional benefit of this approach is that in an unrestricted design environment, very few privileges need to be granted. Note that this

policy does not apply to every privilege. For instance the privilege *PR17* with resource *Design Object not yours* and access type *delete* should not be usable by everyone if no one has been explicitly given permission to use it. Hence, `Privileges` also need a `Policy` attribute to specify what to do if nobody has explicit access to a privilege.

An important restriction is that only those people that have permission to create, modify or delete a role, can give permission to others to do the same. These privileges are typically reserved for framework managers and should not be accessible by others.

4.6 Project-wide resources

In the dataschemas displayed above we only modeled framework-wide resources, privileges and their assignment to roles. The specification of project-wide resources cannot be done at the framework level because that level is beyond their scope. It is for instance not possible, nor desirable, to limit access to a specific design object in a specific project via a framework-wide resource. To model project-wide resources and their use at the project level, we use the roles that were defined at the framework level. The roles used at the project level thus are a subset of the roles used at the framework level. See Figure 9.

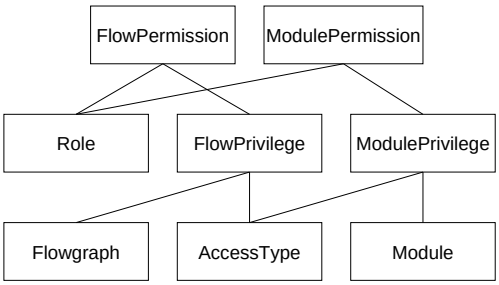

Figure 9. **Definition and use of resources within the scope of design projects.**

The resources defined at the project level are usually objects that exist within the context of a specific project. For example, the Nelsis CAD Framework contains flowgraphs, describing design tasks within a design flow configuration, and modules, which are containers for the different versions of a design description [4][11]. Access control is required for flowgraphs as well as modules. Other frameworks may need the ability to restrict access to workspaces. The Nelsis case has been modeled in Figure 9 where we allow flowgraphs and modules to be used as resources in the access control system. Examples of the `AccessType` attribute of a `FlowPrivilege` are *execute* and *open*. Examples of the `AccessType` attribute of a `ModulePrivilege` are *read* and *write*. Observe that the ability to define privileges at the project level and to assign them to roles, requires the specification of resources and privileges giving access to this ability. These privileges are specified framework-wide but their use is verified within the scope of an individual project.

5. Realization

In this section, we discuss the implementation of the access control mechanism and its integration into the Nelsis CAD Framework. Most of the discussion applies to other CAD frameworks, as well. For the implementation of the access control mechanism we use the widely accepted CAD system architecture of Figure 1, which is also used in the Nelsis CAD framework.

5.1 The framework process structure

Because part of the data managed by the access control mechanism must be framework-wide available and maintained (e.g: team information, resource specifications) we have chosen to implement the access control mechanism as a dedicated server which communicates with the project servers. In CAD frameworks that already have framework-wide processes, the access control mechanism may also be implemented as part of one of these processes. The resulting process structure of the Nelsis CAD framework is depicted in Figure 10.

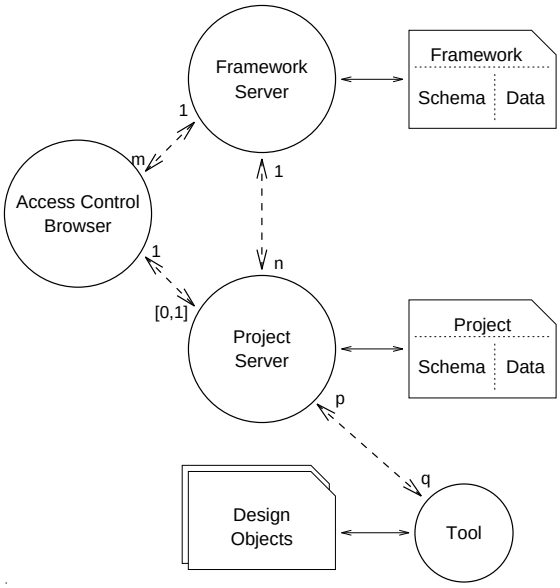

Figure 10. **Process structure in the Nelsis CAD Framework. Circles denote processes, 'files' denote data.**

For each instance of the Nelsis CAD framework there is a separate framework server process which incorporates the access control manager. For each (active) design project, there is a project server process, connected to the framework server, which manages project-specific data. In frameworks that do not have separate processes for managing design data at the project level, the framework server and the project servers collapse into one process.

Since the resource and privilege information is distributed between the framework server and the project servers, the verification of access permissions is also distributed. For privileges defined within the scope of a project, the project server first determines the roles that

have access to the requested privilege. Next, the project server verifies with the framework server that a designer who requests the privilege is permitted to play one of these roles *and* that the team in which he plays that role is also allowed to play one of these roles within the project. For privileges defined at the framework level but used at the project level (e.g. *the right to create design objects*), the project server simply establishes with the framework server that the designer has access to that privilege. Privileges defined and used at the framework level are handled entirely by the framework server.

An important aspect of the implementation of an access control mechanism is its security as discussed in section 1.1. We feel that security can merely be reached if designers can only obtain access to data managed by the framework through tools connected to the framework. To obtain that goal, all design data need to be owned, unaccessible for others, by a special user (e.g. *nelsis*) and all tools that have access to the design data and design meta data must run under the ownership of that special user.

5.2 The access control browser

Via the access control browser, designers can view and modify the information related to the access control mechanism through a navigational user interface as depicted in Figure 11.

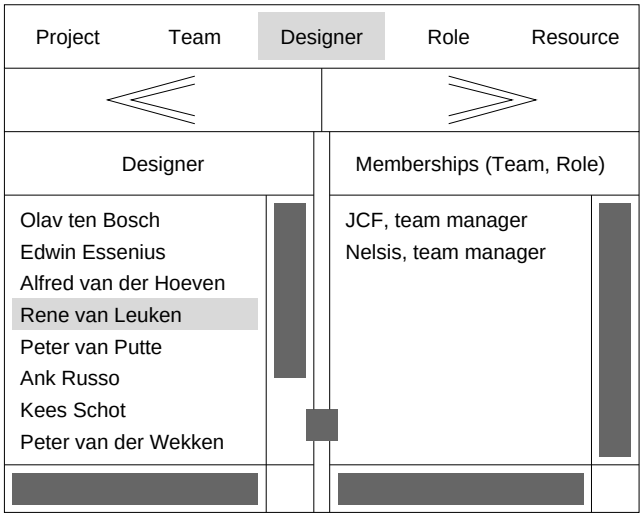

Figure 11. The Access Control Browser of the Nelsis CAD Framework

A framework user can start with any one of the 5 lists as displayed at the top of figure 11. From each list displayed by the browser, he can select an item and have a list of related items displayed. For instance, a designer can have the list of project partners (team, role) of a selected project displayed or the list of privileges a selected role has access to.

For each list or sub-list there are appropriate commands for adding entities to, deleting entities from or modifying entities of that list. There are also commands for saving changes, and for connecting to or disconnecting from projects. If a designer is not authorized to apply some specific browser command, the associated command button is disabled.

6. Conclusion

We have derived a generally applicable access control mechanism from a comprehensive set of requirements. In the course of this derivation we have used the OTO-D datamodel to arrive at a dataschema with a formally defined semantics. In this schema, we assign privileges to roles and we use teams to organize framework users, resulting in a high level of flexibility of the access control system. An access control browser can be used to inspect and modify access control data. The security of the access control system partly stems from the scoping of privileges as a result of which teams can control access to their own design environments, and partly stems from the fact that access to data managed by the framework is only allowed via tools managed by the framework. As a result, the access control management system we described and employ in the Nelsis CAD Framework, is robust against misuse, easy to configure, nonobtrusive, and has a hardly discernible performance penalty. Furthermore, its concepts can also be applied to other CAD frameworks.

References

1. CFI Architecture Technical Subcommittee, ‘‘CAD Framework Users, Goals, and Objectives, Version 0.92’’, *CAD Framework Initiative*, (December 1990).
2. P. van der Wolf, ‘‘Architecture of an Open and Efficient CAD Framework’’, Ph.D. Thesis, Delft University of Technology, Delft (June, 1993).
3. T.J. Barnes, D. Harrison, A.R. Newton, and R.L. Spickelmier, *Electronic CAD Frameworks*, Kluwer Academic Publishers, Boston (1992). ISBN 0-7923-9252-3
4. P. van der Wolf, P. Bingley, and P. Dewilde, ‘‘On the Architecture of a CAD Framework: The NELSIS Approach’’, *Proc. of the European Design Automation Conference*, pp. 29-33 (1990).
5. Mario Silva, David Gedye, Randy Katz, and Richard Newton, ‘‘Protection and Versioning for OCT’’, *Proc. IEEE 26th Design Automation Conference*, (1989).
6. Hewlett Packard, *HP-UX Release 8.0 Reference, Volume 3*, Hewlett Packard, England (January 1991).
7. Cadence, ‘‘Design Framework II Reference Manual’’, Cadence Design System Documentation, Cadence Design System, Inc. (October 1991).
8. Siemens Nixdorf Informationssysteme AG, ‘‘Jessi Common Frame V2.0’’, Desktop User’s Guide, Siemens Nixdorf Informationssysteme AG (1993).
9. I. Greif and S. Sarin, ‘‘Data Sharing in Group Work’’, *Proceedings of the First Conference on Computer-Supported Cooperative Work*, pp. 175-183 Austin, Texas USA (December 1986.).
10. J.H. ter Bekke, *Semantic Data Modeling*, Prentice Hall, Englewood Cliffs, N.J. (1992). ISBN 0-13-806050-9.
11. K.O. ten Bosch, P. Bingley, and P. van der Wolf, ‘‘Design Flow Management in the NELSIS CAD Framework’’, *Proc. 28th ACM/IEEE DAC*, pp. 711-716 (1991).