Prolog to

Managing Design Data: The Five Dimensions of CAD Frameworks, Configuration Management, and Product Data Management

An introduction to the paper by van den Hamer and Lepoeter

The field of Design Data Management is faced with many complexities: designers have more files to contend with, design processes are increasingly complex, data updates occur more frequently, and computer-driven product development has proliferated the quantity of information to be managed.

Many names fall under the rubric of Design Data Management: CAD Frameworks. Configuration Management. Product Data Management, Design Management, and Engineering Data Management. Such variety has problematized the analysis of requirements, solutions, and terminology.

In the effort to find common ground, this paper offers a frame of reference, a five-dimensional model for Design Data Management. The dimensions are as follows: Versions, Views, Hierarchy, Status, and Variants. The model proposed in this paper will assist designers in analyzing user requirements, discussing alternatives, and classifying available support tools.

Since design usually occurs in steps, processes are usually iterative, and the result of each iteration can be looked at as a "version" of the design information. At its simplest, versioning is an automatic and transparent backup mechanism; however, actually implementing versioning is more difficult since CAD tools and operating systems have been built without versioning in mind. Two models for versioning are diagrammed: 1) versions as a numbered sequence of intermediate results, and 2) a version tree model that sequences and also tracks how new versions are created.

As many products are too complicated to depict in any single representation, it is necessary to look at them from multiple "views." In this model, the View dimension is linked to the development process steps by which the views are derived from other views, either by automatic or manual transformations.

Another common method for managing complex designs is to break the design into smaller parts, which leads us to the third dimension: Hierarchy. Two hierarchical paradigms are diagrammed: a nested physical structure and an abstract treelike graph.

The Status dimension of design is used to manage quality, consistency, functionality, reliability, and safety. Common instances of the status dimension would be verification steps and validation steps.

The Variants dimension occurs when more than one flavor of a product is needed. While such variants tend to be quite similar, they are not identical, and thus require management of both their common and different features. This dimension is the least understood of the five, and is getting more attention of late.

The five-dimensional model is simple at face value, but becomes more powerful when we realize that all dimensions are relevant for any of the design disciplines: mechanical design, circuit board design, IC design, software design, system design, and multimedia contents design. Moreover, since the dimensions cover the basics of the design process, they can be used for analyzing the very processes themselves.

Usually, it is necessary to analyze a design process using several of the dimensions simultaneously, which unleashes powerful capabilities for real-world design management, although multidimensional combinations are difficult to explain, configure, and operate, especially in the field of CAD Frameworks. Several combinations of dimensions are discussed in the paper: Versions + Hierarchy (including a comparison of static and dynamic hierarchy models); Hierarchy + Views (including a comparison of the level-by-level model with the nonisomorphic hierarchies model); and Versions + Views (data-centric and roadmap models).

In practice, Design Data Management requires three or more dimensions, and it becomes necessary to prioritize dimensions, since no more than two to three of them can be handled consistently with current tools. For software configuration management systems, Version, Status and

0018-9219/96$05.00 © 1996 IEEE

Variant dimensions are critical. CAD Frameworks for electrical design must support multiple Views and Hierarchical design and, increasingly, Status and Versions. Electronic archiving systems need to accommodate multiple Versions, Hierarchical product structures, simple Design Views, and Status. In general, the Variants dimension is slowly emerging as a priority across disciplines.

The authors situate their work within previously published results on data management modelling. The only application areas that have any significant record of such research are software engineering and electronic (VLSI) CAD.

—*Jim Esch*

Managing Design Data: The Five Dimensions of CAD Frameworks, Configuration Management, and Product Data Management

PETER VAN DEN HAMER AND KEES LEPOETER

With the increasing complexity of designs, Design Data Management is regarded as a key enabling technology to achieve higher efficiency in the development of complex products. However, this technology has proven to be surprisingly complex. Commercial solutions in the area of CAD Frameworks, Product Data Management, and Configuration Management Systems currently only solve parts of the total problem, and the acceptance of these systems in industry is still relatively low.

In this paper, a model is presented which can be used as a frame of reference for the general problem of managing design data. The model distinguishes between five orthogonal dimensions, which are each, when considered separately, quite simple. In practice, however, two or more dimensions must be handled simultaneously to solve real-world problems. This turns out to be nontrivial because multiple fundamentally different solutions exist for each combination of two or more dimensions.

We believe that this 5-D frame of reference can contribute to the creation of better quality solutions in the area of Design Data Management. The 5-D model can also help to explain the familiar phenomenon that a solution which has, for example, proven to be adequate for software development may be rejected by integrated circuit designers, and vice versa.

I. Introduction

The rapid proliferation of computers in the field of product development has lead to a sizeable amount of electronic design information which needs to be managed. The field of Design Data Management has proven to be surprisingly complex for a variety of reasons: the increasing number of files which designers have to deal with, the steadily increasing complexity of the design processes and the high rate at which design data tends to be updated. In addition, some development organizations have started regarding Design Data Management as a key enabling technology to achieve better development efficiency, throughput time reduction and other design process-related business goals.

There are many synonyms for the term "Design Data Management." Each application field has its own names for it: CAD Frameworks (ECAD), Configuration Management (software, system), Product Data Management (mechanical design, system design), Design Management (ECAD), and Engineering Data Management (certain design data management products).

In the past 15 years, many ideas and concepts have been presented in the literature,[1] usually embedded within a description of the author's novel software system for managing design data. In contrast, there have been few publications about the basic nature of the problem itself, how the requirements differ across the different design disciplines or, ways of systematically evaluating available solutions with regard to user needs. Discussions on this level are partly hampered by the relative isolation of some of these disciplines, but also by the lack of a common terminology for basic notions—even within a single discipline.

This paper aims to present a *frame of reference* for the general Design Data Management problem. We present a model which distinguishes between *five orthogonal dimensions* which are in our view fundamental to Design Data Management. This 5-D model is based on the experience of our respective groups within Philips in managing design data within ECAD, software development, end-product/system development and mechanical CAD. In its current form, the model is roughly five years old. In this period, it served as a tool for analysing Design Data Management issues. It is worth noting that these five dimensions can also be used for modeling design processes themselves (see Appendix A).

II. The Five Dimensions of Design Data Management

In this section, we will introduce what we call "the five dimensions of Design Data Management." These five dimensions, when considered separately, are each quite simple: developers in any particular discipline will recognize most, if not all, of them as issues which complicate the

Manuscript received April 10, 1995; revised October 24, 1995. This work was supported in part by the JESSI-COMMON-FRAME project, Esprit project number 7364.

P. van den Hamer is with IST/SEM/Data and Process Management, Philips Research, 5656 AA Eindhoven, The Netherlands.
K. Lepoeter is with the Electronic Design and Tools Department, Philips Research, 5656 AA Eindhoven, The Netherlands.

Publisher Item Identifier S 0018-9219(96)00870-5.

[1] Relevant references are presented in the next sections.

design process and its designs. Familiar solutions for handling each of these dimensions are described. We thus do not claim that the dimensions themselves are new. In fact, inventing new dimensions would be like claiming to have discovered a new major design problem which designers have never noticed before.

A. The Version Dimension

Designers typically modify design information in multiple steps. Each step results in a new *version* of the design information. Sometimes a designer needs these steps to add functionality in a controlled way, e.g., when creating a complicated design from scratch. In other cases, versions correspond to the steps which the designer takes in correcting mistakes (e.g., debugging) or in optimizing the design (i.e., minimizing the size of a chip). In all cases, however, a new version is created because the designer wants to *modify* the design.

In operating systems like UNIX or MS-DOS, newer versions of design files simply *overwrite* previous versions. This overwriting of the previous version can be a problem because a new version may later turn out to be a step backward instead of a step forward: the seemingly good idea may not be so good after all. Alternatively, the designer may have inadvertently ruined parts of the design without knowing it, and may thus need an earlier copy of the design corresponding to the state of the work directly before the accident.

Because designers need the ability to go back to the previous versions, and few operating systems have this capability, CAD Frameworks provide this functionality. In its simplest form, this versioning functionality can be regarded as an automatic and transparent backup mechanism. It has also been suggested that the availability of versions tends to promote an exploratory design style in much the same way that an "undo" function in an editing tool reduces the threshold to perform a tricky operation.

Implementing design versioning is technically rather tricky because both the CAD tools and the underlying operating system were developed without versioning in mind. In practice, versioning is thus implemented by various combinations of intercepting read/write accesses by the CAD tools and by moving/renaming files before and after tool read/write operations. Such techniques are known as tool *encapsulation* (where the functionality is added in a so-called "wrapper") or *integration* (where the tool itself is modified). It is worth noting that this problem is only fully tackled in CAD Frameworks; in software configuration management systems or electronic archives, the designers typically work on a nonversioned copy of the data files which is kept separate from the versioned copies stored in the archive.

Fig. 1 shows a schematic representation of two common models for design versioning. In diagram (a), versions are simply a numbered sequence of intermediate design results. This technique is encountered in many software configuration management and archive-like products. A disadvantage of this model is that the version numbers

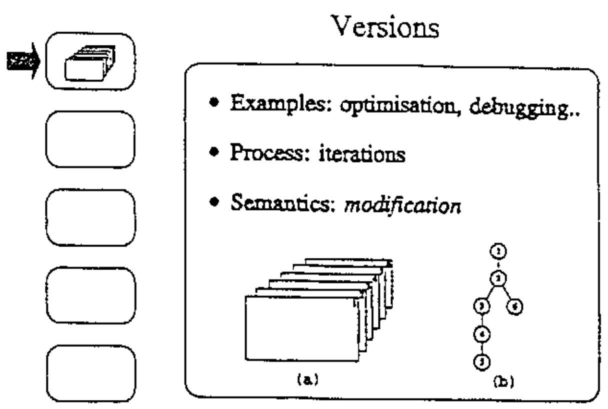

Versions
• Examples: optimisation, debugging..
• Process: iterations
• Semantics: *modification*
(a)
(b)

Fig. 1. Design versions correspond to updates made to the design.

really serve two different purposes: on the one hand, the numbers record that version 6 was submitted directly after version 5, but on the other hand the numbers suggest that version 6 was created by modifying version 5.

Unfortunately these two aspects, time sequence and modification history, do not always coincide. In (b), version 6 was actually created by backtracking to version 2 and creating a new modification. As illustrated by this example, the *version tree* model thus also keeps track of the way in which new versions are created. Version trees are supported by several CAD Frameworks and some advanced software configuration management systems and are typically displayed graphically in the user interface.

We will conclude this overview of versioning with a few examples to illustrate its practical importance in data management:

1) Designs which are released and have found their way to customers. The design data for these products must remain available for many years after termination of production due to service and legal liability requirements. In this period many new versions will typically have been created.
2) Software products typically go through so many changes that they get a two-level update identification: one level corresponds to changes in the product's functionality (e.g., new manual required) and one corresponding to internal improvements (e.g., bug fixes).
3) Philips uses a comparable two-level version identification scheme for change management of consolidated product information: a 12-digit article code is updated whenever the "form, fit or function" of the design changes, and a secondary identification (typically a date) is used for manufacturing changes which do not normally affect the usage of the product.

B. The Views Dimension

Many products are simply too complex to represent in only one single type of representation or diagram and are thus often described at multiple levels of abstraction or "views." A new electrical development project starts by making a "view" with a high-level description of the design and, after this has been analysed, discussed, and optimized, proceeds to a lower level of abstraction with greater detail.

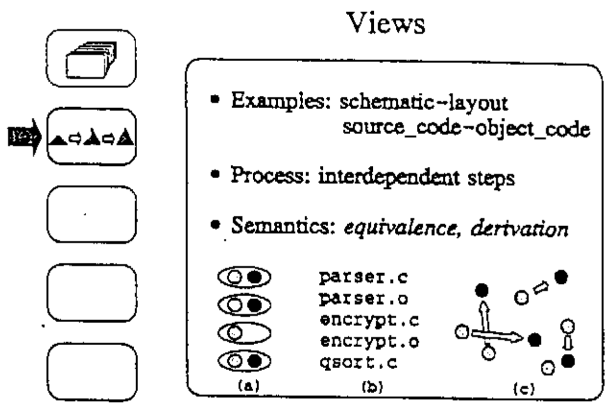

Views
• Examples: schematic-layout source_code-object_code
• Process: interdependent steps
• Semantics: *equivalence, derivation*
parser.c
parser.o
encrypt.c
encrypt.o
qsort.c
(a)
(b)
(c)

Fig. 2. Design views represent different aspects of the same design.

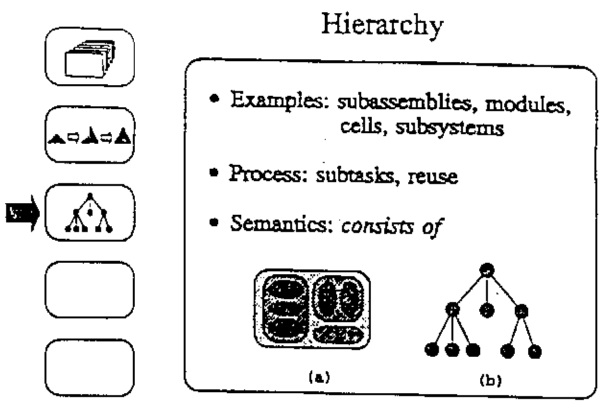

Hierarchy
• Examples: subassemblies, modules, cells, subsystems
• Process: subtasks, reuse
• Semantics: *consists of*
(a)
(b)

Fig. 3. Design hierarchy is used in most design disciplines to manage design complexity and promote reuse of subdesigns.

An example: the classical electronic design process involves 1) the creation of electrical schematic, 2) the creation of a layout diagram, and 3) finally the generation of the data needed for production. Contemporary electronic design processes can also include various other views such as high-level functional descriptions of the design (e.g., VHDL for digital design), simulated waveforms generated to validate the design and test patterns needed for product testing during manufacturing.

In our 5-D model, the view concept is inseparably linked to the *development process steps* by which the views are derived from other views. Such steps can range from fully automatic transformations (e.g., compilation, format conversion, simulation) to steps in which the input views are 'manually' transformed into derived views.

Thus the view concept is closely related to the semantics of *derivation* and (intended) *equivalence*.[2] Some common ways of representing which views belong together are shown in Fig. 2: 1) corresponding views are sometimes grouped into a directory (common in electrical design), 2) corresponding views can have related file names (common in software), and 3) one can directly log the design activities by which the various design views were derived from one another (used in advanced CAD Frameworks).

The view dimension of Design Data Management is especially important when the transformations needed to generate the views cannot be done fully automatically: regeneration of a missing view is costly and error-prone.

Examples: in classical software development processes, only the source view is carefully managed because compilation and linking can be readily repeated. In electrical design, on the other hand, there are multiple views which contain unique information (e.g., schematics versus layout) whereby some of the views (notably simulation results) require lengthy computations to recreate.

[2]By "(intended) equivalence," we mean that the views correspond to the same design. Although the views contain overlapping information, this does not necessarily mean that one view can be derived from the other by a transformation (= both views contain the same information). The equivalence is "intended" rather than guaranteed simply because a derivation step may not have been carried out perfectly. See also the Status dimension.

From a data management perspective, the problem with having multiple views of a design is knowing for certain which views belong together and which don't. If this information is lost or unreliable, the views themselves loose their value. The related data management problem of identifying a view's type and detailed format can be solved by simply recording this information in a more or less standardized way (e.g., an electrical netlist in EDIF 2.0.0 format).

C. The Hierarchy Dimension

In the previous subsection, we noted that complex design processes can be made more manageable by using multiple steps, thus introducing multiple design views. The second commonly used technique is to decompose a design into smaller parts. This process can be repeated until the activities and their deliverables have become manageable. From a data management perspective, this means that the design data has a hierarchical structure.

Hierarchical design is encountered in most design disciplines: complex electrical schematics contain blocks ('cells' and 'symbols') whose internal structure is recorded in a separate schematic. Mechanical designs are created as a hierarchy of assemblies, subassemblies, and parts. In standard software design, decomposition (into modules, objects, and/or functions) is the main weapon one has to combat complexity because software developers rely primarily on a single view of their design.

Fig. 3 shows two equivalent paradigms commonly used to represent design hierarchy: 1) a representation in which the nested physical structure is shown directly, and 2) a more abstract graph[3] representing the design's decomposition.

It is worth noting that design decomposition has an additional benefit: if a design component has a well defined functionality and interface, it can be *reused* in different parts of the design or across multiple projects. This is a major issue in all design disciplines: increasing productivity and

[3]The reuse of components (see next paragraph) means that such structures are formally directed acyclic graphs rather than trees. If a design consists of multiple identical subdesigns, these subdesigns are the same 'node' from an information management perspective.

reliability through the reuse of standard physical modules or through the reuse of abstract designs (e.g., macrocells in IC design).

D. The Status Dimension

In a design environment, the information which is stable, consolidated, and proven is treated differently from information which is tentative, untested, and possibly incorrect. In a sense, the primary goals of a design process are to provide all the required product-describing information (completeness), but also to ensure that the information is fit for use (quality): the design should be consistent, satisfy the basic requirements (e.g., functionality, reliability, safety), and be an intelligent tradeoff between the various other design goals (e.g., manufacturing cost, ease of use, development cost, environmental aspects, time-to-market).

The status dimension of design corresponds to the organizational procedures used to maximize the likelihood that the design is satisfactory. Familiar examples of such procedures are the introduction of verification steps (e.g., electrical simulation) and validation steps (e.g., field testing) and approval procedures (releasing). When design information passes such a step, this results in a change in *status*. The change in status does not correspond to any change in the information itself; it only represents the fact that the organization has decided that the information meets certain requirements.

Note that status tracking is used to determine what can or should be done with the design. The most familiar example is the design status change which corresponds to the decision that the design can be taken into production and shipped to customers.

Status changes are also used to control the subprocesses within the development organization itself. Some examples:

- A design may get a different status if the development team feels that the design is good enough to be submitted to an internal quality assurance department or can be used as input for a different part of the organization.
- An IC design goes through one or more status changes before a costly production run is made.
- Software designs go through alpha and beta testing (testing by close partners) before the product is released to the actual market.

Fig. 4 shows two common paradigms used for representing design status changes. Model (a) shows an example of a state transition diagram where the arrows indicate the possible transitions between states. Model (b) shows a paradigm known as workspaces or environments where the arrows also indicate possible transitions.[4] Characteristic properties of the workspace paradigm are that the levels are related to the way in which the project is organized (e.g., "private," "project," and "public" levels) and the direct link between an item's status level and the persons in the organization for which it is visible or accessible. Thus items

[4]Note that in such diagrams the potential transitions down to lower levels are generally not shown.

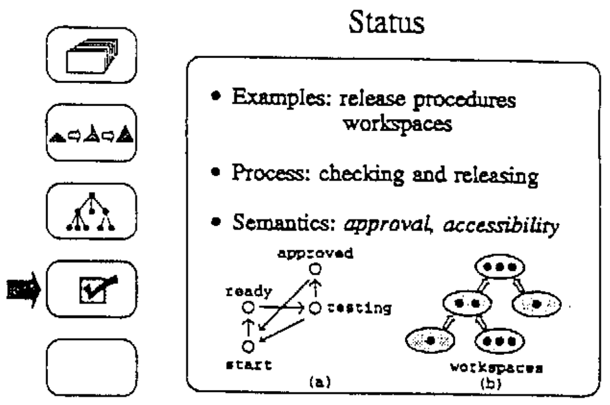

Status
• Examples: release procedures workspaces
• Process: checking and releasing
• Semantics: approval, accessibility
approved
ready
testing
start
(a)
workspaces
(b)

Fig. 4. Design status is an indicator of the degree of maturity of design data and plays an important role in organizational procedures.

at the private level are only available for the individual developers who created these items, while items at the project level are available to all members of the project team.

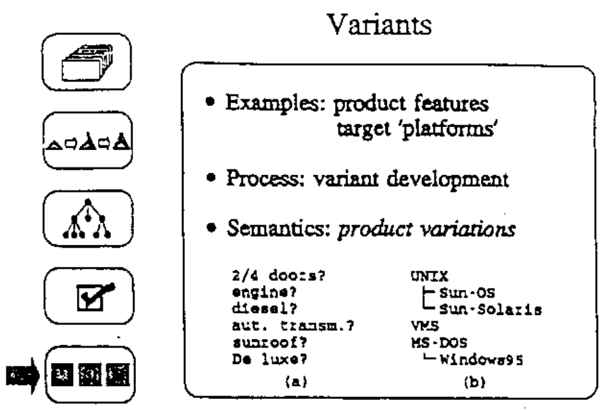
Apart from the organizational importance of the status dimension, it can play a major factor in the design of a data management system if a change in status results in a change in data visibility for certain groups of users. Example: in some systems, the data 'stored' in a designer's private workspace is not visible or accessible to others in the same project. When this data is promoted to a higher level, it suddenly becomes available for use. Although this functionality is in itself reasonable, the workspace paradigm makes the resulting behavior of the supporting data management system significantly more complex.

E. The Variants Dimension

The fifth dimension of Design Data Management occurs in cases where more than one variant of a particular product needs to be designed. The issue here is that these different variants are to a large degree similar, but not identical. This leads to the need to explicitly manage their commonality as well as their differences. Some examples:

- Most software packages which are not specifically developed for one particular customer typically have several variants due to the diversity of the target environments in which they must operate: different operating systems and different user interface standards.
- Products like automobiles are manufactured in different variants due to the required commercial diversity (see Fig. 5). In fact, a car owner will seldom see a car on the road which is identical in all respects.
- Consumer products like television sets are also manufactured in many different variants because of varying price and performance levels, but also because of regional technical differences (e.g., in broadcasting standards).

The approaches taken to handle design variants vary widely per design discipline: in software, multiple "target platforms" are a fact of life while in integrated circuit design, a new design variant is often regarded simply as a new


Variants

- Examples: product features
  target 'platforms'
- Process: variant development
- Semantics: *product variations*

```
2/4 doors?          UNIX
engine?              ├ Sun-OS
diesel?              └ Sun-Solaris
aut. transm.?       VMS
sunroof?            MS-DOS
De luxe?             └ Windows95
    (a)                  (b)
```

Fig. 5. Design variants are important if a product is a member of a family of similar products.

design project in which data and expertise generated in a previous project can be reused. It is our impression that this dimension is the least understood of the five, and is currently gaining momentum.

It is noted that variants and versions are often confused with each other. Sometimes people use the term "version" to represent a product variant, as in "the version of MS Word for Windows 95" or "a CMOS version of the 8051 processor." We will use the word "version" when we refer to design evolution, and "variant" when we refer to the development of families of related products. The orthogonality of the two concepts from mathematical perspective has been pointed out by Wedekind [1].

III. Dimension Mania?

Now that we have finished introducing the five dimensions, it is worth reflecting on whether it is indeed justified to call these five items 'dimensions,' or whether this is actually only an arbitrary list of issues related to Design Data Management which can be arbitrarily abridged or extended.

First some observations:

- The five items are relevant for any particular design discipline—at least we have found this to be the case for the design disciplines encountered within Philips: mechanical design, circuit board design, IC design, software design, system design, and even multimedia "contents" design. Although the relative importance of items differs between design disciplines, the items still appear to be meaningful in each of these areas.
- The items describe the basics of the design process, rather than aspects of a particular support tool or support technology.
- An analysis of a design process or its data is oversimplified unless one looks at several of these dimensions simultaneously. Truly meaningful statements about a design process or method for managing design data must involve two or more dimensions. This is analogous to explaining where to find a room in a building: one needs to know the location in all dimensions (which floor, which hallway, which door; see Appendix C).
- One can obviously wonder whether there will be new dimensions added to the list in the future. All we can say is that the number of items on our list has been stable for several years, so that we would be surprised if we suddenly came across a handful of new design issues of a comparably fundamental nature.

In summary, the term "dimension" stresses the orthogonality of these five design data management aspects and emphasizes the complexity of real world problems which typically involve three or more key dimensions. A comparison of the nature of these dimensions to the familiar geometrical dimensions can be found in Appendix C, "Navigating the five dimensions." Appendix B, "Information modeling for data management," deals with a formal method for representing multidimensional design data management solutions.

The remainder of this article deals with the nature of the interaction between dimensions and should help the reader understand how the 5-D model can be applied in analyzing design data management problems.

IV. 2-D Data Management Models

We will devote this section to illustrating the nature of the *interaction* between the dimensions. An underlying message behind all these examples is that, although each dimension taken separately may seem trivial from a conceptual point of view, the interaction between the dimensions makes real-world Design Data Management and Design Process Modeling nontrivial.

In general, in situations where one must deal with two or more dimensions, one can select from multiple alternative solutions. Each of these solutions has its own characteristics and not every solution is equally suited for a particular design process.

A. Versions and Hierarchy

Consider what happens if one has a hierarchical design (see Fig. 3) and needs to provide versioning (Fig. 1). To simplify the discussion, we will call the various levels assemblies (top), subassemblies (intermediate), and components (bottom).

As soon as one starts keeping track of versions of assemblies and subassemblies, one is confronted with the fact that a new version of a (sub)assembly may have a different composition than its predecessor. This is in fact a common phenomenon: designs can be split up as they evolve, components which were initially left out need to be added, and initially chosen elements can be replaced by other elements. There are two models commonly used to record hierarchies of versioned items.

In Fig. 6(a), the small circles are item versions which in turn consist of versions of lower level items. This approach implies that a particular version of an assembly is *static* in the sense that its composition is fixed to the last detail. This is because the exact versions of all its subassemblies are fixed and the exact versions of all components within these subassemblies are also fixed. Consequently, if one

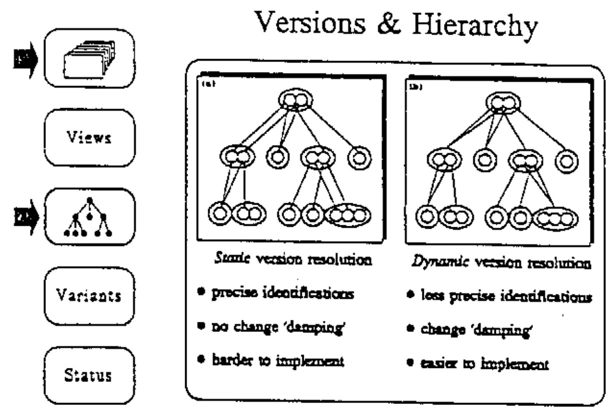


Fig. 6. To manage versioned hierarchical designs, one can choose between keeping track of 'static' or 'dynamic' design hierarchies.

component in the design gets updated, a new version of the corresponding subassembly must be introduced which contains the updated component. Similarly a new version of the assembly must be introduced which contains the new version of the subassembly.

Thus an advantage of the *static* hierarchy model is that the components within a particular assembly version are 100% fixed: there is no uncertainty about which components belong in a current or a previous design version. The price one has to pay for this high degree of clarity is that even a very minor change which needs to be incorporated into the design (e.g., a resistor bought from a different supplier) requires the creation of a new version of the total design (e.g., a large system). Although it is possible to largely automate this process, it typically results in numerous versions at the upper levels of the design hierarchy.

Another problem with the approach shown in Fig. 6(a) is that a design must contain references to specific *versions* of its parts. CAD tools do not traditionally have this capability because most operating systems[5] simply do not support versions of files. Thus static hierarchies require significant effort to implement because one has to 'work against' the operating system.

Fig. 6(b) shows a subtly different model for representing hierarchies of versioned objects. Here the assembly versions do not contain a reference to one specific version of a subassembly, but instead refer to the subassembly *without* reference to a particular version.

In order to fill in the missing details, a rule or algorithm is needed to determine which versions of the subassemblies to use when one needs access to a particular assembly version and all its subassemblies and components. This rule is normally: "Use the most recent item version when one needs to get the current version of the assembly." An old version of the assembly can be retrieved by making intelligent guesses based on version creation dates.

The advantage of this *dynamic* hierarchy model is that minor changes can be made to lower levels in the hierarchy without affecting the upper levels. The disadvantage of this approach is that if one needs to know the exact composition of an old version of a design, one can only give an approximate answer. This can be a major disadvantage if one has strict traceability requirements in an archiving system.

It can give even greater problems in a work-in-progress environment: if a user is debugging version N of design X, one can get nasty surprises. This happens when another user makes a change at a lower level, thereby changing the actual contents of version N of design X without changing its identification. This can be a serious problem when testing software or when simulating complex electrical designs. Consequently IC designers, for example, often prefer static hierarchies in a multiuser CAD Framework: a design will never change without notice due to 'improvement' made by a colleague! It is worth noting that most commercial CAD Frameworks currently still provide dynamic hierarchies.

Some organizations (including Philips) use a subtle variation of dynamic hierarchies to manage consolidated[6] product information. In this variation (known as Technical Product Documentation or TPD within Philips), a two-level item versioning scheme is used. When an item at the bottom of the hierarchy tree is updated, and must therefore get a new identification, the designer must indicate whether the change is relevant for higher levels in the hierarchy. If relevant, the high-level part of the changed item's version identification changes, effectively creating a new item. If it is not relevant for upper levels, the item simply gets a new low-level version identification which is transparent to higher levels in the product structure. The advantage of this technique is that it allows changes to propagate up through the hierarchy tree for precisely the amount of levels which are relevant. Example: a new version of a read-only memory (ROM) chip may be relevant for the developers of the circuit board for which it is intended, but may not change the external specifications of the circuit board and thus will not affect the system design level.

It is worth noting that the interaction between versioning and hierarchy also occurs in manufacturing control systems (e.g., MRP II logistics systems). Versioning and hierarchy are, respectively, associated with "effectivity dates" and "Bills of Material." Such systems use implementations which are functionally similar to the static hierarchy model because their stock control algorithms require precise predictions about which components will be used on a certain date.

In the various design disciplines, it is especially the ECAD area where the interaction of versions and hierarchy is considered very important. The difference between the static and dynamic hierarchy models is stressed in [2] and [3] (and in earlier papers by Katz as well). CAD Frameworks that support the static hierarchy model are e.g., Powerframe [4], Nelsis [5], and JESSI COMMON FRAMEWORK [6]. Cadence's Design Framework II [7], and several other frameworks from CAD vendors, support the dynamic hierarchy model.

[5]Digital's VAX/VMS is the only major operating system with *built-in* support for versioning.

[6]Note that designers do not work directly on copies of the data in the archiving system. In this way the "unexpected change" problem of dynamic hierarchies is avoided.

Versions & Views - 2

Hierarchy

Variants

Status

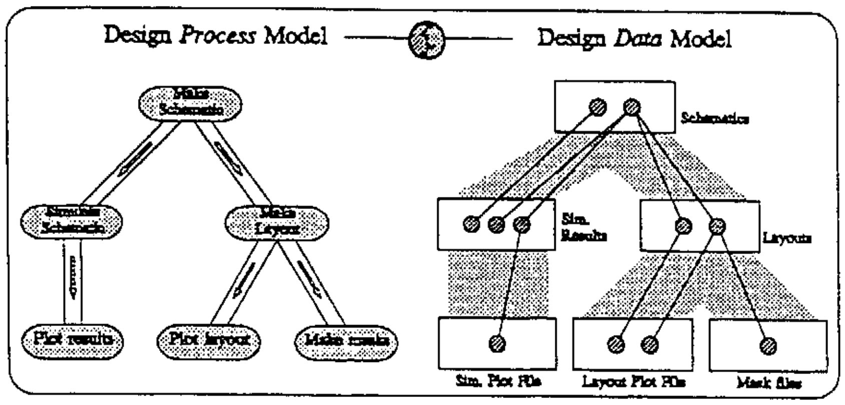


Fig. 10. In the *Roadmap* model, derivation relationships play a major role in organizing and accessing multiview versioned design data.

development projects [6], it has been proposed to combine the level-by-level and nonisomorphic hierarchies models in which the former is primarily used for the upper levels (system design) and the latter is used for the lower levels (detailed design). It is debatable whether this level of end-user complexity is desirable.

The nonisomorphic hierarchies model is supported by most CAD Frameworks, like Nelsis [5] and Cadence's Design Framework II [7]. Also, papers by Katz [3] stress that this model is preferred when managing IC design data.

C. Versions and Views

Fig. 9 shows a simple electronic design process involving three different views of a design. A schematic view is used as the basis for the layout view and the layout view is used to automatically generate the mask files needed for manufacturing.

On the right-hand side of Fig. 9, actual data sets are shown. Thus, for example, the schematic view of design foobar exists in three versions and the layout view exists in four versions. The thin arrows indicate which layout versions correspond to which schematic versions. As in Fig. 2(c), these arrows thus correspond to derivation relationships or (intended) equivalence.

In this model, what we call the *data-centric* model, every derived view should belong to at most one source view. This assumes that every step in the design process requires only one input data set. The model can be readily extended for design steps which require multiple inputs or which create multiple outputs. Example: every mask version should belong to one layout version and every layout version should belong to one schematic version. On the other hand, each view version can have an arbitrary number of derived versions.

The purpose of these derivation relationships is to indicate that a derived view was meant to be equivalent to an existing view. Sometimes the derivation is performed fully automatically. Sometimes the derivation is done more or less manually. Sometimes a small change in the source file means that the previous derived file is ignored and a new derived file is made (comparable to compilation of software). Sometimes a small change in the source file means editing[7] the derived file to accommodate the change.

The data-centric model for combining versions and multiple views is supported in most current CAD Frameworks [4], [9]. It is important to realize that, although shown in Fig. 9, the information about the design process is implicit from the perspective of the CAD Framework. Either the framework does not know about the structure of the design process which it is supporting, or it does not actively use this information to achieve its functionality.

An alternative, and in many cases superior approach to handling versions and views called the *Roadmap* model [10] is shown in Fig. 10. As in Fig. 9, the rounded boxes represent design process steps, and their interconnections represent the flow of data between these steps.

Unlike the previous model, a framework incorporating the *Roadmap* model is explicitly aware of design process "road maps" or "flows" which designers are executing. Each design step thus results in a different design view type—even if two different design steps happen to produce information of the same type and using the same storage format. This allows the CAD Framework to know, for example, that an electrical netlist derived from a schematic

[7]Although this editing is not shown in the diagram, information about the edit history could be recorded in the system.We recommend distinguishing between transformation information ('derivation links') and modification information ('editing links') because the two have different semantics and are governed by different rules.

addition, there is no strong tradition of hierarchical design in managing software: a product is built using files and what is inside the files is unknown to the configuration management system. Consequently, software configuration management systems tend to concentrate on the Version and Status dimensions. A small (and possibly growing) number of systems also provide some support for Variants (e.g., for multiplatform products).

- CAD Frameworks for electrical design must support multiple Views and Hierarchical design, because these dimensions are essential for the electrical design process. There has been support for multiple Versions for a few years. Variants are presently not given a high priority. The importance of Status appears to be rapidly increasing. For example, a software system made at Philips Research [13] provides a circuit simulation environment with support for Versions, Hierarchy and Status.
- Electronic archiving systems, or Product Data Management (PDM) systems, must be able to handle multiple Versions (because this is where historical information is maintained) and Hierarchical product structures (because of reusable subsystems). Design Views are often handled in a simple way (e.g., an extra document type attribute) without recording the derivation information, etc. Status is also important because archives play a formal role in product releasing. Support for Variants is only just emerging, partly driven by the recent interest in product variants ("features and options") in manufacturing control systems.

At present, there are virtually no data management models which cover three or more dimensions in an elegant (i.e., easy to use) way. Often the support for a certain dimension is weak or was apparently tacked onto an existing data management tool when the need for the dimension became apparent.

One fundamental difficulty in designing good multidimensional models is that solutions are closely related to the structure of the target design process (see Appendix A). We thus do not believe that it is possible to provide a single multidimensional solution which is well suited for a wide range of disciplines. We therefore predict that a good solution for one application area (e.g., IC design) will be significantly less "natural" for other disciplines (e.g., software development).

Another fundamental difficulty in creating multidimensional solutions is that, in order to achieve good user acceptance, the underlying model must closely match the way designers think about their data. In particular in a multiview environment (e.g., electrical design), designers strongly relate the structure of the design data to the structure of the design process or "flow." This implies that the concepts employed in the user interface of such a design data management system must closely reflect the structure of the designer's specific design process. This in turn means that vendors of e.g., CAD Frameworks must choose between providing a software toolkit which needs considerable tailoring to match the target design process, or must provide a flexible system which can be customized using a formal high-level design process description [10]–[12].

VI. Discussion on Data Management Modeling in Literature

In this section we will place our work in the context of previously published results on data management *modeling*. Although numerous publications have been written which deal with design data management, the majority of these describe a software system intended to solve data management problems in a particular application area. In this overview we will concentrate on those publications which deal with the basic Design Data Management problems as well as those which discuss the problems in a broader context than one application area.

Interestingly, there are only two application areas which have a significant tradition in data management research and literature, viz., software engineering, and electronic (VLSI) CAD. This is not because the other areas "do" less data management or that data management is considered less important in the other areas. It simply means that in the other areas, selecting or designing a software system for data management is considered to be similar to buying or making software for any other task (e.g., selecting a text editor) and is seldom associated with complex, fundamental problems which have, to date, only partly been solved.

Data management has the longest tradition in software engineering. Software configuration management (CM) systems have been around for about 15 years now. A classic book on CM is [14], describing a.o. "configuration management" and version control, which are two of our five data management dimensions.[9] Early systems around 1980, like *make* and *SCCS* more or less cover these two dimensions independently, using a relatively simple model.

A more advanced model is implemented in *DSEE*, which is Apollo's DOMAIN Software Engineering Environment. This system covers versions, configurations, and status, three of our dimensions. The paper of Leblang and Chase [15] is one of the few papers from the software engineering domain which has influenced the electronic CAD domain.

Several recent papers and books [16]–[20] recognize the wide spectrum of Configuration Management. In [20] Fowler states:

> "From the literature there does not at present appear to be a universally agreed formula defining an exact model for configuration management. ... In practice, different models will be developed depending on the specific requirements of a given situation."

This is very similar to one of our own conclusions. One of the first papers on data management modeling in electronic CAD is [2]. The focus of that paper is on

[9] In [14] the term "configuration management" means keeping track of what parts a product is composed of, and thus corresponds to our design hierarchy dimension. Note that, however, hierarchy support in software is generally limited to a single level decomposition of the system into source files.

Table 2 Data Aspects and Process Aspects of the Five Dimensions

Design DATA Aspect	Corresponding Design PROCESS Aspect
1) Design versions	Design iterations
2) Design views	Design steps
3) Design hierarchy	Subprojects
4) Design status	Design approval and releasing
5) Design variants	Development of families of related products

version modeling. Batory and Kim claim that managing versions is relatively complex because, apart from the versions themselves, several other aspects (dimensions, as we call them) play a role. For example, the concept of parameterized versions in [2] deals with the role of versions in dynamic hierarchies.

A tutorial paper on modeling in electronic CAD [3] presents "a version model based on three orthogonal relationships": versioning, design hierarchy, and design views. Furthermore, Katz's ideas on workspaces are a first step to cover the status dimension. Surprisingly Katz considers the entire model "a version model," even though versioning is only one of the three key ingredients. In [3], Katz also includes an extensive comparison with other papers.

Van der Wolf [21] presents a good overview of the information architecture of a CAD Framework. In fact, Van der Wolf recognizes the important interplay between the Versioning, Hierarchy, and Equivalence (between design views) relationships. The book also presents an extensive literature overview.

VII. Conclusions

Design Data Management and its close relative, Design Process Management, are still relatively young fields. There is a notable lack of common terminology and, more significantly, a lack of agreement about what the main issues are and how to tackle them.

In this article, we have presented a model for analysing Design Data Management problems which has been used within Philips for several years. It serves as a tool with which to analyze user requirements, discuss basic alternatives and classify available software support tools. The central ingredient of this reference model is that data management issues involve five orthogonal *dimensions*. The list of dimensions appears to satisfy a wide range of design disciplines and can be expressed in both data management as well as in process management terms. This is highlighted in Table 2.

A product development environment almost always involves more than one of these dimensions simultaneously. In fact, the increasing complexity of design processes necessitates paying attention to increasingly large subsets of this list. An ideal Design Data Management product would cover the dimensions which are most prominent in any particular environment using a simple and consistent model.

When taken separately, the user requirements for each of these dimensions can be readily satisfied using relatively straightforward and well known design data management and design process management techniques. Unfortunately the nature of the interaction between the dimensions is such that combining multiple 1-D solutions (e.g., versions and hierarchy) is never straightforward.

Underestimation of this problem has led to a generation of Design Data Management products whose models and basic principles are typically hard to explain and thus difficult to configure and operate. The problem is larger in CAD Frameworks than in products intended for managing relatively consolidated information (archive-like products). This is because work-in-progress involves tracking more information, with a greater level of detail, and requires a higher level of user friendliness: the system is used by many users on a daily basis.

Although various reasonable 2-D solutions have been devised, each solution incorporates different assumptions about the target development process. In other words, the type of product being developed and the process by which it is designed will largely determine the suitability of a particular Data Management solution or tool.

Unfortunately, because this is a relatively young field, neither the vendors of data management tools nor their customers appear to be sufficiently aware of the alternatives and the tradeoffs. The main use of this 5-D model may thus be to contribute to this awareness by providing a frame of reference. The frame of reference can then be used to express the basic functionality of design data management tools as well as the basic needs of an organization.

Appendix A
Design Process Management and the Five Dimensions

In the main text, we have repeatedly stated that the five dimensions are not only fundamental for Design Data Management but can also serve as a tool to analyze Design Process Management issues (see Table 2). This should not come as a real surprise because *data* forms the final and intermediate deliverables of the design *process* so, for any structural pattern which is truly important for design data, it is presumably worth knowing how that structure is created within the development process. Taking this one step further, one can surmise that any structure which is considered important in the design data can only be truly important if it is a prerequisite for the chosen design process.

Thus, for example, the ubiquitous use of design views to describe design data at different levels of abstraction and different levels of detail is inextricably tied to the fact that the design process is apparently organized into relatively distinct stages of activity. Each stage contributes to a particular type of information needed to fully describe a satisfactory design.

Similarly, when complex products are represented as a hierarchy of interrelated subdesigns, this raises the question of how the process is organized which produces this hierarchy of designs. In the case of a top-down design process (as often practised in software engineering), the design project spawns a hierarchy of subprojects or tasks, each responsible for producing a particular subdesign. In other cases (e.g., the design of a commodity-type computer), many of the subdesigns are created by specialized third-party manufacturers who supply that subsystem to multiple customers. In such cases, the design activities may more closely resemble a collection of concurrent activities whose plans are mutually coordinated by communication and negotiation processes.

As our last 1-D example, consider the organizational implications of having to produce multiple product variants (sometimes collectively known as a "product family" or "product platform"). One methodology would be to develop one particular variant first (e.g., the most complex one) and then to derive the next variant from the original design. An alternative methodology is to first develop a single generic design which can then be use to generate the various design variants in a relatively straightforward way. The generic design basically contains all the information needed to generate the specific design variants. Typical examples of the latter approach can be found in software development (e.g. #ifdef's handled by the C preprocessor) or mechanical design (parametric design).

As in our discussion of Design Data Management, the design process implications of multiple dimensions can be somewhat tricky and an analysis can reveal multiple alternative methodologies which can handle any particular combination of dimensions. Some 2-D descriptions of design processes can be found in Appendix C.

In our experience, classic techniques for modeling (design) processes do not have sufficiently rich semantics to distinguish between such alternative design methodologies. Many notations or languages for modeling design processes focus on decomposing the total process into smaller units of activity and relating these activities to each other (mutual dependencies and data flow) or to a time or milestone axis. In terms of our 5-D frame of reference, these notations are suitable for emphasizing the Views dimension (and sometimes the Status dimension). The notations provide little or no grip on the other three dimensions. Further work is needed to determine whether the five dimensions can serve not only as a checklist of issues which a process model should cover, but also as a basis for new process modeling formalisms. The latter appears to be promising given the close ties between design process activities and their data deliverables.

Appendix B
Information Modeling for Data Management

As illustrated in the main text, one can communicate a particular data management strategy by graphically depicting sample data illustrating the intended structure. Such graphical examples are useful because they can be readily understood by nonspecialists. An alternative technique known as information modeling is available which originated in the field of database design but has since spread to other data intensive disciplines. Information models are basically direct representations of the structure of data. Some information modeling notations are graphical (e.g., the entity/relationship model [22]) while others focus on syntactic representations (e.g., the Express language [23]).

The benefits of using a good information modeling language as compared to "defining by example" are

- Completeness: if something doesn't occur in the example, it still might be legal.
- Overview: an information model is much smaller than a representative set of sample data.
- Unambiguity: the use of a standard notation is less dependent on the interpretations of individuals.

We will illustrate information modeling using a no-frills European modeling language known as Xplain/OTO-D [24]. Xplain is based on earlier work in the US on database abstractions [25]. Xplain differs from other major information modeling languages primarily in that a given structure can generally be modeled in Xplain in only one way. This property ("object relativity") greatly simplifies the analysis and comparison of alternative models because differences in model topology imply true specification differences rather than arbitrary preferences in modeling style.

In Fig. 11, two alternative models for managing hierarchies of versioned designs from Fig. 6 are shown alongside the corresponding information models. In the *static* hierarchy solution (a), the ProductTypeEdition concept is defined as the combination of a ProductType and an EditionCode. By definition, each ProductTypeEdition object thus corresponds to exactly one ProductType object. Each ProductType object can, on the other hand, have an arbitrary number of associated ProductTypeEdition objects. The connections between circles in model (a) are named PartsListLines in the information model. Each PartsListLine object relates exactly one "assembly" ProductTypeEdition to exactly one "component" ProductTypeEdition.

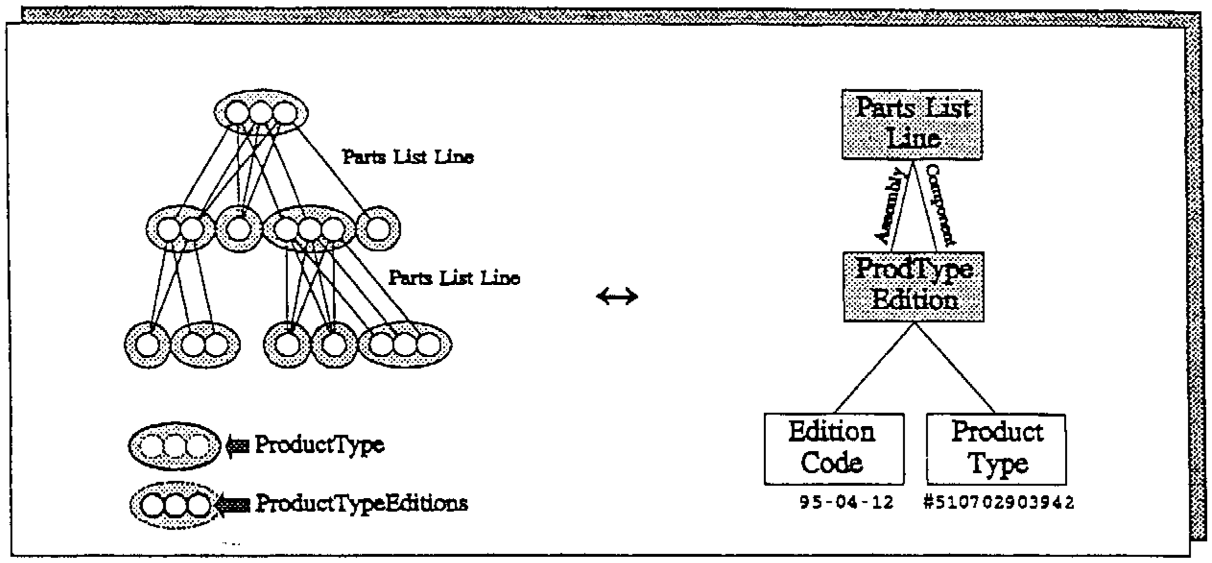
In the *dynamic* hierarchy solution (b), in contrast, each PartsListLine object links one "assembly" ProductTypeEdition to exactly one (unversioned) "component" Product-Type. This difference accounts for the dynamic nature of this solution.

Information models of full-scale design data management solutions typically involve 10–20 key concepts (Xplain object types).

Appendix C
Navigating the Five Dimensions

The five dimensions of Design Data Management can be regarded as the information structures along which one navigates in order to select or browse design data. In any design process, one or more dimensions may be degenerate (e.g., because only a single view of the design is created, because a single product variant is required, or because only the latest version is saved) thereby effectively reducing the number of dimensions through which one must navigate.

In this text box we compare these dimensions of design data to the familiar dimensions of 3-D geometric objects. In particular, we examine the problem of how a person



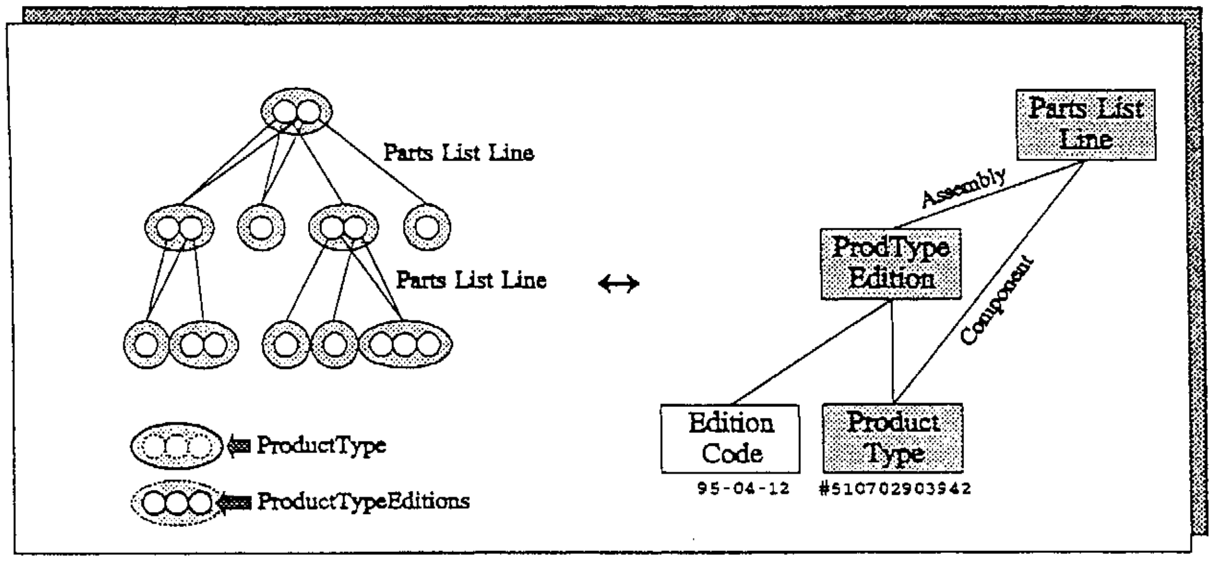


Fig. 11. Two alternative models for managing versioned hierarchical designs (see Fig. 6) with their corresponding format information models.

navigates through a building because this introduces the elements of navigation, human cognition, and constrained degrees of freedom—all of which also apply to collections of design information.

A large building can obviously be regarded as a collection of rooms which happen to be distributed in three dimensions. It is the responsibility of the building's architect to ensure that all rooms in the building can be reached reasonably efficiently at reasonable cost. The way to reach a room obviously depends on the building's architecture (we will ignore the occasional Hollywood stunt man who prefers scaling outer walls or other persons with eccentric navigational behavior). The constraints imposed by the building's architecture imply that a straightforward mathematical representation of the target room's coordinates probably do not conform to the way inhabitants of the building regard the building. Most civilians tend to require and give directions in operational terms ("turn right, take the elevator ...") rather than stating that the destination is 25 m North and 5 m West and at 10 m elevation.

Thus humans tend to use coordinate systems which match the navigational paths which are available to them. Imagine a cross shaped building with three floors, with a single stairway at the intersection of the building's four wings. Navigating to a room in this building involves taking the stairway to the desired floor, selecting the appropriate wing and finding the appropriate room in that hallway. From an information perspective, the navigation strategy for a building with this architecture corresponds to a decision tree where any room is found via a door in a hallway of a certain floor of the building. The numbering schemes in such building typically reflect this decision process by uniquely identifying rooms in terms of these three components.

Imagine now another building with three floors where each of the four hallways has its own stairway and in which you can only move from hallway to hallway at the ground level. In this scenario, navigation involves selecting a wing of the building, followed by a hallway in that wing followed by a door in that hallway.

It is worth noting that buildings with three geometric dimensions do not necessarily have to have three navigational dimensions. A row of separate cross shaped buildings would be 4-D from a navigational perspective, while a cylindrical building would be 2-D.

We now return to the problem of organized design data in such a way that the data elements can be searched by navigating along semantically meaningful paths. Assume a software development process in which a generic file is maintained which contains all the information needed to automatically create several specific variants of this file. In this case, a navigation path to a specific file would be "a variant of a version of the design." The role of version-selection in this "process architecture" corresponds to the role of the central staircase in our building analogy. The concept of "version of the design" corresponds to the concept of "floor" in a building.

In an alternative design process, several design variants evolve independently over time. The navigation path would involve identifying a specific file as "a version of a variant of the design." This is a different "process architecture" altogether. The previous concept of "version of design" has become meaningless. This is analogous to a building with stacked rooms with an internal staircase per room and where the stacks and even the rooms themselves may have a different height: the "floor" concept has become meaningless and has been replaced by the more fitting "room stack" or "tower" and "floor-in-tower" concepts.

Another pair of examples would be a design process which supports "versions of a view of a design" (implying that the views evolve more or less independently) versus an alternative design process which supports "views of a version of a design."

These examples can also be extended to cover more than two dimensions (giving rise to a larger number of alternative processes): in a particular type of design process, a data element may be identified as "a part of a variant of a view of version of a design."

As this way of formulating the navigation process resembles the use of nested operators, one can be led to conclude that apparently these 5-D operators do no commute: Version(Variant(design)) is not equivalent to Variant(Version(design)). This statement about the 5-D operators is probably of limited value, however, because the basic problem is not that these two expressions return different results. The fundamental problem is that, for one type of design process Variant(design) is meaningful and Version(design) is meaningless, and vice versa for another type of design process. In our analogy about navigating through buildings, this corresponds to a statement that says that in some architectures the concept of floors is meaningless. In a castle with a number of towers, the concept of floors which span the entire building is not very intuitive, and fails altogether if the height of the rooms in the various towers do not match.

This brings us to a key observation about the nature of the five dimensions themselves: in multidimensional problems, the five dimensions, when regarded separately, do not have a formal definition. For a given design process (and thus a given design data management model), however, a chain of concepts do have meaningful definitions. If we assume that Part(Variant(View(Version(design)))) is meaningful, than also the intermediate concepts Variant(View(Version(design))), View(Version(design)) and Version(design) are meaningful. As, in practice, people tend to give concepts names like "view" or "design view" rather than "View(Version(design)))," this implies that terms like "design view" tend to have different meanings in different design processes. Thus in multidimensional situations, terms like "design view" cannot have an exact definition.

Note that this doesn't mean that the concept of "design view" is meaningless. It just means that the precise meaning of "design view" must be defined in terms of how it interacts with any other relevant dimensions and that this interaction depends on the design process one is examining.

As a final observation on this topic, we would like to point out that the meaning of terms like "design view" can, within a given design process context, be formally defined using information modeling techniques. Prerequisite for this is that a formal language for information modeling is used (see Appendix B) and that the data schemas cover the interaction between all relevant dimensions. Readers with a formal information modeling background will undoubtedly have concluded for themselves that the navigation processes described in the text box directly correspond to the concept of queries in databases and information modeling. The navigational paths then correspond to a query path which connects a notion of design (e.g., called "design project") to a notion representing an elementary unit of design data as managed by the data management system (e.g., called "file"). The "design process architectures" along which one navigates correspond to the information models described in Appendix B, or data schemas [26] as they are sometimes more formally known.

ACKNOWLEDGMENT

The authors would like to thank Ger van den Broek, Hans Kok, René van Loon, Willem Reijntjens, Henk te Sligte, and Bram Riemens (all at Philips Research Laboratories Eindhoven), and Casper Baghuis (University of Twente) for valuable discussions on this topic.

Furthermore, the authors would like to thank their European colleagues in the JESSI-COMMON-FRAME project for providing an inspiring environment for parts of this work.

REFERENCES

[1] H. Wedekind, "Are the terms 'version' and 'variant' orthogonal to one another?," *SIGMOD Rec.*, vol. 23, no. 4, pp. 3–7, Dec. 1994.

[2] D. S. Batory and W. Kim, "Modeling concepts for VLSI CAD objects," *ACM Trans. Database Syst.*, vol. 10, no. 3, pp. 322–346, Sept. 1985.

[3] R. H. Katz, "Toward a unified framework for version modeling in engineering databases," *ACM Comput. Surveys*, vol. 22, no. 4, pp. 375–408, Dec. 1990.
[4] J. Brouwers and M. Gray, "Integrating the electronic design process," *VLSI Syst. Des.*, pp. 38–47, June 1987.
[5] P. van der Wolf and T. G. R. van Leuken, "Object type oriented data modeling for VLSI data management," in *Proc. 25th ACM/IEEE Design Automat. Conf.*, Anaheim, CA, June 1988, pp. 351–356.
[6] D. C. Liebisch and A. Jain, "JESSI COMMON FRAMEWORK design management–The means to configuration and execution of the design process," in *Proc. EURO-DAC 92*, 1992, Hamburg, Germany, pp. 552–557.
[7] L-C. Liu, P-C. Wu, and C-H. Wu, "Design data management in a CAD framework environment," in *Proc. 27th ACM/IEEE Design Automat. Conf.*, 1990, pp. 156–161.
[8] Sherpa Corp., "Sherpa DMS (R) Command Interface Reference Manual (c)," 1987.
[9] S. Banks, C. Bunting, R. Edwards, L. Fleming, and P. Hackett, "A configuration management system in a data management framework," in *Proc. 28th ACM/IEEE Design Automat. Conf.*, 1991, pp. 699–703.
[10] P. van den Hamer and M. A. Treffers, "A data flow based architecture for CAD frameworks," in *Proc. 8th IEEE/ACM Int. Conf. on CAD*, 1990, pp. 482–485.
[11] K.O. ten Bosch, P. Bingley, and P. van der Wolf, "Design flow management in the NELSIS CAD framework," in *Proc. 28th ACM/IEEE Design Automat. Conf.*, 1991, pp. 711–716.
[12] J. B. Brockman and S. W. Director, "The Hercules CAD task management system," in *Proc. 9th IEEE/ACM Int. Conf. on CAD*, pp. 254–257, 1991.
[13] W. J. M. Reijntjens, A. K. Riemens, F. G. M. van Heuven, and J. T. M. Kok, "Teamwork design management concepts and implementation in Cadence design framework II," Philips Res. Labs. Eindhoven, private communications, 1993.
[14] E. H. Bersoff, V. D. Henderson, and S. G. Siegel, *Software Configuration Management*. Englewood Cliffs, NJ: Prentice-Hall, 1980.
[15] D. B. Leblang and R. P. Chase, Jr., "Computer-aided software engineering in a distributed environment," *SIGPLAN Not. (ACM)* vol. 19, no. 5, pp. 104–112, May 1984.
[16] P. H. Feiler, "Configuration management models in commercial environments," Tech. Rep. CMU/SEI-91-TR-7, Mar. 1991.
[17] S. A. Dart, "The past, present, and future of configuration management," Tech. Rep. CMU/SEI-92-TR-8, July 1992.
[18] J. Papillon and J. Dick, "Configuration management," in *Systems Engineering, Principles and Practice of Computer-Based Systems Engineering*, B. Thome, Ed. New York: Wiley, 1993, pp. 97–134.
[19] D. Schefström and G. van den Broek, Eds., "Configuration management dynamics," in *Tool Integration, Environments and Frameworks*. New York: Wiley, 1993, pp. 58–66.
[20] A. Fowler, "Models and applications of configuration management," *OMEGA Int. J. Manage. Sci.*, vol. 21, no. 4, pp. 425–431, 1993.
[21] P. van der Wolf, *CAD Frameworks: Principles and Architecture*. Amsterdam: Kluwer, Sept. 1994.
[22] P. P.-S. Chen, "The entity-relationship model—Toward a unified view of data," *ACM TODS*, vol. 1, no. 1, pp. 9–36, Mar. 1976.
[23] ISO/IEC, "Product data representation and exchange—Part 11: The EXPRESS language reference manual," Geneva, Switzerland, Aug. 1992.
[24] J. H. ter Bekke, *Semantic Data Modeling*. London: Prentice-Hall, 1992.
[25] J. M. Smith and D. C. P. Smith, "Database abstractions: Aggregation and generalization," *ACM TODS*, vol. 2, pp. 105–133, 1977.
[26] C. J. Date, *Introduction to Database Systems*, 5th ed. Reading, MA: Addison-Wesley, 1990.

Peter van den Hamer received the Ph.D. degree in applied physics from Delft University of Technology, The Netherlands, in 1987.

In 1987 he joined Philips Electronics and has been working as a consultant and researcher in the area of Design Data Management. He is presently based in the Software sector of Philips Research Laboratories-Eindhoven. His current research interests are information and process modeling, diversity management, data management strategy, and computer-aided kite choreography. He is the author of several publications in the area of CAD Frameworks architectures and holds a patent pertaining to process-based data management.

Kees Lepoeter received the M.Sc. degree in mathematics and computer science from Eindhoven University of Technology, The Netherlands, in 1985.

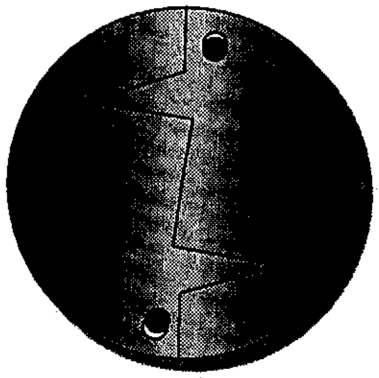
He then joined Philips Electronics and has been working on CAD Frameworks and ECAD data management issues since 1989. He is presently working in the Electronic Design & Tools (ED&T) department of Philips Research Laboratories Eindhoven where he provides design management consultancy for several Philips product divisions. He also coordinates Philips' data management activities within European collaborative projects.

The Philips 5-Dimensional Framework
for modelling design data and design processes

Dr.Ir. Peter van den Hamer (IST)
Ir. Kees Lepoeter (ED&T)
Philips Research Laboratories
Eindhoven, The Netherlands
3-Jun-96

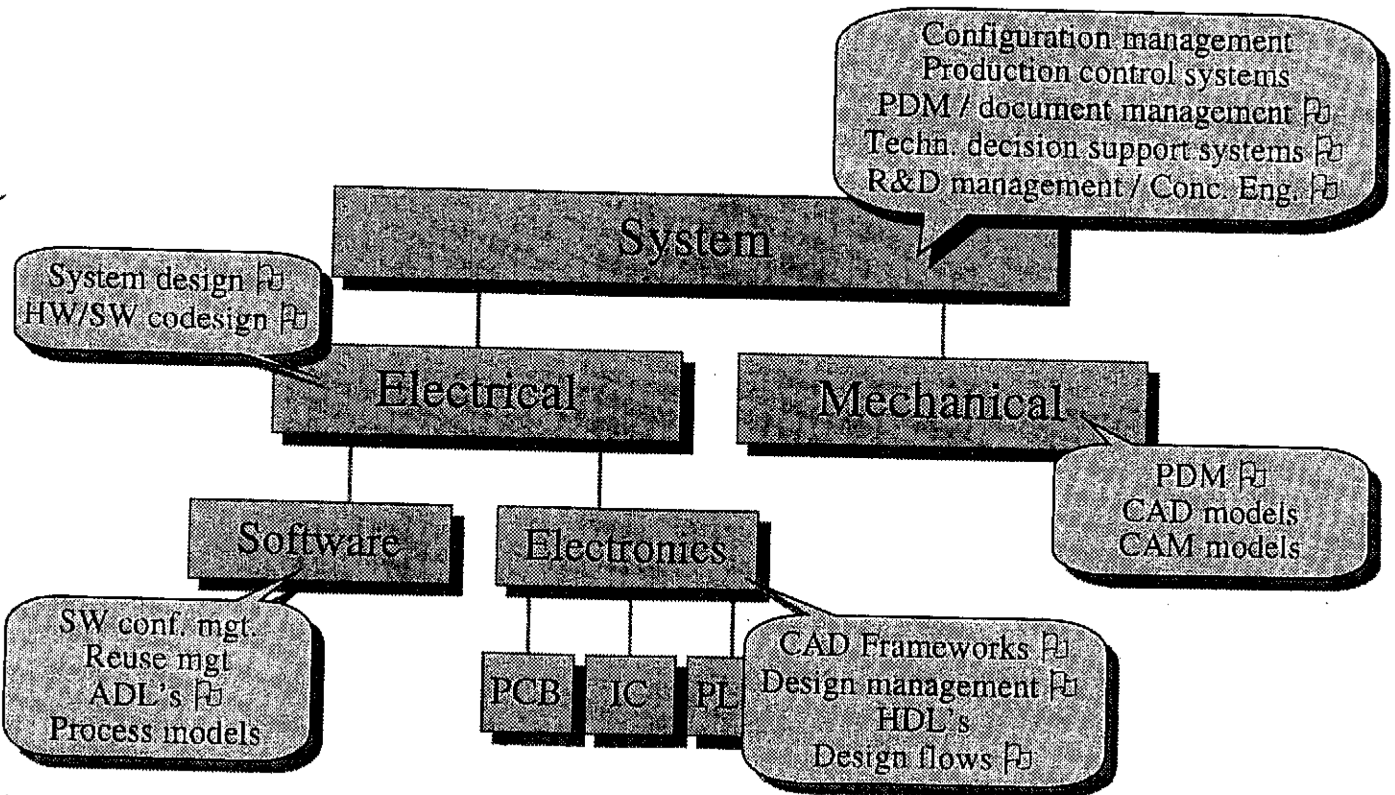
WDK Workshop - Delft
Full paper appeared in
Proceedings of the IEEE
(Vol. 84, No. 1, Jan 1996)

Design Information Management



Design Process Management

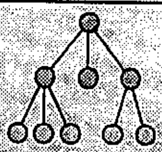
Scope of model: Disciplines and keywords



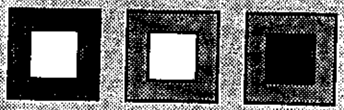
The 5 dimensions of design

1. *Design **Versions*** *Design **iterations***

2. *Design **Views*** *Design **steps***

3. *Design **Hierarchy***  ***Subprojects***

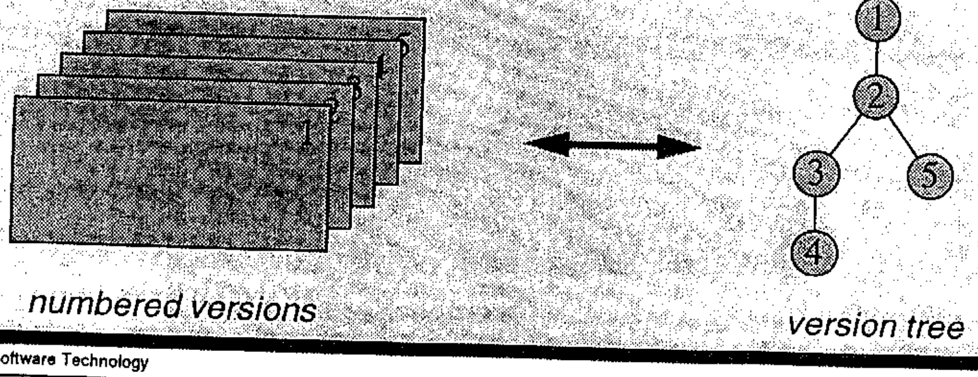
4. *Design **Status*** ***Quality control***

5. *Design **Variants***  ***Product families*** ***Product platforms***

1. *Design **Versions*** *Design iterations*

Basic semantics: design *modification*

Examples of why designs are changed:
- correction of errors (e.g. debugging of software)
- optimization (e.g. chip size reduction)
- changing requirements (e.g. new insight)



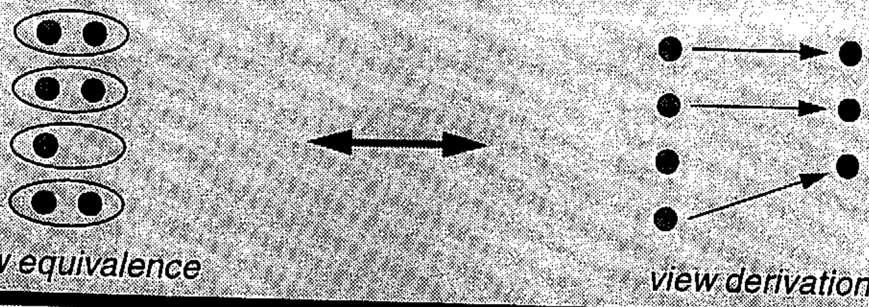
2. Design *Views* *Design steps*

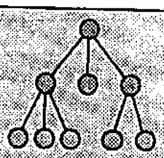
Basic semantics: *equivalence and derivation*

Goals: mgt of design complexity & expertise

Examples:

- electrical schematic ➔ electrical layout
- source code ➔ object code



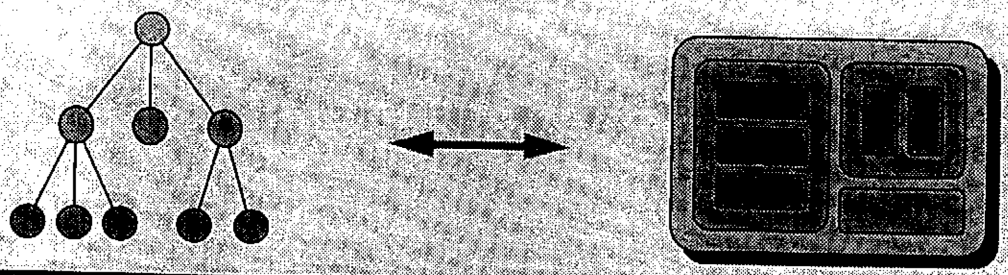
3. Design *Hierarchy*  *Subprojects*

Basic semantics: *consists_of* or *part_of*

Goals: mgt of design complexity & design reuse

Examples:

- assembly ⇔ subassembly
- IC design ⇔ cells
- software ⇔ module

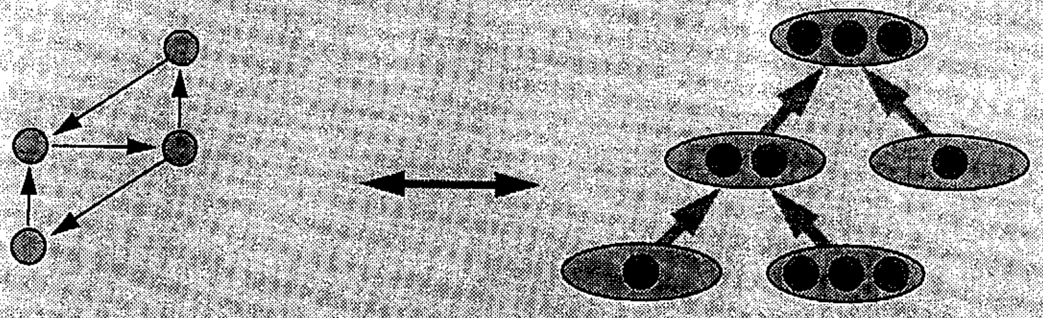


4. *Design **Status*** — *Quality control*

Basic semantics: *approval & accessibility*

Examples:

- release procedures
- workspaces (e.g. CAD or CASE)


state transitions — *workspace hierarchy*

5. *Design **Variants*** — *Product Families* *Product Platforms*

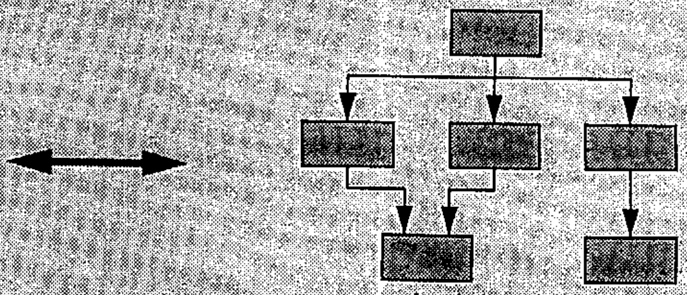
Basic semantics: *required product variations*

Examples:

- feature diversity (e.g. cars)
- market coverage (e.g. price range)
- environment diversity (e.g. software applications)

☐ alarm
☑ sun roof
☑ radio
☑ auto. transmission
☐ spoiler

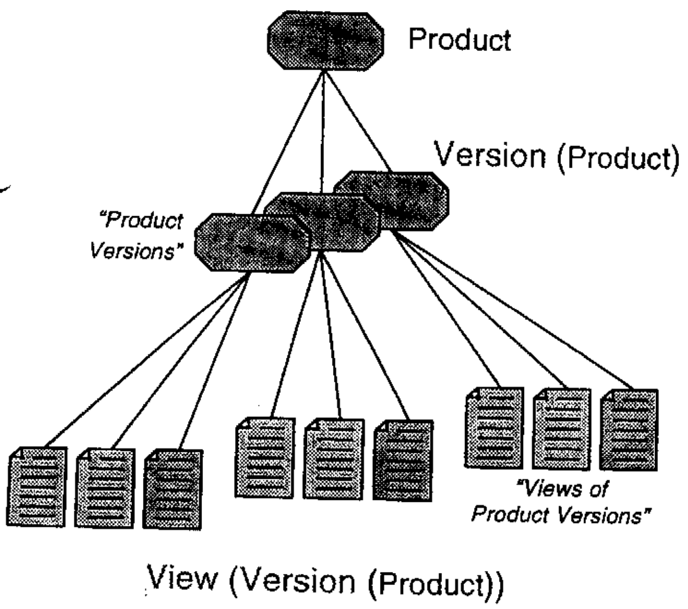
option list


decision 'tree'

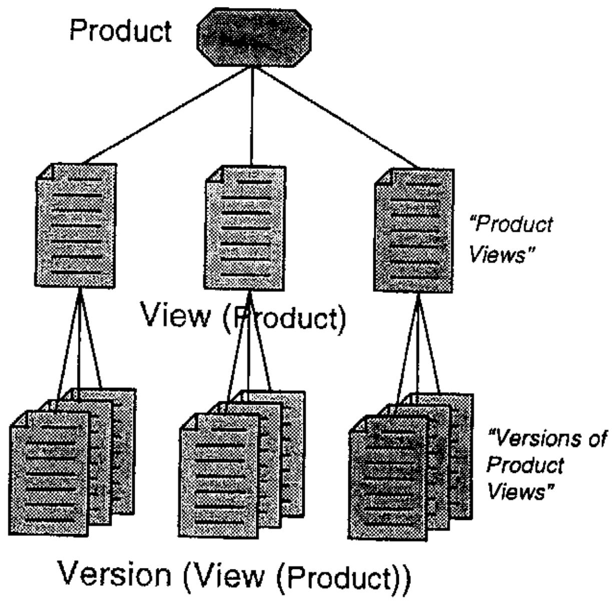
Conclusions - part 1

- The 5 individual dimensions are recognizable in all design disciplines.
- 1-D product models are conceptually trivial, but...
- real-world design processes require multi-D models.
- List of dimensions are based on experience within various disciplines.
- But we cannot prove that there are no more than 5 dimensions, but...
- these 5 items are more than just our list of "important" issues.

Combining Versions & Views

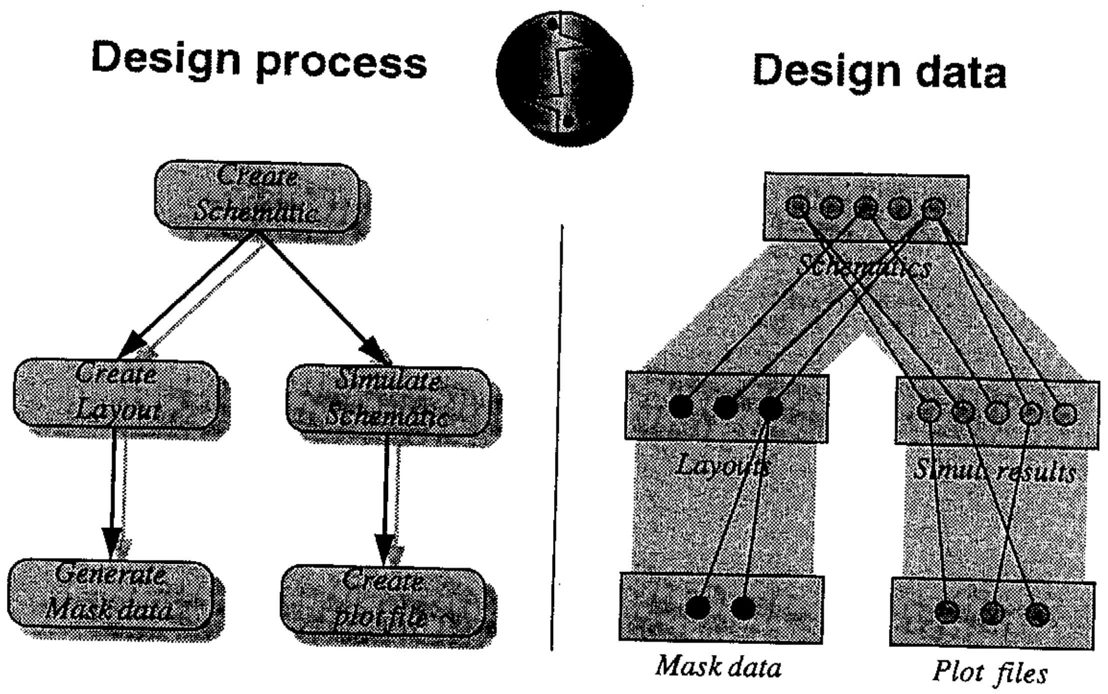


Example: Philips standard for Technical Product Documentation
Characteristic: assumes consistent & stable documentation sets
Disadvantage: requires extra concept to avoid duplication of data



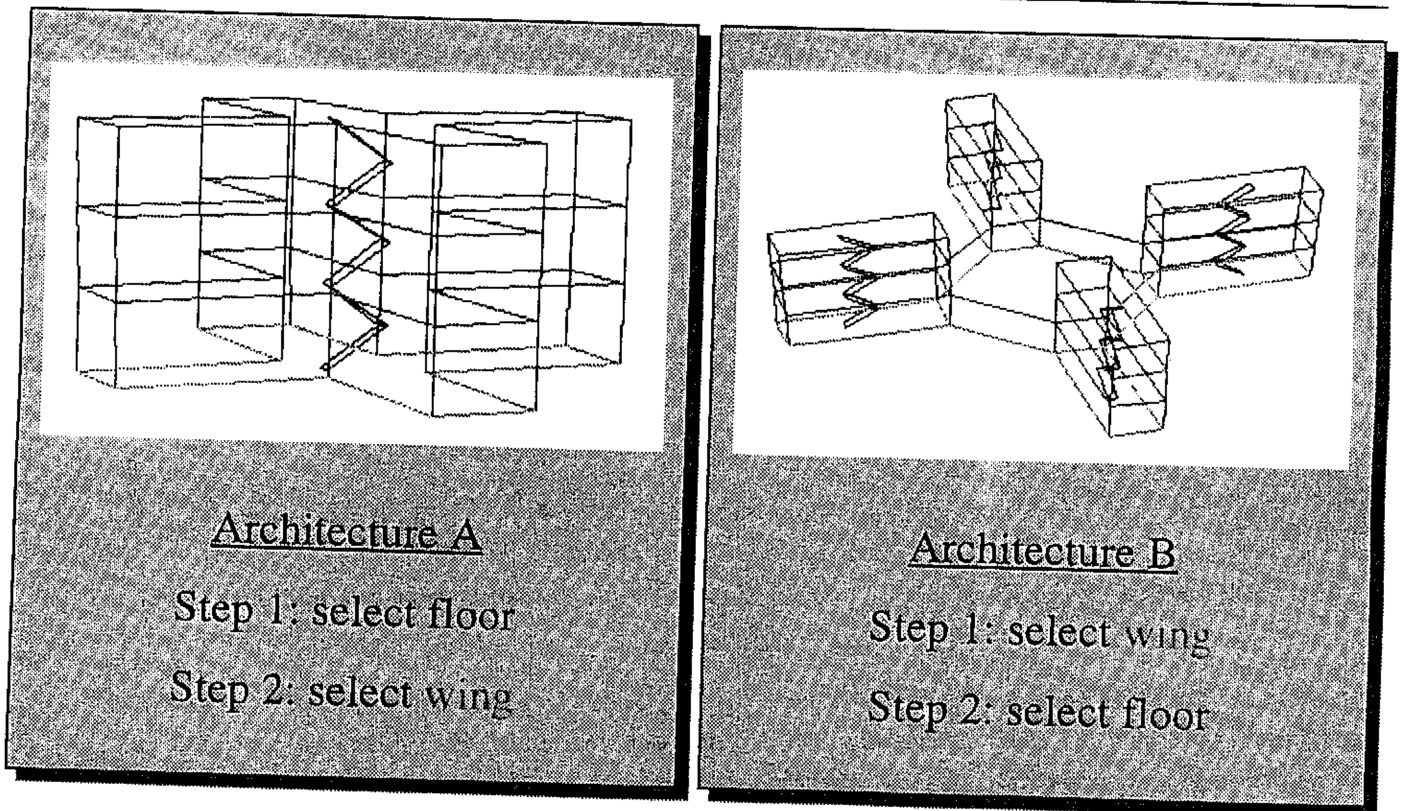
Examples: VMS operating system, Nelsis VLSI design framework
Characteristic: assumes independent view modification processes
Disadvantage: requires extra 'equivalence' relationships

Views & Versions: configurable solution



[*see also Proceedings of ICCAD-90 conf. (Santa Clara) page 482-485*]

Navigational dimensions



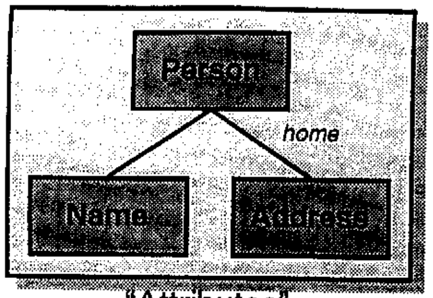
Conclusions - part 2

- 2-D product models are non-trivial.
- Main 2-D models can be found by permutating the 'operators':
  - variant(version(design)) ⇔ version(variant(design))
  - view(part(design)) ⇔ part(view(design))
  - version(view(design)) ⇔ view(version(design))
- These alternatives correspond to fundamentally different processes.
- The "5-D" are navigational dimensions through constrained space.
- Major challenge to create adequate *and* simple 3-D to 5-D solutions

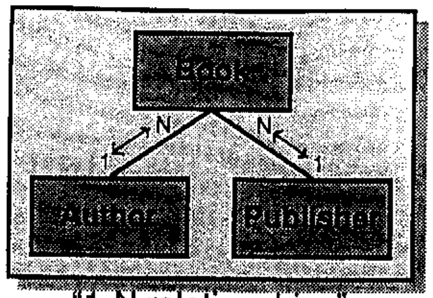
Bonus-1: Uses of the 5-D model

- reference terminology
- insight into high-level structure of design information
- insight into overall structure of design process
- checklist for data management (data modeling) issues
- checklist for process management (process modeling) issues
- basis for catalog of data/process management solutions

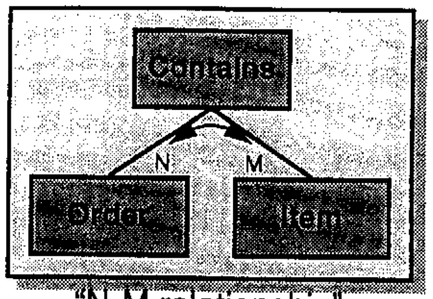
Bonus-2: Information modeling notation



"Attributes"

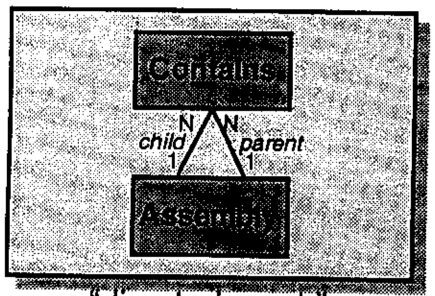


"1-N relationships"

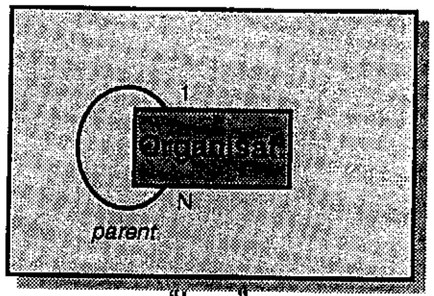


"N-M relationship"

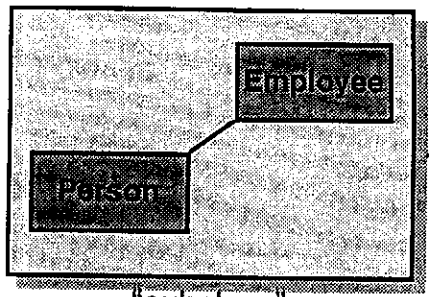
Notation (Smith&Smith/Ter Bekke) only models information which is verifiable.
Black information (above) is part of model. Blue information is not.



"directed graph"



"tree"

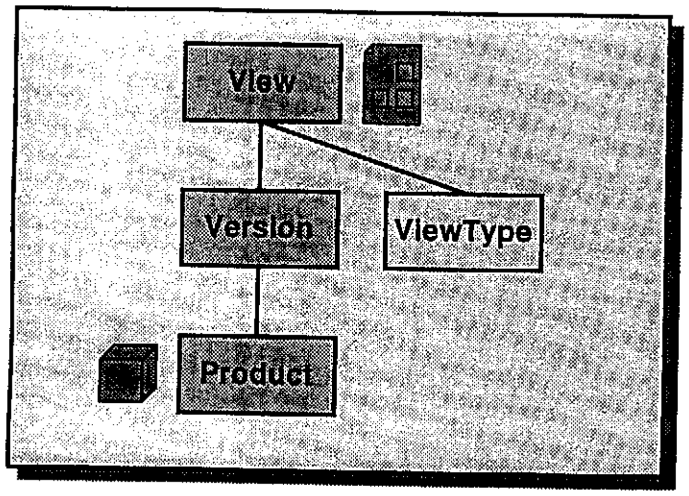


"subclass"

Bonus-3: formal Versions & Views

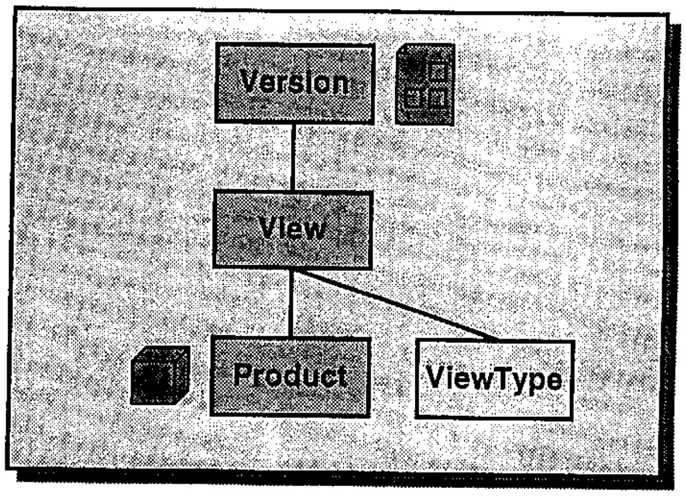
= View (Version ( ))

= Version (View ( ))



Design process regarded as sequence of consistent product releases.

Example: archived consolidated designs



Design process regarded as sequence of independently iterating steps.

Example: IC design management