LAYERING AN ENGINEERING INFORMATION SYSTEM

Gio Wiederhold* Marianne Winslett** * Lt Nicholas Naclerio†
*Stanford University **University of Illinois-Urbana †AFWAL/ETL, Wright-Patterson AFB

Abstract
In this note we define a layering intended to structure the many concepts which arise in the development of an Engineering Information System. The industrial benefits to be gained by such a layering are enumerated.

1. OBJECTIVE

An Engineering Information System (EIS) is a collection of facilities and services that provide support for the design, testing, manufacture, and maintenance of engineering artifacts, and for the managerial and administrative concerns that accompany these tasks. Engineers and their managers engaged in hardware, software, mechanical, and architectural design, or other complex engineering activities involving large amounts of information, will benefit from the availability of EIS services, which we expect to become widespread in the next decade.

The need for EIS services is particularly acute in the digital hardware engineering domain. As more and more circuitry is packed onto VLSI chips, no single engineer can have a full grasp of the the chip design. Instead, the design process is caaried out by multiple engineers working concurrently. Their needs for controlled sharing of information have inspired much recent work in the database community (e.g., [Banerjee 87, Batory 86, Chou 86, Katz 87]). This work includes support for versions and alternatives of objects, configuration management, long transactions, and object-oriented databases.

Current-day chip designers find that no single vendor can supply all the design tools needed. Unfortunatly, tools from different vendors often require different data input formats, and assign different semantics to the same pieces of data. To pass data between incompatible design tools, data format translators must be written. Translation often incurs loss of information [Wiederhold 86]. An EIS can avoid translation by providing base data in one or two standard hardware design representation languages, such as EDIF and VHDL. To attach a new tool should require only writing an interface translator.

With large design projects, managers will require automated means of enforcing design policies and of monitoring the status of designs. Again, an EIS will facilitate these tasks by providing a way for managers to declare design constraints and to specify moments at which the constraints must be validated.

The DoD feels that for the U.S. to remain competitive in the world market for application-specific integrated circuits, the design cost and risks associated with building these circuits must be lowered. In order to do so, design environments are needed that provide seamless interfaces between system specification, chip design, layout, manufacturing, and test. Since it is unlikely that any one vendor will ever be able to provide all of these functions, the DoD feels that a framework of standards is necessary to allow heterogeneous systems to interoperate in a uniform manner. To meet this need, the DoD initiated the Engineering Information System (EIS) program. The goal of the program is to develop and demonstrate a framework of standards for computer-aided engineering tool integration.

As part of the DoD EIS project, one of the subcontractors [Kumar 88] identified thousands of object and function types that are needed to support digital hardware design; this set is intended to define the information needs of hardware design activities. Having so many concepts, operating at many different levels of abstraction, made it hard to get an overall understanding of the EIS. It could hence not serve as an information model, and its consistency could not be verified. In addition, the plethora of concepts made it unclear whether the EIS could be efficiently implemented.

The users of an EIS must be presented with a sufficiently simple model to permit them to carry out their tasks. For a model to be easy to understand, it should be limited to a small (7 ± 2) number of concepts. If the number of necessary concepts is larger, then ease of understanding can be preserved by vertically partitioning the concepts into layers. Our objective in this paper is to *partition* the manifold concepts arising in an EIS, so that the model seen by users can be simple. Section 2 proposes a layering for the concepts encountered in EIS modeling discussions. However, so many concepts populate the EIS world that a horizontal partitioning alone is insufficient for certain layers. To aid users, Section 2 also proposes a vertical partitioning of selected layers into domains of expertise (e.g., layout, logic, other design phases, system administration; or by design tools and methodology, e.g., VHDL or EDIF). This partitioning has been adopted for use in the data modeling activities of the DoD EIS program, in the hope that it will focus future work.

An advantage of a layered model is extensibility. As new concepts are needed, they can be defined at the appropriate layer, in terms of lower-level concepts. Without a layered structure, the information model isjust a grab-bag of objects and functions.

2. EIS SYSTEM LAYERING

The population of EIS users can be subdivided into engineers, managers, and EIS administrators, each with their own concerns. This section proposes a layering of the concepts needed to address these concerns, and Section 3 contains a discussion of this layering.

A Seven-Layer Model

Figure 1 presents a seven-layer division of EIS concepts. Besides layers 0–6, one might also imagine a layer 7 containing the actual design data and versions of user defined objects at a particular instant in time.

Each layer is intended to be *complete* in that objects and functions at that layer can be specified in terms of the other concepts at that layer and at the preceding layer. Thus each layer except the first can be thought of as a view, in the database sense of the term, of the layer below. In order to achieve this completeness, some concepts must appear at multiple layers; for example, data types such as strings will be needed at levels above layer 1. To avoid clutter, the discussion below only mentions a concept at the layer at which the concept first appears.

We will now discuss the individual layers in more detail.

Layer 0: Abstract object model concepts.

We use the Fugue model [Heiler 88a] to provide a minimal set of abstract primitives, which we adopt as layer 0:

0 *Abstract object model concepts.* The primitive abstractions of objects, functions, types, and views, which provide the conceptual basis for layer 1.

1 *Primitive concepts.* Type definitions for basic objects and functions.

2 *Primitive EIS concepts.* Large set of basic EIS-specific objects and functions.

3 *EIS building blocks.* The building blocks to assemble an EIS system.

4 *Domain-specific concepts.* EIS concepts germane to a particular EIS application, such as computer-aided digital hardware design.

5 *Product-specific concepts.* Framework germane to an entire project.

6 *Site-specific concepts.* Framework relevant to a particular EIS product and site.

Figure 1. Seven-layer model for EIS concepts.

1. Objects.
2. Functions.
3. Types.
4. Views.

The totality of *objects* provides the structure and defines the state of an EIS. *Functions* can be applied to objects to yield new objects and create new EIS states. *Types* provide a classification of objects, and *views* partition objects into groupings relevant to a particular design domain, task, or goal. These four categories are not mutually exclusive; for example, Fugue functions are objects, and can be manipulated as such. Similar primitives are seen in other object-oriented meta-languages. References [Heiler 88a,b] provide a detailed description of the Fugue primitives.

Views [Heiler 88a] provide a means of composing objects and functions, where the functions are constrained to be applied to objects within that view. The view provides a constrained scoping mechanism which makes only selected sets of functions and objects available for a particular design task.

Because standard procedural languages already provide the concepts of types, objects, functions, and views, we do not expect that layer 0 will be explicitly implemented. Rather, layer 0 is intended as a conceptual underpinning for the higher layers.

Layer 1: Programming language concepts.
Layer 1 can be defined with the Fugue model; it also corresponds to facilities already available in programming languages. At this layer we find

1. The data types integer, real, boolean, etc.
2. The type constructors array, string, set, list, stream.
3. Schema manipulation functions as: create-relation.
4. Database data types such as tuple, relation;
5. Database functions as: retrieve, insert, delete, modify.

Primitive Fugue object types include strings, integers, booleans, reals, time, etc. Non-primitive objects are composed from primitive objects by the use of *attribute functions*. For example, a `gate` object can be defined with the attribute functions `x-position` and `z-position`, which take instances of, say, integers as their values.

Layer 2: EIS primitives.
Layer 2 contains only those functions specific to the EIS and EIS services, such as the operations specified in early TRW reports on the engineering information model for the DoD EIS project [Kumar 88]. For example:

1. In the user interface area, the basic notions are those of scripts, interaction objects, and non-interactive core objects, and type definitions for these are given in layer 2. A script is a scenario for the interaction of a user with an EIS; an interaction object is an object via which the user interacts with an EIS; and a non-interactive core object is any other type of object that is part of the EIS user interface. Subtypes of these types will appear in layer 2.
2. For a multi-site EIS, this layer contains type definitions for network, host, etc.
3. To allow for the administration of management's policies, this layer gives type definitions for event, control point, and policy. An instance of a policy is a function to be invoked at certain control points. An instance of a control point is a description of a system state; if the system enters that state, then the policies associated with that control point are to be performed. For example, when the user asks to edit a design module, one might wish to test for access rights.
4. The data types for tools, tool adaptors, and database adaptors are defined here. A tool adaptor instance is an encapsulation of a tool that is not fully integrated with the EIS, so that the tool can be invoked from within the EIS in a convenient and consistent manner, without the user having to understand the pecularities of the external tool. A database adaptor performs the same function for an external database management system.

To contrast the purpose of layer 1 with that of layer 2, it is helpful to draw an analogy to a Lisp programming environment: although the basic primitives of Lisp (the equivalent of layer 1) are simply `CAR, CDR, CONS`, and `REPLACA`, standard CommonLisp specifies a large set of functions for direct implementation (the equivalent of layer 2), to improve processing efficiency and to ease application development.

Layer 3: EIS building blocks.
Layer 3 specifies the building blocks used to assemble most of an EIS. At this layer we have interaction objects like menus and icons, database services such as distributed join, and administrative macros like define-configuration. Layer 3 is probably the lowest level interface that anyone would ever implement as an EIS product. Examples of concepts at this layer include

1. In the user interface area, we have the type definitions for button, window, menu, icon, text box (all varieties of interaction objects).

2. Database and tool adaptor toolkits. A database adaptor toolkit would contain the primitive functions from which a database adaptor can be built, i.e., the bridge between an external DBMS and the EIS. The toolkit might contain functions for computing joining distributed data, or converting streams to tuples.
3. The type definitions for system functions to request more memory, encode passwords, etc.
4. Configuration management object type definitions: versions, revisions, alternatives, configurations.
5. Configuration management primitive functions to allow definition of configurations, creation of revisions and versions, selection of alternatives, etc.

Building blocks as specified can be found within a wide variety of design tools. Commonality can provide economies due to shared development and maintenance.

Layer 4: Domain-specific concepts.

This layer contains those concepts that pertain to a particular type of engineering design. The DoD EIS project is using digital hardware design as its application focus, and our examples will be drawn from that domain. The EIS project will define the portion of layer 4 which pertains to CAD tools, design processes, and the structural, behavioral, and physical representation of integrated circuits and printed circuit boards. As resources permit, the project will address other areas such as embedded software or testing. The intention at this layer is to define a useful class of objects for integrated circuit design and show how they could be extended to include other application domains. This is the highest level where new objects will be proposed for standardization.

1. Object type definitions for netlist, gate, cell, signal, etc.
2. Function type definitions for extracting netlists, creating cells, instantiating cells, assigning signal values, etc.
3. Specific management policies, such as the government's 2167 policy for software development.

The richness and variety of the concepts present in layer 4 has two implications.

1. The layer must contain internal substructure (a partitioning, with perhaps a further sublayering) to maintain comprehensibility.
2. We cannot expect the set of concepts at this level to be complete for all time. As design practices and techniques evolve, new object and function types will appear. There must be adequate tools for system administrators and their programmers to permit the definition of new concepts at this layer. To allow clean extensibility, as mentioned earlier, any new concepts must be defined in terms of other concepts at the same and immediately preceding layers.

Application subdomains within the scope of the DoD EIS project include VHDL, which is used to describe the behavioral and structural characteristics of digital circuits; and EDIF, which is used to describe the physical characteristics of digital designs. New subdomains can be established as needed. The use of existing EIS data structures and operations to define new domains is an important feature of this model; it can help assure a degree of inter-tool compatibility which could not be assured if composition occurred at more primitive layers. For example, suppose that the application domains could be defined directly in terms of layer 1 concepts, and imagine that VHDL and EDIF are new subdomains to be added to the preexisting model. In this case, the course of least resistance might be for the two new subdomains to make use of completely disjoint data structures with no constraints between them, with the likely result that EDIF and VHDL design information would not be consistent or sharable.

Partitioning into subdomains will make this layer of the EIS information model more understandable. For example, objects and functions can be classified as follows:

1. Objects and functions that are *joint*, that is, fully equivalent in both VHDL and EDIF.
2. Objects and functions specific to VHDL. Functions in this category may access objects in the joint category.
3. Objects and functions specif to EDIF. Functions in this category may may also access objects in the joint category.

In this example, different functions are applicable to a given object, depending on whether the object is classified as belonging to the VHDL or the EDIF domain. The functions and objects that can be shared between the two domains are placed in the *joint* category.

Layer 5: Product-specific concepts.

Layer 5 specifies a framework for computer-aided design activities. It includes the specifications for the additional tools and services that the vendor provides as well as those from layer 4 which have been implemented and are available to the framework product user. For example:

1. For user interfaces, a script editor, button creator, window design tool.
2. Product-specific database browsers.
3. Instances of CAD tools such as editors, simulators, netlist extractors.
4. Product-wide management policies, such as a versioning policy.

Note that some of the service of this layer, such as layout editors, may actually be provided by pre-existing externally coded tools, attached to layer 5 via adapters.

The design process is one of general progression along two roughly orthogonal axes:

1. Refinement.
   1.1. Requirements specification.
   1.2. Solution architectures.
   1.3. Design objects and functions, as seen, for instance, in logical design.
   1.4. Application domain refinements, for instance, circuit design and layout.
   1.5. A full geometric and physical specification.
2. Stages.
   2.1. Conceptualization.
   2.2. Market studies.
   2.3. Design for feasibility.
   2.4. Detailed design.
   2.5. Test design.
   2.6. Manufacturability.
   2.7. Product plan.
   2.8. Life cycle assessments.

A matrix derived from these two time-related dimensions can provide insight into the information flow that must be served by the tools we consider [Wiederhold 88].

Management policies are used to organize the progression of a design through this coordinate space; a management policy is a superstructure that relates the various design tools as they are applied during the process of bringing a concept to fabrication. For example, a particular design tool is typically appropriate for use only in a small, well-defined subregion of this two-dimensional space, and layer 5 encodes this. Universal rules on flow of information, design policies, and tool sequencing belong in this layer. Managers will determine the contents of layer 5, as they specify the proper sequences of tools to use in design, and express the procedures and systems of approval that are to govern tool usage.

For different application subdomains at layer 4, different policies at layer 5 may be appropriate; in general, we expect the division into application subdomains of layer 4 to be perpetuated at 5.

Layer 6: Site-specific concepts.
Layer 6 is the specification of an EIS as it is made available to an end user. It includes all of the particular tools and databases which have been integrated into a particular EIS, as well as the services from layer 5 are to be made available to the end user. For example, one would expect to see local tools and design management policies at this layer. Other local information, such as hardware configuration descriptions for the local EIS, password files, and access lists all belong here.

The site-specific policies of this layer are once again laid down by the managers of engineers. Policies at this layer 6 are needed in addition to those of layer 5 because usage of the same tool can be governed by different policies in different companies. By making this layer explicit, it will be possible to compare policies, their enforcement, and their effects. Eventually, we will be able to relate these important factors to quality and productivity.

3. DISCUSSION OF THE SEVEN LAYERS

As we move from layer L0 to L6, the semantics attached to an object increases, and the generality or applicability of concepts correspondingly decreases. Layers 0–3 are domain-independent, and their concepts should be equally valuable to EISes for mechanical engineering, software engineering, etc. Layer 4 is where domain or application specific object types are introduced. At the topmost layers (5–6), we find managers setting policies regarding the methodology of the engineering process. In general, the value of the upper layers lies in their ability to constrain the design, its documentation, and the design process in accordance with physical, logical, and policy constraints, such as requirements relating to access to data by designers, correctness of released designs, and design methodologies.

As mentioned earlier, each layer is intended to be a complete specification, containing all the functions, data, tools, and services that are needed to specify the layer above it. (One exception: Some capabilities will be provided by external services, attached via adaptors, and hence will not make use of the lower levels for specification.) In other words, the specification of the capabilities of layer i may make use of a composition, with constraints, of the capabilities of layer $i-1$, but not of lower layers. This means that certain objects and functions will be included in the specifications of multiple layers. For example, a policy on design change notification (layer 6) might require that a mail message (layer 3) be sent to affected designers when a particular design component is revised. However, without an explicit promotion to the next level, lower-level concepts are invisible at higher levels.

The purpose of our layering is to provide an orderly *specification* of EIS concepts. We do not intend for all of these layers to be implemented, and we do not intend the layering as a guide for implementation. For example, the concepts of objects and functions in layer 0 are intended as an abstract underpinning for layer 1, and we do not intend a materialization of that primitive level. Furthermore, for those concepts that are implemented, for the sake of efficiency a concept at layer i may well make direct use of implemented concepts specified at layers below $i-1$. The important point is that the layering scheme not only brings organization to the specification for purposes of clarification, but also allows different software developers an opportunity to implement and market services at various levels in the hierarchy. Because each layer is complete, the implementor of a particular interface need not base an implementation on the lower level objects.

4. EXPECTATIONS FROM LAYERING

We hope that the layering scheme proposed in this paper will provide immediatly a management tool for EIS development. It is critical that the DoD EIS project, and large, multi-user information systems in general be structured so that they can be perceived to be conceptually simple.

A well-designed and demonstrated structure can support standards, which can have major long-range benefits. We envision that within such a structure several classes of vendors can produce EIS products without needing to field large, functionally complete systems.

Perhaps the most obvious of these is the small CAD tool vendor who is currently producing a number of tools for a particular niche market. These vendors' currently cannot offer a potential customer a complete design environment. By building products based on standard EIS interfaces, they will be able to exploit the leverage of other products on the market that complement theirs, and will be able to focus on their specialty areas.

A second market is the relatively young framework systems market. Framework companies focus on the design environment and its management, but their success is limited by the effort needed for a customer in integrating essential design tools into the framework. EIS standards will make it much easier to port design tools into a framework product and have them share data and be managed by the environment's administrative tools.

A third market is the custom CAD tool retailer who combines together a number of tools licensed from other vendors into a complete turn-key system. Again, the retailer is limited by the amount of effort required to integrate tools and the level of interoperability which can be attained. EIS standards will make integration possible and make it realistic for the retailer to offer and maintain custom systems at a reasonable cost.

A fourth market will continue to be with the handful of major CAD vendors who sell a complete design environment. They will continue to offer their customers tightly integrated systems, with all of the tools and support coming from one source. Additionally, by supporting the open EIS interfaces they can offer their customers the ability to add their own in-house tools or a few specialized tools from another vendor. Furthermore, their products may someday be able to communicate with manufacturing, business, or DoD information systems which also support part of the EIS standard.

Finally, there will be additional opportunities for people who want to supply lower level components such as a user interface package to any of the above vendors or to end users or who will provide support for building and customizing in-house Engineering Information Systems.

The layered approach that we have presented supports each of the above vendors. The niche CAD tool builder will use interfaces which are specified on layer 4 or 5. The framework product vendor will provide the services specified by layer 3 and portions of layer 4. The custom system builder will package together a number of tools which possibly interface at layer 5 and offer the user an interface at layer 6. The complete workstation vendor will provide the layer 6 interface. The component vendors will probably sell pieces of layer 3 which the others can use to build the higher level interfaces. These vendors are free to implement the specified functionality of a particular layer however they choose, and can add functionality on top of it to make their products more attractive.

Acknowledgments
The motivation for this paper was the need to relate the many concepts which arose in discussions during the development of the EIS effort. We have tried to integrate notions developed and presented at EIS meetings and related literature, specifically due to Sandy Heiler of CCA, Rob Maffit of TRW, David Bodreau of DACOM, and Bill Kent of Hewlett-Packard. We are also experimenting with a structured glossary to help formalize this work.

The research leading to this note is based on experience gained within the US Air Force VHSIC data management task force, WPAFB Dayton. Fundamental research support for this area derives from DARPA contract N39-84-C211, and from NSF grant DTMP 8619595.

References

[Banerjee 87] J. Banerjee, H-T. Chou, J.F. Garza, W. Kim, D. Woelk, N. Ballou, and H-J. Kim, "Data Model Issues for Object-Oriented Applications"; *ACM Transaction on Office Information Systems,* Vol. 5 No. 1, January 1987, pp. 3–26.

[Batory 86] D. Batory and W. Kim, "Support for Versions of VLSI CAD Objects," *IEEE Transactions on Software Engineering*, 1986.

[Chou 86] H-T. Chou and W. Kim: "A Unifying Framework for Version Control in a CAD Environment," *Proc. 12th VLDB*, Kyoto, Sep. 1986.

[Heiler 88a] S. Heiler and S.B. Zdonik: Views, Data Abstraction, and Inheritance in the FUGUE Data Model; CCA, 17 May 1988. (More background is available in Honeywell, EIS project status report, March 1988, appendix A.)

[Heiler 88b] S. Heiler and S.B. Zdonik: "FUGUE: a Model for Engineering Information Systems and Other Baroque Applications"; CCA, 27 May 1988; International Conf. on Data and Knowledge-bases, Jerusalem, June 1988.

[Katz 87] R. Katz and E. Chang: "Managing Change in a CAD Database"; *Proc. VLDB*, 1987.

[Kumar 88] A. Kumar, R.B. Maffit, M.J. Neally, M.A. Richardson, and M.P. Yeung: EIS Engineering Information Model, TRW report of 1 March 1988; incorporated in Honeywell, EIS project status report, January 1988, appendix A, specifically pages 44–157; overlaps with CCA document, Honeywell, EIS project status report, March 1988, appendix A, pages 128-283.

[Wiederhold 86] Wiederhold, Gio: "Views, Objects, and Databases"; *IEEE Computer*, Vol.19 no.12, December 1986, pages 37-44.

[Wiederhold 88] Wiederhold, Gio: "Engineering Information Systems: Prospects and Problems of Integration"; *Digest of Papers IEEE Spring COMPCON March 1988*, pages 228-229.

5. A Glossary of EIS Terminology

This glossary lists the interrelationships between the myriad EIS concepts. We do not *define* the concepts here; rather, we simply show the linkages between them. Concept names are in italics.

We find that the relationships between concepts fall into three rough categories:

- is composed of — $\Leftarrow$
- is used to model — $\models$
- is a selected subset of — $\subset$

Adaptor $\Leftarrow$ pseudo *design functions*, defined via *tool* documentation.
Aggregation $\subset$ *derivation* by computation over *children*.
Alternative $\subset$ *design objects* related to chronologically prior *objects* on the same layer.
Array countable grouping, $\Leftarrow$ *value objects*.
Assertion $\subset$ *state constraints*, defined declaratively.
Attribute $\subset$ primitive *functions*, used to obtain *value objects*.
Binary relationships $\subset$ *relationships*.
Browsing A sequence $\Leftarrow$ of simple *queries*, probably aided by a *model*.
Children *Dependent objects* one level lower in a *hierarchy*.
Complex object $\subset$ *object* that is composed of other *objects*.
Composition function $\subset$ *functions*, associated with modeling *objects*, *hierarchies*, and *views*.
Configuration $\subset$, and $\Leftarrow$, *versions*, used to establish a higher level *design object*.
Connection $\subset$ *one-to-many relationships*, formalized to apply to a normalized relational model.
Constraint Restriction on the allowable compositions of *objects* and *functions*.
Corollary $\subset$ *design objects* related to another *object* at the same layer.
Data type $\subset$ *type*, defined corresponding to primitives in Ada.
Derivation $\subset$ *function* which creates a *dependent object*.
Dependent object $\subset$ *object*, depends on value and existence on another *object*.
Design function $\subset$ *functions* associated with creative engineering tasks.
Design object $\subset$ *value objects* significant at the highest layers.
Domain of expertise $\subset$ an engineering layer.
Domain $\subset$ *constraints* on values for an *attribute*, specified by `RANGE` and `ENUMERATION`.
Entity $\subset$ *objects*, used to *model* sets of *objects* recognized at a higher layer. Rules may be associated with an *entity* which define and constrain *object* membership.
Engineering Information Model (EIM) $\Leftarrow$ (*design objects* $\times$ *design functions*).
Event $\subset$ *function, application* invoked by a trigger from some extern activity, typically intiated by a designer on a terminal.
File $\subset$ *object*, implemented for persistent *object sets*.
Function abstraction $\models$ ranging over all *relationships* among *objects*, which can be invoked, for instance by an *event* to create new *objects* or instantiate new *relationships*, thereby changing EIS *state*.
Function application invocation of a *function*, possibly generating *state* changes.
Global Schema $\subset$ *EIM* which is machine processable.
Hierarchy $\subset$ *generalization relationships*, establishing *children* and *parents*.
Host $\subset$ *implementation objects*, $\Leftarrow$ *files, tools*.

Identity connection $\subset$ *connection*, used to define *replication* and functional integrity maintenance.

Implementation function $\subset$ *functions* needed for implementation, not visible at the user layers.

Implementation object $\subset$ *objects* needed at implementation level, i.e., not visible at the user layers.

Information flow composition of *transfer functions*.

Inheritance $\subset$ attributes determined by default from *parents*.

Instance functions $\subset$ *functions* which produce *value objects*.

Intersection matrix is a table to document *relationships* between two sets of *objects* or *relationships*.

Key Implementation $\models$ a visible *object ID*.

M-to-N relationships $\subset$ *relationships* which which combine *entity* sets via a joint *intersection* matrix. For each *object* the *intersection* matrix can be used to define subset instances.

Merging $\subset$ *functions*, used to combine *alternatives* into a new *design object*.

Methods $\subset$ *functions* visible at the user layers.

Model a complete and consistent set of concepts for a certain objective.

N-ary lists $\subset$ *complex objects* composed by catenation.

Network $\subset$ *implementation object*.

Objects $\subset$ entity, instance $\models$ a collection of *state* defining variables satisfying *entity constraints*.

Object ID implements *uniqueness constraint*.

One-to-many relationship $\subset$ *relationships*.

Ownership connection $\subset$ *connections* defining *object composition*.

Parent Related *objects* at one level higher in a *hierarchy*.

Path assertions $\subset$ *constraints* attached to *relationships* to define *uniqueness constraints* and other types of *constraints*.

Primitive object $\subset$ *object* of non-decomposable type, such as `INTEGER`, `BOOLEAN`, `REAL`, `CHAR`, `TIME`.

Procedures implementation of a selected subset of *functions*.

Procedural interface $\Leftarrow$ *transfer functions*.

Query $\subset$ *functions*, used to locate and present subsets of *objects* and *functions*.

Reference connection $\subset$ *connections*, used to constrain *object* deletion and insertion.

Relationship $\subset$ *method* with simple declarative semantics.

Schema $\subset$ the machine processable part of a *model*.

Set $\Leftarrow$ related *objects* or *functions*.

Service lattice matrix of *design functions* and *objects* vis-a-vis *application* domains.

Subset connection $\subset$ *connections*, used to define generalization/specialization *relationships*.

State constraints $\subset$ *constraints*, attached to *objects*.

Tool $\Leftarrow$ *design functions* or equivalents made available externally via an *adaptor*.

Transfer function $\subset$ *function applications* which transfers information among *objects*.

Transition constraints $\subset$ *constraints* attached to *functions*.

Type $\subset$ *oconstraints*, used to classify other *object sets* of identical *type composition*.

Uniqueness constraint $\subset$ *constraints*.

Value object $\subset$ *complex objects*; all *complex objects* can be recursively reduced to *primitive objects* using *attribute functions*.

Version $\subset$ *design objects*, typically related to and invalidating a similar prior *object* on the same layer.

View $\Leftarrow$ *functions* and *objects*, implementing a *hierarchy*.

Some domains may be devoted to support services, and not include any design functions per se. For example, a domain may simply provide functions for browsing particular types of information and for passing information between designers and users.

For example, layer L1 might permit instances of any function written in the ADA programming language; the capabilities of layer L1 are not specific to the EIS task. At layer L5, however, only functions relating to high-level allocation and coordination of design tasks might be permitted.

[Wiederhold 86] Wiederhold, Gio, "Views, Objects, and Databases", *IEEE Computer,* Vol. 19, no. 12, December 1986, pages 37-44; to be reprinted in <xx> has this already appeared? Dittrich, Dayal, Buchmann (eds.), *Object-oriented Database Systems*, Springer Verlag, 1988.