Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

B. Veth

An integrated data description language for coding design knowledge

Computer Science/Department of Interactive Systems Report CS-R8731 July

---

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

An Integrated Data Description Language for Coding Design Knowledge

B. Veth
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Abstract: *We present in a unifying framework the basic notions of IDDL (Integrated Data Description Language) to code design knowledge in the IIICAD system. IIICAD is an intelligent, integrated, and interactive computer-aided design environment we are currently developing at the Centre for Mathematics and Computer Science.*

Keywords: CAD, design theory, logic, theory of knowledge, theory of design objects, qualitative reasoning, object oriented programming, logic programming, knowledge engineering, software engineering, prototyping

CR Categories: D.2.m, F.4.1, I.2.1, I.2.4, J.6.
Note: Group Bart Veth in alphabetical order: Varol Akman, Peter Bernus, Paul ten Hagen, Jan Rogier, Tetsuo Tomiyama, Paul Veerkamp.

1. INTRODUCTION

In this paper we deal exclusively with the following issue: How to code design knowledge? We shall start in Section 2 with the IIICAD (Intelligent Integrated Interactive CAD) concepts and present our methodology to develop a useful representation language — i.e. a theoretical approach. Accordingly, we first concentrate on the theory of CAD and then derive requirements and specifications for IDDL. A full account of the implementation details will be left to an upcoming paper although we touch on this subject briefly.

The theory of CAD consists of three parts: theory of design, theory of knowledge, and theory of design objects. In Section 3.1 we introduce a design process model which is derived from a design theory, and in Section 3.2 we show its logical notation. In Section 4 we deal with the theory of knowledge. In Section 5.1 we describe the theory of machine design as an example of the theory of design objects. Designing is a process where we materialize our imagination. Any design process, therefore, cannot escape

from the real world restrictions. In Section 5.2 we illustrate naive physics which treats this aspect. In Section 6 we count the general requirements for IDDL from CAD and software engineering viewpoints. Section 7 closes the paper by citing design policies for IDDL and showing our prototype implementation with an example of bridge design. A word about presentation: throughout the paper we prefix with **DM** the so-called design maxims (Yeomans, Choudry, and ten Hagen 1985) which will be collected in Section 7 and converted into specifications for IDDL.

2. OVERVIEW OF IIICAD

2.1. The Concept of IIICAD

CAD systems are vital elements of almost every facet of the technology but it is also admitted that they are plagued by inflexibility. It is not unjust to claim that the majority of the existing systems are but sophisticated workbenches for engineering drawing. As the application domain becomes serious, designing becomes unmanageable with only this type of support. Since design is essentially an intellectual activity, we need, not surprisingly, more *intelligence* in a system — hence the first I of IIICAD.

Borrowing an analogy from (Bobrow, Mittal, and Stefik 1986), until now CAD systems were built using the *low road* and *middle road* approaches. The low road approach involves *ad hoc* programming (mostly in prehistoric languages like Fortran) and is biased towards geometric information. Middle road systems are more interesting in that they are aware of the fact that they have to incorporate intelligence. They focus on a well-defined domain and collect specialized knowledge coded as say, *if-then* rules. In other words, they become expert systems (e.g. PRIDE (Bobrow, Mittal, and Stefik 1986)). An annoying problem with expert systems is that genuinely expert performance can only rest on knowledge of a model in which an underlying mechanism *understands* what is going on (Kuipers 1986).

Finally, one distinguishes the *high road* systems which IIICAD is aiming at. High road systems are deep systems (as opposed to low and middle road systems which are shallow) in that their knowledge represents the principles and theories underlying the subject "design." In the case of IIICAD, the fundamentals of General Design Theory which is based on axiomatic set theory can be found in (Tomiyama and Yoshikawa 1987).

We do not deny the fact that there are several domain-specific sides to design. For instance, VLSI design is two-dimensional (although this is changing) while mechanical design is inherently three-dimensional. IIICAD incorporates similarities in design, leaving the application-dependent issues to further consideration as side requirements and using intelligence based on a clean and robust design theory. Thus here we are not working on yet another geometric modeler or expert system.

The other two I's of IIICAD correspond to *integration* and *interactivity*. Design systems should support integration because human designers have a unified view of design objects. Interaction requires almost no validation. Good design systems cannot be obtained without using the best man-machine communication techniques.

To summarize:

- In IIICAD we pursue a top-down theoretical approach incorporating more intelligence than expert systems, more integration than geometric databases, and high-level interaction using advanced computer graphics.
- We want IIICAD to be a system based on expandable ideas and a framework where designers can exercise their faculties at large. We believe that the essential thing in a designer is that he builds us his world and IIICAD must give him the freedom to do so.

2.2. Elements of IIICAD

The Supervisor (SPV) is at the core of IIICAD and controls all the information flow. It adds intelligence to the system by comparing user actions with *scenarios* which describe standard design procedures, and by performing error handling when necessary. Since SPV is the central authority for control the following becomes relevant.

DM 1. *IDDL should be able to describe status and control information of the system with origin, destination, and time stamp of the control information.*

While SPV corrects the obvious user errors, it does not have the initiative for the design process itself because IIICAD is envisaged to be a designer's apprentice, not an automatic design environment.

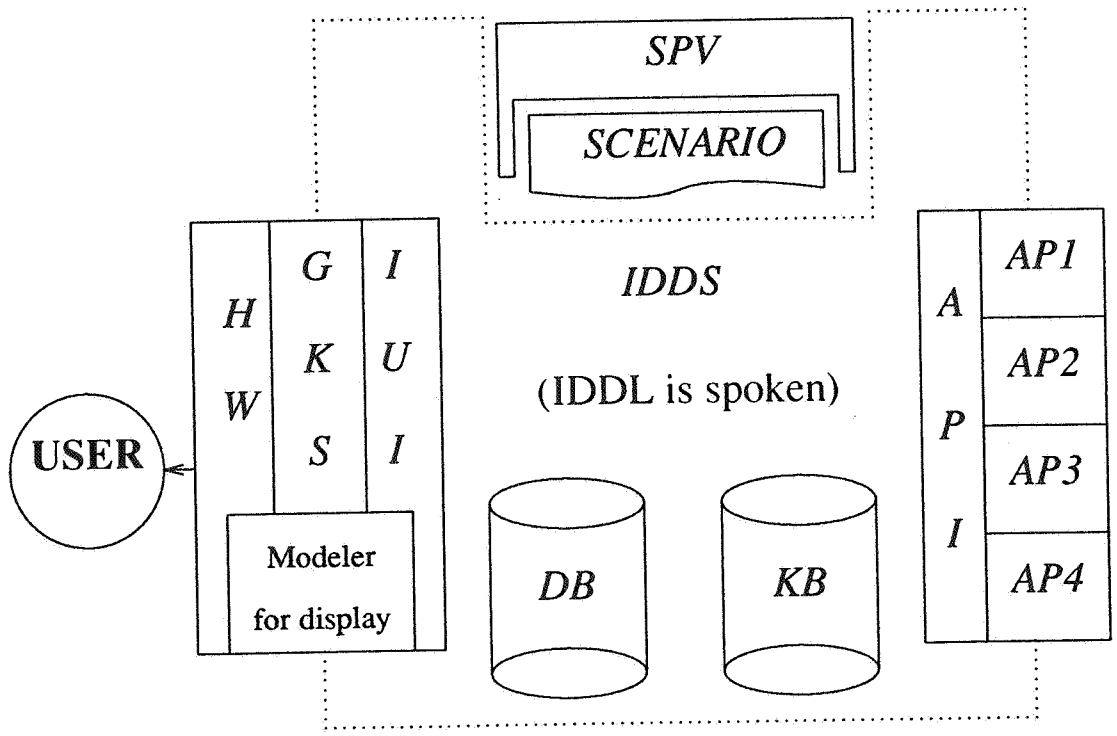

Fig. 1. IIICAD architecture

The Integrated Data Description Schema (IDDS) regiments the data and knowledge bases relieving the user from the burden of specifying where and how to store/retrieve data. IDDS has a language called Integrated Data Description Language (IDDL) spoken by all system elements. IDDL is the means to code the design knowledge and the design object to guarantee integrated descriptions system-wide. Like most modern programming languages, IDDL differentiates between what is commonly known as the *external* and the *internal* contents. The former is essentially nonmathematical information such as input-output behavior and diagnostics. The latter consists of mathematical operations which do the job. More on IDDL will be said in Section 6 and Section 7 which show how IDDL codifies design knowledge. Here it should suffice to remark that internally IDDL will be based on logic and accordingly knowledge engineering is the key factor in building the IIICAD system. IDDL is an essential step in developing IIICAD.

In addition to the above principal elements, IIICAD has a high-level interface called Intelligent User Interface (IUI) which is also driven by scenarios written in IDDL, and the Application Interface (API) which secures the mappings between the central model descriptions about the design object and individual models used by application programs such as geometric modelers, finite element analyzers, etc. Figure 1 shows the preceding elements in block diagram level.

2.3. Software Engineering Viewpoint

Maintainable software systems should be modular both "in the small" to allow alteration of minor components in specific applications and "in the large" to allow changes in major components based on say, the advances in technology. They should be designed for evolution for long time horizons. They should be sturdy and open-ended (Wegner 1984). In this regard, software engineering will always be a leading concern in developing intelligent CAD software such as IIICAD because even the conventional CAD systems are large and complicated. In other words, knowledge engineering is more than software engineering but probably not much more (Bobrow, Mittal, and Stefik 1986).

DM 2. *IDDL, as a language to construct a knowledge base, should support easy maintenance.*

In the development of intelligent CAD systems the underlying strategy is "Plan to throw one away. You will anyhow." (Brooks 1975). Emerging trends of software engineering such as exploratory programming and rapid prototyping are thus crucial. These methods are somewhat more permissive than the more rigid method of formal specification in that they follow the idea of iterative enhancement and consequently, an evolutionary life-cycle approach (Wegner 1984). One starts with a skeletal implementation (rapid prototype) and adds new parts until the system is reasonably completed. This incremental approach is fruitful when the set of tasks and the end result are incompletely defined. Also one is more interested in seeing a glimpse of a future system built as a prototype in order to assess its strengths and weaknesses globally. Exploratory programming using powerful workstations and modern languages (e.g. Smalltalk-80[1]) makes this process very effective (Ramamoorthy, Shekhar, and Garg 1987).

---

[1] Smalltalk-80 is a trademark of Xerox Corporation.

DM 3. *IDDL must support incremental programming.*

3. DESIGN THEORY

3.1. Modeling of Design Processes

Design theory (Yoshikawa 1981) provides a strong basis for formalizing design processes and design knowledge. For this purpose, we use General Design Theory (Tomiyama and Yoshikawa 1987; Yoshikawa 1981) which is based on axiomatic set theory and models designing as a mapping from the function space where the specifications are described in terms of functions, onto the attribute space where the design solutions are described in terms of attributes.

There are many interesting results derived from General Design Theory; we emphasize in Section 3.2 the possibility of a logical formalization of design processes. Figure 2 shows a design process model derived from General Design Theory. The basic idea is as follows (Tomiyama and ten Hagen 1987a; Tomiyama and ten Hagen 1987b):

- A designer, given the specifications, may try to select a candidate and refine it in a stepwise manner, rather than trying to get the solution directly from the specifications.
- Therefore, a design process can be regarded as an evolutionary process of such intermediate descriptions of the design objects rather than just a mapping. The collection of these intermediate descriptions can be used as the central model about the design solution and we call it a *metamodel.*
- The designer will evaluate the candidate to see whether it satisfies the specifications or not. To do so, he derives various kinds of models of the design object from one central model (i.e. the metamodel).

Our discussion leads to the following design maxims:

DM 4. *IDDL should be able to describe not only design objects but also design processes.*

DM 5. *IDDL should be able to describe metamodels and models (for evaluation) derived from the metamodel.*

DM 6. *IDDL should be able to describe the stepwise nature of the design process.*

DM 7. *IDDL should be able to describe knowledge to detail the metamodel, to check its feasibility, and to control the detailing process.*

DM 8. *IDDL should be able to describe knowledge to derive models for evaluation from the metamodel and knowledge to evaluate models.*

DM 9. *IDDL should allow multiple views of a design object, which are possibly independent but still correlated.*

To illustrate a design process, we need to recognize three major components: *entities, attributes* of entities, and *relationships* among entities. A design process is thus a collection of small steps to obtain complete information about these three. We can also observe components which do not change during design. For instance, when we

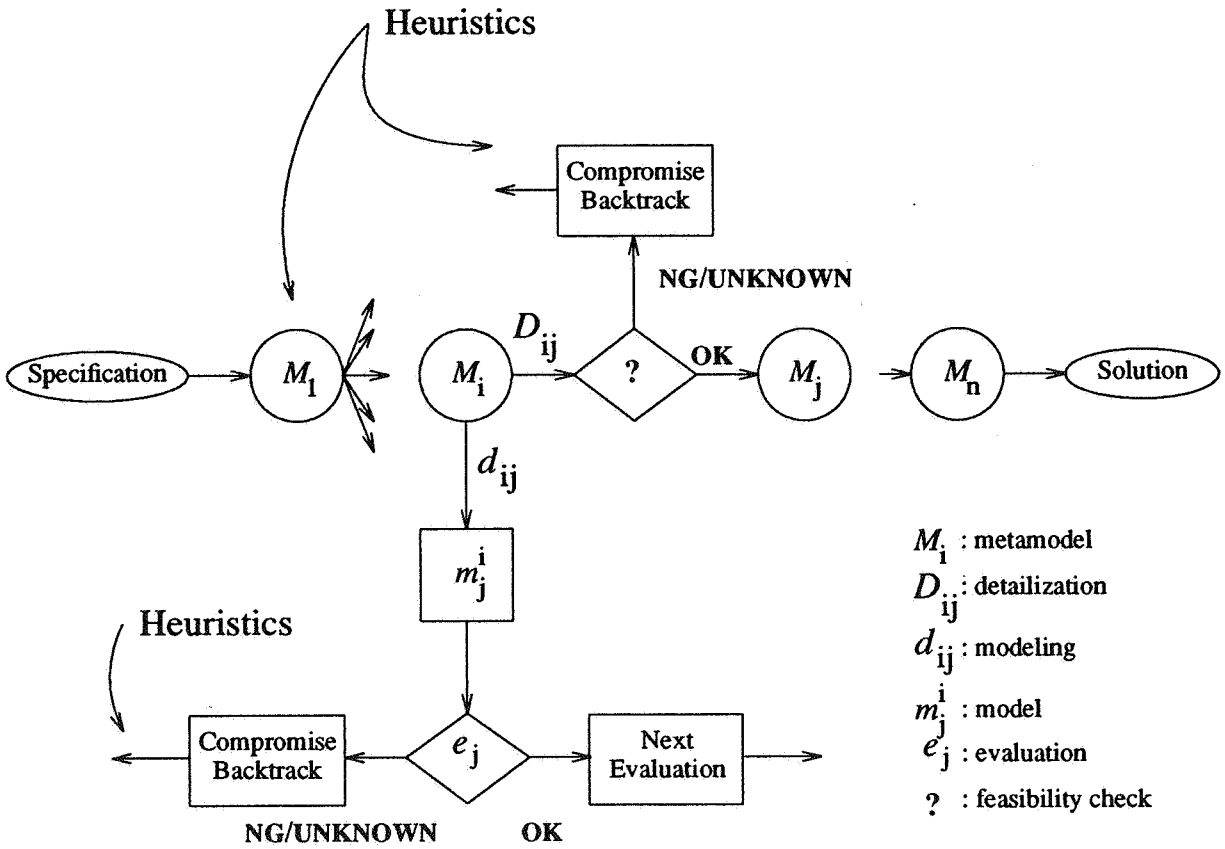

Fig. 2. Design process model due to General Design Theory

design a board we simply use a VLSI chip as a building block that cannot be changed. Let us call such objects *invariants* in a design process. Descriptions about the board at this level are, oppositely, dynamically changed during the design process. Let us call them *variants* in a design process. Clearly, the concept of e.g. a point should not change while designing a geometric entity; however attributes such as coordinates of a point may be frequently changed. Let us call such changeable concepts associated with variants *covariants.*

DM 10. *IDDL should be able to describe invariants, variants, and covariants in design processes.*

Another important thing about the design process model is that eventually we need to check physical constraints (i.e. feasibility check). This implies further that we need to distinguish success and failure, known and unknown, necessity and possibility, etc.

DM 11. *IDDL should be able to describe positive and negative information, known and unknown, and modalities such as necessity and possibility.*

3.2. Design Processes: Logical Formalization

Let us assume predicate logic as the basis of our discussion. To control the stepwise refinement of the design process there is a need to express unknown, uncertain (default), and temporal information about the design object. For this reason we equip our logic language with three-valued logic, modal logic, inheritance, and situational calculus.

Considering the metamodel evolution model in Fig. 2, the system starts from the specification S of the design object and it continues the design process until the goal G is reached:

$$S \rightarrow M^0 \rightarrow \cdots \rightarrow M^i \rightarrow M^{i+1} \rightarrow \cdots \rightarrow M^n \rightarrow G$$

We define $\{q^i\}$ as the set of propositions at the metamodel state M^i. In other words $\{q^i\}$ is the current state of knowledge about the design object. There are two possibilities: either the current state of knowledge is complete and consistent or there is some incompleteness or inconsistency. In the first case the goal is reached and we finished the design process. In the latter case we need to proceed to a next metamodel in order to solve the incompleteness or inconsistency.

DM 12. *IDDL should let the inconsistency of a certain metamodel be represented, but this inconsistency needs to be resolved when transferring to a next metamodel.*

We need language constructs to evaluate a metamodel and to derive new properties or to update uncertain or unknown properties in order to get more detailed knowledge about the design object. The decisive point is how to proceed from M^i to M^{i+1}; i.e. given $\{q^i\}$ how do we find $\{q^{i+1}\}$? We shall adopt the following strategy:

- From $\{q^i\}$ we can derive p, so the next state $M^{i+1} = \{q^i\} \cup p$. This means that the knowledge base is extended by asserting property p. In practice p might come from design procedures, default assumptions, results of engineering analyses, and so on. Before the acceptance of state M^{i+1} the consistency of the new metamodel has to be checked. For this purpose we can use the appropriate set of logical inference rules.
- We use the modal operator $\Box$ to express default values. Thus $\Box p$ means it is possible, but probably not the case that p; $\blacksquare p$ states it is necessary that p. Note that $\Box p$ is equivalent to $\neg\blacksquare \neg p$ (McDermott 1982).

DM 13. *IDDL should incorporate modal logic.*

- If we have $\Box q_j$, can derive $\neg q_j$, and it is not based upon default properties then we assume $\neg q_j$. In other words $M^{i+1} = \{q^i\} - \{\Box q_j\} \cup \{\neg q_j\}$. To see this, imagine that the designer wants to design a bridge and the system needs the length which is unknown. In this case the system knows that bridges normally have a length and it can conclude by an inheritance mechanism that the length of the bridge is L. This will be asserted to the knowledge base as $\Box$ *equal*(*length*(*bridge*), L). If in a next state of the design process the real length RL of the bridge is determined, $\Box$ *equal*(*length*(*bridge*), L) can be changed into $\blacksquare$ *equal*(*length*(*bridge*), RL).
- The Skolem constant ω is used to denote unknown values; so if we have $q_j = p(\omega) \wedge \cdots$ and we can derive q_j' without unknown values, then we have

the next metamodel $M^{i+1} = \{q^i\} - \{q_j\} \cup \{q_j'\}$. Referring to the previous example about the length of the bridge, the system can assume an unknown value instead of a default value in *equal*(*length*(*bridge*), ω). When we want to express the fact that a certain property is unknown, we use $\perp p$.

DM 14. *IDDL should have both the Skolem constant and the unknown operator.*

- If a severe inconsistency is encountered which cannot be resolved by the system in terms of deriving more knowledge or stepping back to a previous metamodel, apparently an inconsistency in the specification provided by the designer has been found. The designer should be notified with this inconsistency together with exact transactions so that he can fix it and restart the design.

DM 15. *IDDL should have facilities for error handling, when it encounters inconsistent or incomplete states, with the help of the designer.*

To add a certain proposition an assertion operator *assert*($p(x)$) is needed. To modify propositions we need a *change*($p(x)$) operator. After these operations the knowledge base must still be consistent (cf. **DM** 12).

DM 16. *To control the behavior of the system IDDL should have metaknowledge that chooses which rule to apply at a certain time.*

In other words, IDDL must be able to describe a design process so that during designing we can "design" designing procedures using metaknowledge.

4. THEORY OF KNOWLEDGE

Theory of knowledge is necessary especially to put our knowledge into a particular framework and to utilize it in the IIICAD architecture. The following discussion is based on our result (Tomiyama and ten Hagen 1987c) and its most significant contribution is the distinction between two opposing knowledge representation methods, i.e. *extensional* vs. *intensional* descriptions.

In order to discuss design, we need to describe entities, their properties, and relationships among entities as mentioned in Section 3.1. In an extensional description method, the fact that an entity e has property p is described by $p(e)$ and the fact that entities e_1 and e_2 are in a relationship r is described by $r(e_1, e_2)$. In an intensional description method these two facts can be represented by $e(p)$ and *relation*(e_1, r, e_2), respectively. The extensional descriptions do not assume any preconceptions while the intensional descriptions assume preconceptions such as that e's property is limited to one particular p. This means that an intensional description is equivalent to an extensional description with some assumptions, such as the number of arguments, the order of arguments, the type of arguments, etc.

These two description methods are basically equivalent except for assumptions. Since an intensional description assumes something predefined, when it has to be changed this results in changing those predefined (and perhaps implicit) conditions. For instance, a mechanical part, say a shaft, might be represented by

shaft(*diameter*, *length*, $bearing_1$, $bearing_2$).

If we now want to add new attributes, such as transferring power, this results in a redefinition of this *shaft* predicate. On the other hand, an extensional description might be the set of the following facts:

$$shaft(s),\ equal(diameter(s),\ D),\ equal(length(s),\ L),$$

$$supported-by(s,\ b_1),\ supported-by(s,\ b_2),\ bearing(b_1),\ bearing(b_2).$$

In this example, an extensional description does not assume anything, e.g. s is just a name.

DM 17. *IDDL has two kinds of names: system names are internal and should be unique whereas user names are external and modifiable.*

In an extensional description we need to write numerous (and often very obvious) descriptions. However, modifying such a representation is just adding or deleting facts (thus incremental, cf. **DM** 3). On the other hand, an intensional description assumes a predefined scheme which might be difficult to change but shows high performance. For instance, it is easily predicted that the computation to determine two bearings supporting a shaft is reduced to an address calculation.

It is well-known that CAD applications request a flexible data description scheme which is easy to modify (Lorie 1982). From this point of view, the extensional description method is important in IIICAD because of the incremental nature of design processes. Independent (but still correlated) multiple views of design object demand independent small partitionings in the database. This is easily achieved by an extensional description method because we only have to pick up relevant facts. In an intensional description method this requires to create a totally new scheme.

As discussed in Section 3, there are variants which change during the design process. The extensional description can be used to describe these variants. To this end, we know that the logic programming paradigm (Kowalski 1978) is very useful to implement such description methods. On the other hand, there are invariants we use as building blocks for designing. They do not change their structural properties although values of their attributes might change. Invariants, therefore, can be represented in an intensional description method and their properties (or attributes) can be represented as covariants. Having intensional descriptions may also contribute to improving the performance.

DM 18. *IDDL should have both an extensional description method and an intensional one.*

DM 19. *Invariants in IDDL will be represented as objects, variants will be constructed on the predicate level, and covariants will be represented by functions.*

5. THEORY OF DESIGN OBJECTS

5.1. Theory of Machine Design

Design is regarded as a mapping from the function space onto the attribute space. This requires IDDL to have both attributive and functional representations. There are several issues in representing the attributive information (Tomiyama and Yoshikawa

1985; Tomiyama and ten Hagen 1987a). First, an attribute does not necessarily have a value. In the design process, it often happens that an attribute is only known to exist and its value is not yet decided. Hence IDDL should be based on three-valued logic including *unknown* (cf. **DM** 14). Second, attributive information refers to the structure of an entity. The structure of an entity might be characterized by existence of substructures and relationships among substructures. Information on the number of substructures is also needed.

DM 20. *IDDL should make a distinction between the facts that an entity has an attribute and that an attribute has a value.*

DM 21. *IDDL should be able to represent part-assembly relationships and relationships among parts in order to represent structures.*

DM 22. *IDDL should be able to deal with cardinalities (i.e., number of elements in a set).*

On the other hand, the representation of functions is a rather difficult issue. Unfortunately, it is not yet known in which language we can describe functions of e.g. machines. There is, however, a hope that functions can be represented in terms of physical phenomena that the machine exhibits (Tomiyama and Yoshikawa 1987). From this point of view, the representation of functions can be reduced to the representation of physical phenomena and qualitative reasoning (cf. Section 5.2).

At one time in the metamodel evolution model, the designer will focus at a particular part of the design process or the design object. When this focusing is taking place, the information about the rest of the design process or the design object should not be accessible and stay unaffected. (This is the principle of abstract data type languages.) In order to make focusing more effective, we must create a small world which represents it. For example, we must be able to control applicable predicates to particular classes of entities, although this inevitably asks for higher order predicate logic.

DM 23. *IDDL should have a focusing control mechanism which is able to create a small world where it is clearly defined what kind of information is accessible.*

We must also be able to see a particular part of the design object. When we are considering a particular object, we must be able to see its inner structure represented by variants. On the other hand, we may use that object as a building block when we are working on another object. This requests transition between different abstraction levels and the same information (e.g. an object at some level) must be seen differently (e.g. as a collection of predicates) (Fig. 3).

DM 24. *IDDL should be able to describe hierarchical enclosing controls.*

5.2. Naive Physics, Qualitative Reasoning, and Design

Naive physics observes that people are generally very good at functioning in the physical world and tries to develop a formal framework to serve as a basis to export this human capability to computers (Hayes 1985). As such, it constitutes a major part of what is known as *commonsense reasoning* in AI. Naive physics concepts are needed in design because in many cases design objects will have a physical existence and accordingly obey natural laws. If we want to create designs corresponding to physically

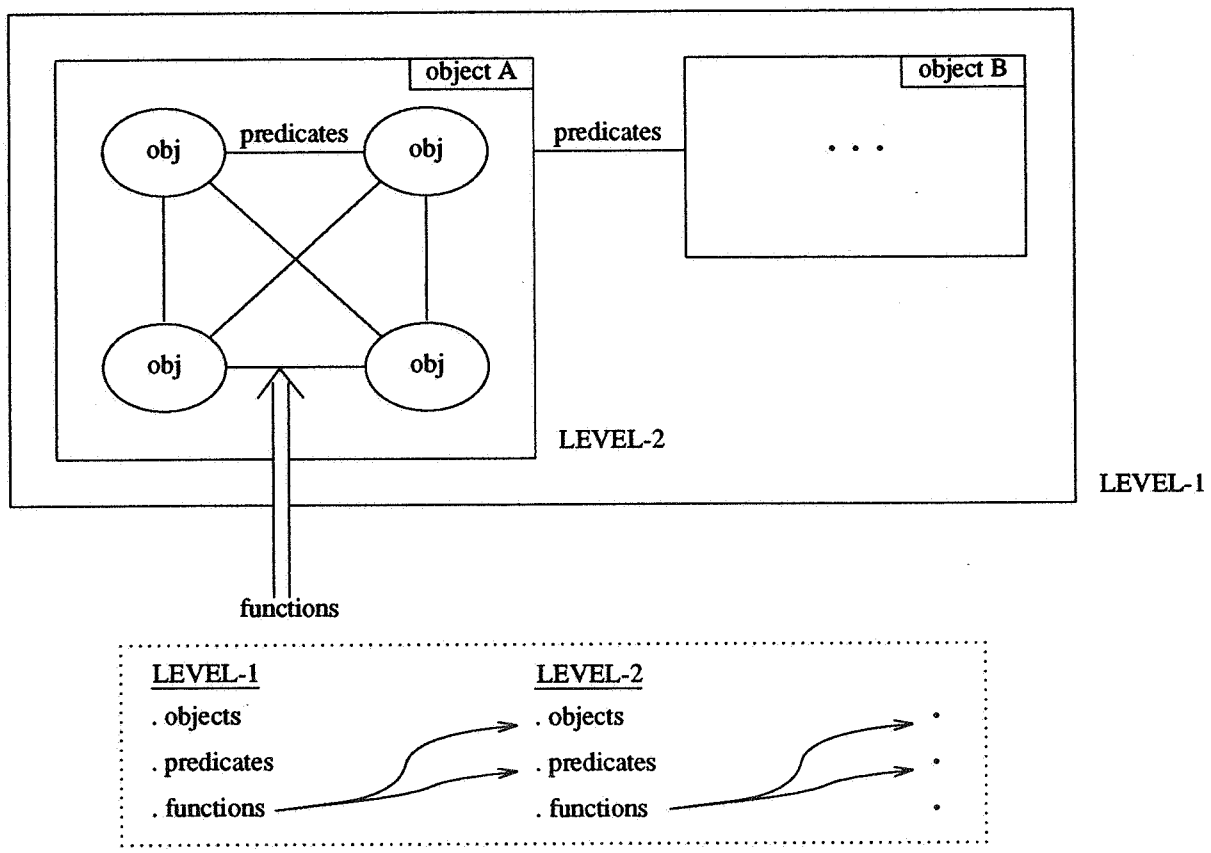

Fig. 3. Hierarchical enclosing control

realizable (read manufacturable) design objects then we will have to refer to naive physics primitives such as solids, space, motion, etc. Furthermore, if we want to reason about a design object in its destined environment (think of a pressure regulator to be installed in a nuclear reactor) we will need naive physics notions such as envisioning, simulation, diagnostics, etc.

A basic commonsense notion is *causation.* We seem to have no difficulty in grasping the relationship between two events (or states of affairs) such that the first brings about the second. Basic notions of modal (e.g. temporal) logic can be used to talk about causality. An intricacy is brought about by *teleological explanations* — i.e. certain phenomena seem to be best explained by intentions or purposes rather than by means of prior causes. This closely resembles to our way of looking at a design object from different viewpoints.

Classical physics seems not too useful in formalizing naive physics because it describes everything in exact terms. Starting with the introductory level textbooks, physics laws are based on the presupposition that the readers have a shared *prephysics* knowledge (de Kleer 1975). In fact, mathematical formalizations of physics, unless aided by verbal explanations, occasionally hide causality. What makes then a good

formalization of naive physics? Both mathematical and computational criteria are important. Some guidelines may be given at this point:

- *Categorical angle:* Study objects through their universal properties which characterize them, rather than through their anatomical properties. See intriguing similarities and exploit abstractness to arrive at theories of utmost generality unattainable in other ways.
- *Mechanistic outlook and deductionism:* Regard physical situations as machines comprising individual components each of which contributing to the overall behavior of the machine. This means that the behavior of a physical structure will be completely accounted for by the way its physical constituents act. To a first order approximation, three kinds of constituents are enough (de Kleer and Brown 1984): materials (such as air, water), components (such as containers, wheels), and channels (such as electric cables, conduits).
- *No-function-in-structure:* This follows from the preceding guideline. Briefly, one has a catalog of components and these have associated laws which do not make assumptions about how they are employed in a certain context (de Kleer and Brown 1984).
- *Confluences:* These are better known as qualitative differential equations (Forbus 1984; Kuipers 1986). One first reduces continuous real-valued variables to discrete-valued variables taking only a small number of values, say $+$, $-$, and 0. This process maps differential equations to confluences. Normally, a single confluence will not be able to characterize the behavior of a component over its entire operation region.
- *Envisionment, simulation, and diagnosis:* In the former one starts with a structural description and determines all possible sequences of behavior. In simulation, one starts again with a structural description but this time he is given some initial conditions to determine a probable course of future behavior. In diagnosis, one starts with some specified behavior expected from a system and tries to see why the system is misbehaving.
- *Topological scene description:* This suggests that one may temporarily ignore the exact coordinates in some geometric situation and model it using topological notions like homotopies, isolation, etc.
- *Frame problem:* This is well-known. When some action takes place in a situational calculus-like representation, how does one tell what facts change and what facts stay unaffected? The answer is, one has to write explicit axioms that state what changes and what remains the same. One may avoid this problem at least partially by adopting *histories* (Hayes 1985) — descriptions extended through time but always spatially delimited (in contrast to situational calculus situations which are instants spatially unbounded). This reflects a choice to ban action at a distance (Forbus 1984).
- *Modal logic:* We find it handy to use logic with modes of truth, viz. modal logic with necessity and possibility. As explained earlier, here we have not only affirmations such as that proposition p is true, but also stronger ones such as that p is necessary, and weaker ones such as that p is possible. Modal logic is useful to code naive physics. Consider examples such as

$$below(obj,\ surface) \wedge not - glued(obj,\ surface) \supset \blacksquare\ fall(obj)$$

and

$$at - top(obj,\ inclined - surface) \wedge above(obj,\ inclined - surface) \supset \Box\ slide(obj)$$

where the latter possibility being dictated by our incomplete knowledge about e.g. friction.

According to these guidelines, **DMs** 9, 13, and 21 are relevant. In addition:

DM 25. *IDDL should be able to carry out simple algebraic manipulations — this is necessitated by e.g. qualitative reasoning with confluences.*

6. REQUIREMENTS TO IDDL AS A KERNEL LANGUAGE OF CAD

6.1. CAD Perspective

CAD is no longer considered as a tool for speeding up the creation of exact product definitions in the form of text and drawings. Because CAD needs a coordinated flow of information between system and user, the functional view of system design should consider the totality of design, not only those functions which will be carried out by a computer. It needs a language to represent the flow of design (cf. **DM** 1). The number of design rules which exhaust a realistic area within electrical, mechanical or civil engineering lies around tens of thousands. The amount of attributive data to be handled by the actual management of the design process could be around hundreds of megabytes, not mentioning the attributive data in the background databases of a design office.

DM 26. *IDDL should have a mechanism for structuring knowledge.*

DM 27. *As with design knowledge, design object representation calls for an encapsulation and structuring mechanism for it to be representable in reasonable form.*

Design produces intermediate results which are incomplete and even inconsistent during time spanning series of transactions. The design object's attributive representation must allow assumptions to be used for the evolution of the design object (cf. Section 3). Result of nonmonotonic reasoning must be checked for feasibility and consistency.

DM 28. *IDDL, using nonmonotonicity, should be able to retract assumed but later on unconsidered propositions.*

DM 29. *IDDL should be able to check consistency and completeness.*

From the viewpoint of interactive design, it is desirable for the human designer to be able to "mark" intermediate design stages, and later go back to them for examining or resuming from there.

DM 30. *The stages of design evolution must be representable on the level of IDDL.*

6.2. Software Perspective

From a software engineering point of view two types of requirements influence our language design. One is that the system to be built using IDDL will after all be a software system with high complexity. Therefore the language design must reflect considerations for managing complexity in software design (cf. Section 2). Especially, due to our inability of separating specification from experimentation, we would like to have an environment where the two can be done in parallel.

Excellent opportunities are found in object oriented style of programming (Stefik and Bobrow 1986). Object oriented programming delivers extensibility, flexible modifications of code, and reusability. Important issues in object oriented programming are data encapsulation and information hiding. Thus an object can be regarded as an independent program which knows everything about itself. Reuse of software is helped by the class inheritance mechanism. Other flexibilities of languages like Smalltalk-80 (Goldberg and Robson 1983) and Loops (Stefik, Bobrow, and Kahn 1986) are incremental compilation and dynamic binding. It is an additional benefit of object oriented programming systems that they offer rich system building tools and good user interfaces. Therefore object oriented programming is a good choice for creating IIICAD. On the other hand, logic programming is powerful for problem solving since it reflects the reasoning process most naturally and directly.

DM 31. *IDDL uses the logic programming paradigm to express the design process for manipulating design objects, whereas the object oriented programming paradigm is used to express design objects.*

DM 32. *In IDDL invariants, variants, and covariants will respectively be represented by objects, their internal and external relationships, and behavior of objects.*

The second aspect of language design is how representational tasks can be unified. A typical epistemological view (Brachman 1979) includes class – subclass, part – subpart, class – member-of-class, prototype-of-class – member-of-class, and functional abstraction.

DM 33. *IDDL should have mechanisms to represent inheritance.*

7. IDDL SPECIFICATIONS

7.1. From Design Policies to Specifications

In the preceding sections we have counted 33 design maxims. The derivation of IDDL specifications took place as follows. First we classified them into several categories so that we could see the relationships among them. We extracted a "keyword" from each of those categories, for example *encapsulation* was arrived at after considering **DMs** 10, 18, 19, 27, and 32; for *modalities* we considered **DMs** 13 and 28.

We then derived features considered to be essential for IDDL from those keywords. For instance, the feature *no automatic backtracking* was derived from the keyword *metaknowledge for control*. This is due to the expectation that if an automatic backtracking mechanism and metaknowledge for control are installed together then the control would become unmanageable.

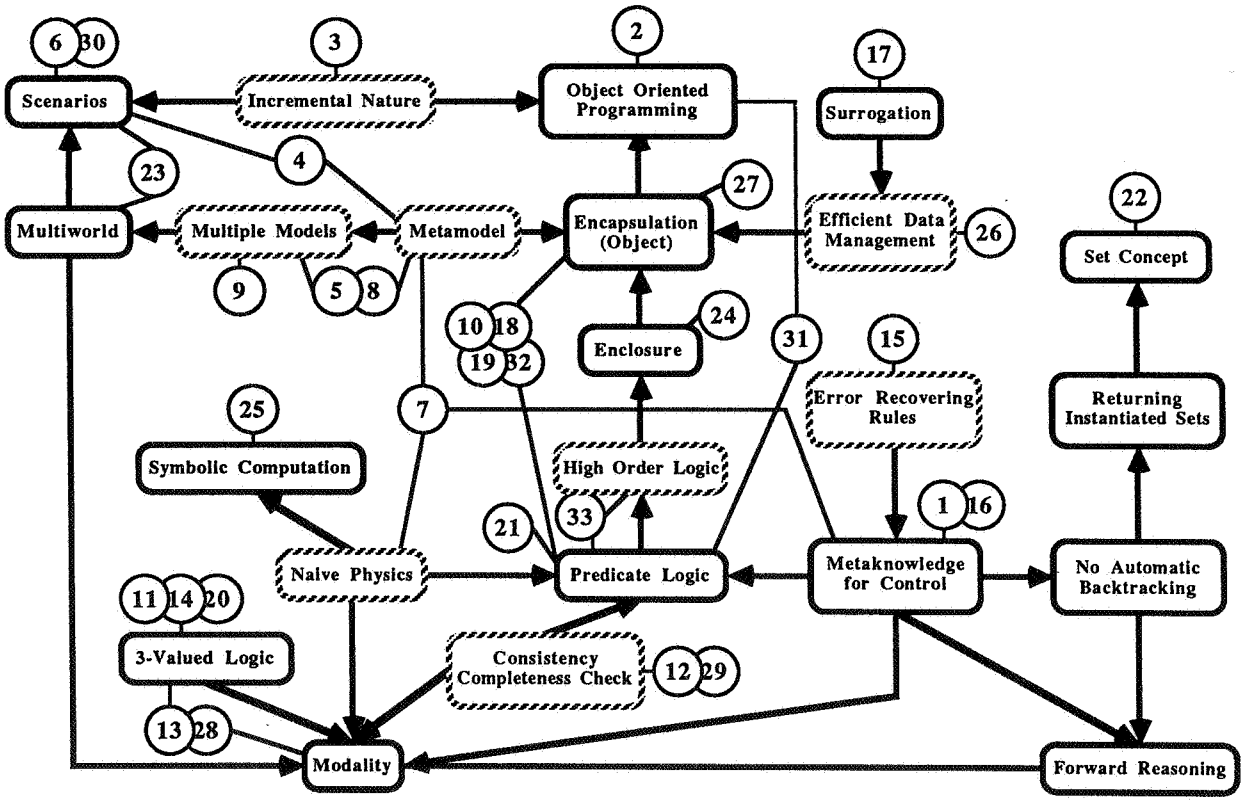

Fig. 4. Classification of IDDL design maxims

Figure 4 shows the relationships among design maxims, keywords, and features. In this figure, the small circles correspond to **DMs,** dotted boxes are keywords, and the solid boxes are features. Some nontrivial features are explained below.

- *Enclosure* is a mechanism to absorb the difference in abstraction levels (see Section 5.1).
- *Forward reasoning* is requested by the features *metaknowledge for control* and *no automatic backtracking.*
- *Metaknowledge for control* demands other features like predicate logic, no automatic backtracking, forward reasoning, and introduction of modality.
- *Modality* in the logic system is one of the main issues in IDDL, for the inference control will be dependent on it. We are thinking of incorporating necessity/possibility and known/unknown operators.
- *Multiworld* mechanism is used to describe metamodel evolution and evaluation of model. The scenario mechanism creates a completely isolated world which is independent but still capable of having relationships with other worlds.
- *Returning instantiated sets* to an inquiry is requested due to *no automatic backtracking* feature. Elements of IIICAD will return all possible instances to an

inquiry, which does not require automatic backtracking for exhaustive search and gives possibilities for both depth- and breadth-first searches. This further begs for the introduction of *set concept*.

- *Scenarios* are the kernel mechanisms to realize multiworlds which are important to describe the stepwise nature of design processes.
- *Three valued logic* will play an important role in IDDL. It will however be introduced as operators rather than truth values to avoid unnecessary complexity in the inference algorithm.

7.2. Example from IDDL Prototype

Figure 5 shows a typical display from our prototype implementation of IDDL. Based on the discussions in Section 7.1 we have developed this version on our Smalltalk-80 system. In Fig. 5, design browsers are used to manipulate design information such as scenarios, constraints, predicates. In this version of IDDL, *constraints* correspond to the covariants of Section 3.1 and are meant to be the design specifications, while *predicates* correspond to the variants and are used for object description.

The two windows on the left of Fig. 5 are snapshots of the constraints for the bridge and its subparts. The other two overlapping windows on the right show the part-assembly relationships among substructures. The designer can explore various possibilities by communicating with the system through these windows.

8. CONCLUDING REMARKS

In this paper, we presented a unifying framework to describe design knowledge. Our starting point, theory of CAD, allowed us to formulate design maxims which were converted into IDDL specifications. Theory of CAD enabled us to understand, clarify, model, and formalize design processes and design knowledge in an intelligent CAD environment.

The development of IDDL was given priority to the development of other subsystems of IIICAD. The current goals of the project are as follows:

- To implement more powerful prototyping tools for IDDL development.
- To construct a more complete version of IDDL by using these tools.

Near future work includes:

- To develop subsystems of IIICAD such as SPV, API, and IUI.
- To incorporate existing CAD tools and "knowledge-based systems" in the IIICAD framework.
- To justify our methodology of developing IIICAD and eventually to prove the effectiveness of our theory of CAD by case studies.

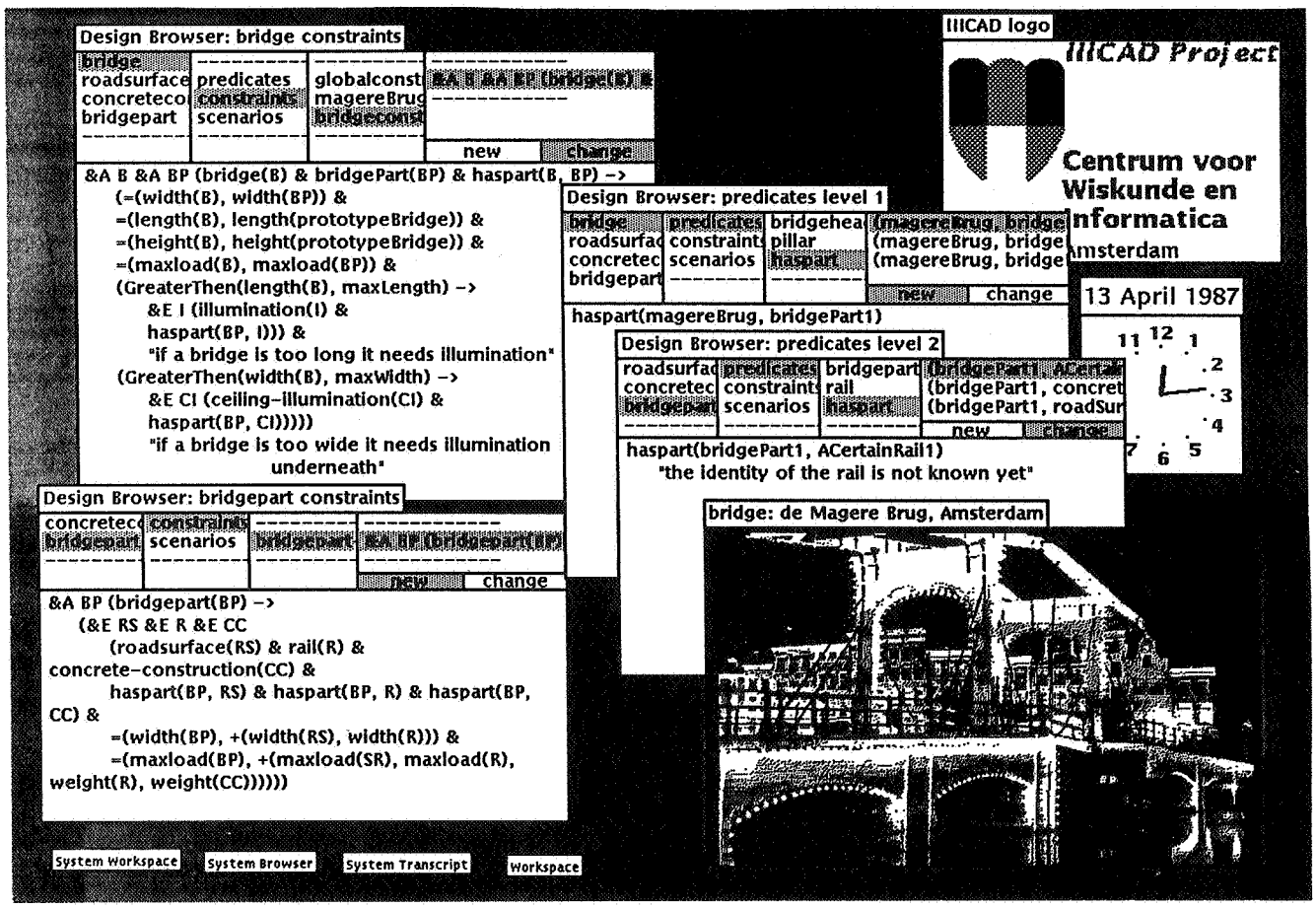

Fig. 5. Example bridge design with IDDL prototype

ACKNOWLEDGMENT

We are grateful to M.M. Megens for her enthusiastic work in IDDL implementation.

REFERENCES

Bobrow, D. G., Mittal, S., and Stefik, M. (September 1986): "Expert systems: Perils and promise," *Communications of the ACM*, **29**(9), pp. 880-894.

Brachman, R. J. (1979): "On the epistemological status of semantic networks," in *Associative Networks: Representation and Use of Knowledge by Computers*, Findler, N. V. (ed.), Academic Press, New York, pp. 3-50.

Brooks, F. (1975): *The Mythical Man-Month*, Addison-Wesley, Reading, MA, USA.

de Kleer, J. (December 1975): "Qualitative and quantitative knowledge in classical mechanics," Technical Report No. AI-TR-352, Artificial Intelligence Lab, MIT, Cambridge, MA, USA.

de Kleer, J. and Brown, J. S. (1984): "A qualitative physics based on confluences," *Artificial Intelligence*, **24** , pp. 7-83.

Forbus, K. (1984): "Qualitative process theory," *Artificial Intelligence*, **24** , pp. 85-168.

Goldberg, A. and Robson, D. (1983): *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading, MA, USA.

Hayes, P. J. (1985): "The second naive physics manifesto," in *Formal Theories of the Commonsense World*, Hobbs, J. R. and Moore, R. C. (eds.), Ablex, Norwood, NJ, USA, pp. 1-36.

Kowalski, R. (1978): "Logic for data description," in *Logic and Data Bases*, Gallaire, H. and Minker, J. (eds.), Plenum, London, pp. 77-103.

Kuipers, B. (1986): "Qualitative simulation," *Artificial Intelligence*, **29** , pp. 289-338.

Lorie, R. (1982): "Issues in database for design applications," in *File Structures and Data Bases for CAD*, Encarnacao, J. and Krause, F. L. (eds.), North-Holland, Amsterdam, pp. 213-222.

McDermott, D. (January 1982): "Nonmonotonic logic II: Nonmonotonic modal theories," *Journal of the ACM*, **29**(1), pp. 33-57.

Ramamoorthy, C., Shekhar, S., and Garg, V. (January 1987): "Software development support for AI programs," *IEEE Computer*, **20**(1), pp. 30-40.

Stefik, M. and Bobrow, D. G. (Winter 1986): "Object-oriented programming: Themes and variations," *AI Magazine*, **6**(4), pp. 40-62.

Stefik, M., Bobrow, D., and Kahn, K. (January 1986): "Integrating access oriented programming with a multiparadigm environment," *IEEE Software*, **3**(1), pp. 10-18.

Tomiyama, T. and Yoshikawa, H. (1985): "Requirements and principles for intelligent CAD systems," in *Knowledge Engineering in Computer-Aided Design, Proceedings of the IFIP Working Group 5.2 Working Conference 1984 (Budapest)*, Gero, J. S. (ed.), North-Holland, Amsterdam, pp. 1-23.

Tomiyama, T. and Yoshikawa, H. (1987): "Extended general design theory," in *Design Theory for CAD, Proceedings of the IFIP Working Group 5.2 Working Conference 1985 (Tokyo)*, Yoshikawa, H. and Warman, E. A. (eds.), North-Holland, Amsterdam, pp. 95-130.

Tomiyama, T. and ten Hagen, P. J. W. (April 1987): "The Concept of Intelligent Integrated Interactive CAD Systems," CWI Report No. CS-R8717, Centre for Mathematics and Computer Science, Amsterdam.

Tomiyama, T. and ten Hagen, P. J. W. (1987): "Organization of design knowledge in an intelligent CAD environment," in *Expert Systems in Computer-Aided Design, Proceedings of the IFIP W.G. 5.2 Working Conference 1987 (Sydney)*, Gero, J. S.

(ed.), North-Holland, Amsterdam, pp. 119-147.

Tomiyama, T. and ten Hagen, P. J. W. (June 1987): "Representing Knowledge in Two Distinct Descriptions: Extensional vs. Intensional," CWI Report No. CS-R8728, Centre for Mathematics and Computer Science, Amsterdam.

Wegner, P. (July 1984): "Capital-intensive software technology," *IEEE Software*, 1(3), pp. 7-45.

Yeomans, R., Choudry, A., and ten Hagen, P. J. W. (eds.) (1985): *Design Rules for a CIM System*, North-Holland, Amsterdam.

Yoshikawa, H. (1981): "General design theory and a CAD system," in *Man-Machine Communication in CAD/CAM, Proceedings of the IFIP Working Group 5.2 Working Conference 1980 (Tokyo)*, Sata, T. and Warman, E. A. (eds.), North-Holland, Amsterdam, pp. 35-58.