

DISTRIBUTED AND COLLABORATIVE COMPUTER-AIDED ENVIRONMENTS IN PROCESS ENGINEERING DESIGN

Arthur W. Westerberg
Department of Chemical Engineering
and Engineering Design Research Center

The n -dim group¹
Engineering Design Research Center

Carnegie Mellon
Pittsburgh, PA 15213

Abstract

n -dim is a computer environment to support collaborative design. It is also a history capturing mechanism for complex corporate activities such as design. We first discuss the use of information modeling to aid these processes and then outline relevant attributes of n -dim. The major part of this paper is to describe some of the applications we have built or are building on top of n -dim, where our long-term goal for each is to test our hypothesis that this system will support collaboration and will capture a usable history better than previous approaches.

Introduction

With the marked improvement in network and computer technology, we see the opportunity to create a new type of computer-based system to support collaboration among distributed teams of persons carrying out a complex process such as design. We also see the opportunity to support the capturing of these processes for subsequent reuse and analysis with a much higher "fidelity" than possible in the past. We base our approach on a system we call n -dim for capturing, organizing and sharing of data and information through the use of network-wide information modeling.

Any time we place a structure over data, we are organizing it [Robertson, et al, 1994]. One example is when we organize files in directories and subdirectories, another is when we put data into databases, and another is when we use document management systems to develop and exchange documents for sign-off during a design process.

The Computer Supported Collaborative Work (CSCW) literature discusses issues on group collaboration and is rapidly developing, with many conferences (for example, see Turner and Kraut [1992]) occurring each year in this area. The formidable database literature is relevant to information modeling, with much current interest in the creation of distributed object-oriented databases [Oszu et al, 1994; Stonebraker, 1994; Kim, 1995]. Within chemical engineering, J. Ponton's group at the University of Edinburgh is developing the *épée* and KBDS systems [Ballinger et al, 1994; Bañares-Alcántara, 1995] to record and organize information developed while designing chemical processes. These systems also capture design intent and track whether designs are meeting stated objectives.

This paper summarizes several design features we have previously described for n -dim [Westerberg et al, 1989; Robertson et al, 1994; see also summary description in Krieger, 1995]. The major part of the paper then describes several of the applications we have created or are creating on top of n -dim.

Information modeling

To understand our approach to information modeling, imagine the existence of two distinctly different worlds in which we would like to organize information. Our first world is an empty one. In this world we first define a number of object types with which we wish to populate it. Our object type definitions are themselves constructed of parts which may be instances

¹. The work reported on here is the product of the n -dim group comprising the following persons: Robert Coyne, Douglas Cunningham, Allen Dutoit, Eric Gardner, Suresh Konda, Sean Levy, Ira Monarch, Robert Patrick, Yoram Reich, Eswaran Subrahmanian, Michael Terk, Mark Thomas, and Arthur Westerberg. All are part of the Engineering Design Research Center except Y. Reich who was but now is in the Department of Soil Mechanics at Tel Aviv University, Israel.

of previously defined types, giving us a powerful part/whole capability in creating new type definitions. To each such object type definition we can send a message to make an instance of itself. These instances contain slots into which we place values that give each instance its individual character. For example, we may create an instance of type "car" and indicate in its roof color slot that it has a blue roof. This world soon becomes filled with numerous instances of these object types. They are strongly organized through recording the parent-child and part/whole relationships among them. Over time we augment our world by defining new types with which we can populate it. We might describe our mind set in creating this first world as one of "patterns first, then instances."

In contrast to starting with no objects, our second world starts with a population of millions of objects of various types already existing. Some of these are text objects created by Word and others by Framemaker; still others are simple e-mail messages. We find numerous "gif" files representing graphic objects. Some are objects defined within databases, such as an entry describing a reflux pump belonging to a column in our plant south of Houston. It is our goal to organize these already existing objects so that we can better understand them. We organize them by establishing a variety of as yet undefined relationships among them. One relationship we might devise is to indicate that this file is the output resulting from having run this other input file through that simulation program, labeling the grouping "Simulation 3 of new methanol process design."

We can further subdivide our first world. There are at least two ways an object can make an instance of itself. In a *class-based* object world, the type definition object contains rules on how to construct an instance of itself. If we later alter any of these rules, the parent type object will seek out and propagate these changes to all instances of it, i.e., to its children. In a *prototype-based* object system, the parent simply makes a copy of itself when creating an instance. The user of this system then edits in the ways this instance is to differ from its parent. If one later edits changes into the parent object, the parent will not seek out and propagate these changes to its children. The more recent literature on object-oriented systems argues that this latter approach more nearly mimics what we wish for design. To see the distinction, imagine we wish to design a new control system for our column. We start by going to our files and copying a blueprint for an earlier design we deem to be close to the one we want. We red-line this copy to create our new control system. In a class-based inheritance system, if someone subsequently alters the blueprint we copied, the system would seek out our copy and alter it to reflect these changes. We might find ourselves enormously annoyed when this happened.

We can augment our second world with the ability to add pattern definitions much like the first when, with experience, we find patterns that should be useful in the construction of future objects. We might describe our mind set for our augmented second world as "instances first, then patterns." Of course, once one has these patterns, we see a completion of the cycle. We can use these to create future objects.

We suggest that the two mind sets, "patterns first then instances" and "instances first then patterns," are profoundly different. This difference is as subtle and important as the distinction between class-based and prototype-based inheritance in object oriented systems. The first world characterizes the mind set behind the creation of object-oriented information systems. It is the mind set behind the creation of centralized database systems. The second world, without the ability to extract patterns, is the mind set behind the creation of the World Wide Web (WWW). The augmented second world is the mind set behind the information modeling system we are creating called *n-dim*.

We view our system as a base on which we and others can add applications. This base provides important functionality to all systems built on top of it, including history maintenance, access control and revision management. As *n-dim* tolerates diverse information objects and any kind of relationship between two objects, the diverse applications we describe below can co-exist if we were willing to have them do so.

n-dim

As we have described in earlier papers on this topic, our work in information modeling is based on experiences that we and others have had on collaborative design projects [Westerberg, et. al, 1989; Robertson, et. al, 1994]. One goal is to *support collaborative work* by allowing the timely sharing of information, allowing users to develop personal and likely very diverse views of that information. A second goal is to *capture the real processes occurring* when an organization carries out a complex activity such as design. Such information can serve as an excellent prototype for those who wish to repeat the same type of activity later and as data to find patterns and establish understanding about the processes really occurring (in other words to develop theories about the design of this type of artifact). We suggest that it is only with this understanding that one can hope to make significant design process improvements.

The underlying premise for our approach is that we believe designers are always developing and using a diverse set of models. They sketch organization charts (a model interrelating the people on the project and in the company), they write quantitative models for the performance of physical artifacts (a simulation model, for example), they draw activity charts to establish critical paths and allocate resources vs. time and so forth. Our aid to support collaborative activities such as design is, therefore, in the form of a general modeling environment, an environment we call *n-dim* (***n* -dimensional information modeling**). One can place relationships over files and their contents, over objects stored in databases, over pages in the World Wide Web (WWW), and so forth, by creating *n-dim* models built of nodes and labeled links. A model contains a list of pointers, displayed as nodes, to anything one can see on the network. Between any two nodes one can place a labeled link

that indicates a relationship between the objects to which the nodes point.

Fig. 1 is an example of a typical model in n -dim. A simulation requiring two input files has produced an output file. Notes in the form of a text file describe this simulation. Each node is actually a pointer not unlike a URL (universal resource locator) on the WWW to a data file, text file or simulation package. Labeled links describe how these objects are related. The simple act of aggregating them implies these particular objects are related to each other, a relationship of type Universal that we call model1.

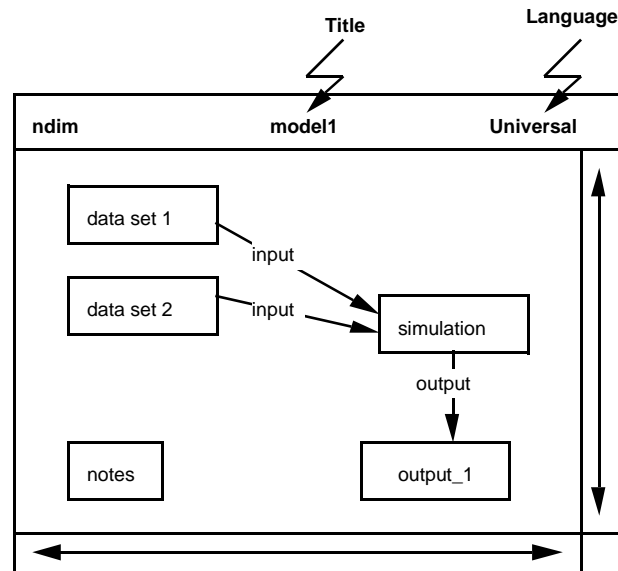


Fig. 1 A typical n -dim model

We summarize several properties/capabilities we have within n -dim.

- **Flat space:** We view all the objects anywhere in the network as existing in a flat space. The WWW adopts a similar viewpoint for all the files in it. It is over these objects that we place structure using n -dim models, which themselves become objects stored in the same flat space.
- **History keeping:** We want this system to record a history of any activity with a better fidelity than we have seen done in the past, keeping track of all communications, all the paths taken, all the mistakes made and so forth. We accomplish our history capture by keeping all the computer-based information the participants find, develop, organize and share while carrying out the project.
- **Private models:** A user prepares new models in his private workspace. Known as private models, the user may alter them at any time with no record kept of such alterations.
- **Published models:** We call the mechanism to keep information permanently in the system *publishing*; the name reflects what happens when one puts an article in the literature. In n -dim, to share an object one must publish it. A published object is immutable. We are free to make copies of it knowing it will not change, ever. In principle it will not go away either.
- **Revision management:** Published items will typically contain errors. We need a mechanism to correct errors. Any object can be designated as a revision of another. We have special operations in the system that allow us to trace revisions quickly.
- **Public models:** We have a specialized form of publishing/revision management that allows several users to share and revise a model. It is similar to what others have referred to as a "sticky" model. A user can create a public model. By controlling access to the operation (also an object) that can alter this model, the user controls who is allowed to alter it. Changes are by adding descriptions of the changes to the end of the internal model description. A user can alter a public model by adding parts to it and/or by apparently removing parts from it. He may do this while someone else is also modifying it. Rules dictate the revision sequence if two changes occur at the same time. Since we record every change made in sequence, we are able to reconstruct and display the revision history for it. Removed parts are in effect placed and then removed whenever the model is recreated for display. The default display for a public model is its latest revision.
- **Access and ownership control:** We are implementing access control mechanisms that allow users to restrict access to and alter ownership of objects. If a user has access, he can see and copy the object. Otherwise he can

neither see it nor copy it. The "owner" of an object can revise its access at any time; he can also transfer ownership to someone else (and later retrieve it if he desires). We are implementing access so that the system maintains a revision history of access and ownership for each published object (e.g., who has had access and when). Our access mechanisms will not defend against malicious behavior any more than the access mechanisms of the UNIX operating systems can.

- **Pedigree management:** If someone uses the n -dim "copy" command to copy an object and its associated operations (prototype-based inheritance), then n -dim automatically adds a link labeled "copy of" between the original and the copy in a special model that keeps the pedigree of that object. Anyone can trace both forward and backward on the pedigree model for any object; however, the system suppresses the display of parts to which a person has no access, appropriately displaying and linking those parts that he can see.
- **Inductive learning:** We want this system to aid users to learn how they should organize their information. This is in contrast to having a team of experts tell the user how he or she may organize information. We introduced the concept of a *modeling language* (ML) to support this learning. A designer may create an instance of a new kind of model he labels an activity model. Another team member may observe this model and like the way it organizes activities. She may start creating such models, agreeing verbally with the originator on the "rules" by which they construct these models. After a while, they have a very good understanding of rules for this model structure. One of them may wish to establish the construction rules more formally to pass them to others. To do this that person can write a modeling language describing how one is to construct an activities model.

A modeling language is itself an n -dim model built of nodes and labeled links. The nodes are pointers to other languages in the system; call them modeling languages ML1, ML2, etc. We may use such a model, call it OurML, as a modeling language to construct a new model which we call OurNewModel. OurNewModel may contain a pointer to any object in the system provided it was constructed using one of the languages ML1, ML2, etc. We may place a labeled link between two objects provided that in OurML such a labeled link connects their modeling languages (e.g., ML1 --(linklabel)--> ML2). We have added mechanisms to state cardinality, "parentheses," and the "or" operator to augment our ability to create modeling languages. An example of cardinality is that we might wish to have precisely one object only constructed using the activities modeling language in OurNewModel, and it may have to have exactly one link emanating from it with the label "DoneBy."

- **Tool integration:** We want n -dim to be an environment within which one can integrate diverse tools. We shall talk more about tool integration later in this paper.
- **Scalability:** We expect there to be billions of objects stored all over the network. Searches for information have to be fast in spite of the amount of the information stored.

We now have a third generation version of n -dim with most of these properties implemented. It is built on top of a prototype-based object system BOS [Levy and Dutoit, 1994a] with an interactive interpreted language to create and manipulate objects called Stitch [Levy and Dutoit, 1994b]. The BOS/Stitch system allows us to implement and test new ideas rapidly.

We are using this version of n -dim in a variety of ways to test our hypotheses that this form of environment can enhance collaborative activities in significant ways and that it will capture enough information to allow us to understand complex activities such as design better. We shall report on the outcome of these tests in a later paper. Here we describe briefly several of the tests that are underway.

Example 1: The Information WEB²

Motivation and Approach

The goal of the Information Web (IWEB) project is to develop an information modeling and management tool specifically designed to meet the needs of software engineers working in a group environment on large projects.

As more people work on a project, the requirements placed on the communication infrastructure which supports their work grows rapidly. The more people there are working on a project, the more difficult it is to maintain a shared understanding of a project's goals, status, design, and implementation. This problem is heightened in a part-time and/or distributed development environment where both the managers and the developers involved in a project often have many other projects or unrelated activities for which they are responsible.

Current environments available to support collaborative software design and implementation are insufficient for the task at hand. Typically, for each project, developers cobble together an ad hoc combination of communication means such as e-mail, electronic bulletin boards, word processing and document preparation tools and a set of disjointed CASE tools which

² Members of n -dim group active on this project: R. Coyne, A. Dutoit.

they use to support their work and communications.

With these requirements in mind, we are experimenting primarily within the context of the project-based software engineering courses in the School of Computer Science at Carnegie Mellon University. These courses reflect much that is typical of the “state of the art” and outstanding issues in the current practice of software engineering. The classes are large (typically 25 to 60 participants), require the delivery of a running prototype to a real client, and generally attempt to provide a realistic software engineering experience to the participants who are generally junior or senior computer science students.

IWEB has been developed on top the n -dim information modeling environment. It is presented here as an example of how computer supported collaborative work (CSCW) applications can be developed using n -dim. In this paper, we have focused on presenting specific modeling examples illustrating a selected number n -dim concepts which IWEB uses; a more detailed examination of IWEB itself is available in Coyne and Dutoit [1994].

IWEB Overview

Given that we found everyday intra- and inter-group communication to be a major bottleneck in the cases we studied [Coyne et al, 1995], most of the core functionality of IWEB is targeted at improving information exchange and capture. On one hand, IWEB provides a gIBIS-like [Yakemovic and Conklin, 1990] Issue Modeling Language for structuring discussions and maintaining lists of unresolved issues. On the other hand, IWEB provides a notification mechanism allowing users to send informal notices to a specific user or group of people. The IWEB notification mechanism is very similar to an e-mail system with the exception that notices may contain references to issue models and/or other n -dim models. Users usually create issue models within an Issue Forest, which contains a set of related issues. One can view an issue forest as a context for discussion. Users may create and reorganize issue forests as their communication needs evolve. Similarly, users can post notices in Notification Boxes, which represent a target audience (e.g. a person, a team or the whole project). Anyone in the project can create a notification box as the organizational structure of the project evolves.

By providing users with both structured and unstructured media for communication, together with means for creating and restructuring contexts in which discussions occur, we hope to increase the amount of information exchanged while decreasing the amount of information lost or misdirected.

Fig. 2 illustrates the use of issue forests and notification boxes. The leftmost window shows an issue forest used by the development team of a project for discussion of development related issues. The rightmost window shows a notification box used for notifying management. The issue forest of the development team contains two issues: Team Formation and Project Deliverables. The description of the Team Formation issue appears at the bottom of the issue forest window, together with its author and the date it was posted. The nodes displayed at the right of the issue are proposals and arguments related to the issue that other members of the development team have posted in response to the issue. The management notification box displays a notice entitled Team Formation Issue which one of the members of the development team created. The notice includes the issue being discussed (displayed in the upper right hand corner of the notification box) and includes a description (displayed in the lower right corner of the notification box). At this point, a management member may select the issue in the notification box and decide to include it in the management issue forest. From there on, the issue will live in both issue forests, i.e. any proposals, arguments or resolutions related to this issue will be seen in both issue forests.

Modeling Examples

Modeling Languages (MLs) & Operations. We implemented the issue forest and notification box MLs using the ML concept of n -dim. For example, the issue forest language is a model containing only those languages which can be included in an issue forest: the issue, proposal, argument and resolution languages. The issue forest language also provides operations for posting new nodes in an issue forest, or including existing nodes from other issue forests. All legal operations on issue forests are similarly implemented as operations in the issue forest ML.

Public Models. Operations on the issue forest and notification box languages ensure that any new issue forest, notification box, notices and nodes created within an issue forest are public (see our description of *public* models earlier), therefore accessible to the rest of the project. Making these models public also ensures that they cannot be removed or modified (only new models can be added), and thus provides a simple history mechanism.

Flat Space. Anyone can refer to issues, issue forests and notification boxes (as with all n -dim models) from more than one model. This allowed us to be able to share an issue across multiple issue forests. This also allows users to build their own models containing references to issues of interest: for example, one user may want to maintain his/her own “hot list” of issues by adding references to them in a private unpublished “current hot list” model.

Access Rights. Access rights allows us to restrict access of certain models by certain users. In the above example, managers may want to allow only managers to access the management issue forest. Separate access control of the operation for a model allows us to control who can add to a forest.

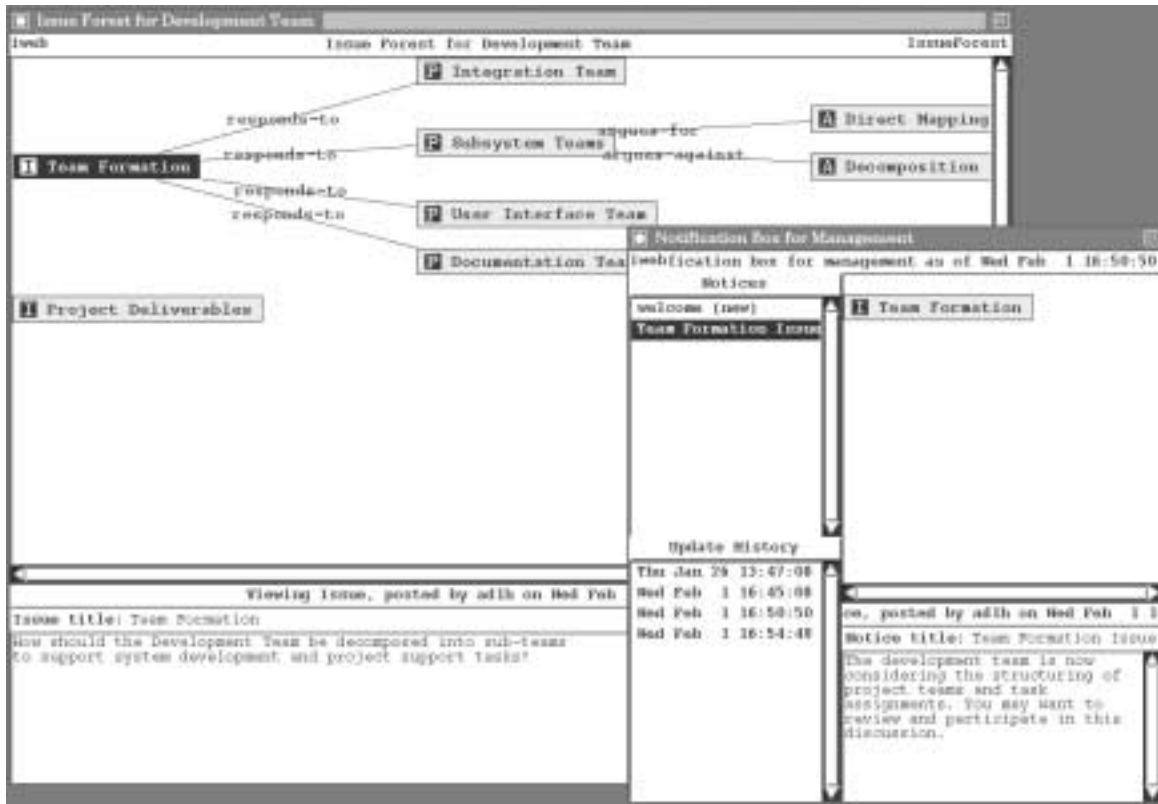


Fig. 2 IWEB Issue Forest and Notification Box

IWEB Status and Future Plans

The communication functions of IWEB have been implemented in n -dim. We are in the process of training about 20 students participating in software engineering classes to use the IWEB rhetorical model for capturing meeting minutes and structuring design rationale documents (weekly information exchange). Once the students are familiar with this mode of working, they will be introduced to the IWEB tool for daily information exchange. We plan to scale the use of IWEB to a full software engineering class by Fall '95 (roughly 60 participants) and hope that the data captured by the use of IWEB will enable us to improve our software engineering program, our software engineering process and the IWEB system itself.

To illustrate the type of information we expect to learn from this study, we analyzed the bulletin board information created by the members of previous classes. We characterized the amount and type of information each group posted vs. the time they posted it. We were able to correlate the type of information exchanged and when it was exchanged during the term to the assessed success of the group. If this correlation is real, it should stand up to future testing, and it would represent one way we can teach and support future groups to improve their performance.

Example 2: An Industrial Engineering Design Support System³

We have worked on a project with ABB to support their process for designing a particular kind of electrical equipment. As a first step, two members of the n -dim group (Subrahmanian and Gardner) visited two of their equipment design groups for several days each, with the purpose of developing an information flow model and an understanding of roles played by all persons involved in their current design processes. They and personnel from ABB developed an information flow model, starting with the original customer order and ending with a final design. It showed several interesting features; for one, in each group someone voluntarily assumed the role of organizing a catalogue of past designs. They saw points in the flow where significant delays occurred in passing information to those who really needed it. A particularly costly example was the delay in discovering a design flaw in the product and having a several month delay in feeding this information back to both the customer ordering personnel and the product designers. The mechanisms were in place; they simply did not function quickly.

³. Members of n -dim group active on this project are: Eric Gardner, Sean Levy, Ira Monarch, Eswaran Subrahmanian

For this project both ABB and the n -dim group selected a project to record the entire design process for this electrical product, a project we felt would test the hypothesis that they could significantly improve their design process if they improved the recording of it. At the present time the electrical product design involves establishing values for several tens of parameters. Many are discrete parameters indicating such things as the type of component to use; others are continuous parameters indicating such things as sizes. ABB captures each of its designs in a design database. A design starts with preliminary values for only a few of the parameters established by interpreting what the customer says he wants. The interpretation reflects the "tacit" information (e.g., colors for painting the product) available from previous contracts with the customer as well as taking numbers directly from correspondence between the customer and the ABB negotiation engineer. After successfully winning a bid to build a piece of electrical equipment for a customer, the negotiation engineer passes a preliminary design to the design manager with his best estimate for values for a few of the parameters, particularly those that specify what it must be capable of doing.

The design manager assigns a design engineer to complete and "optimize" the design. The assigned designer runs any of a number of design programs using input from the current design information. After running a tool the designer may incorporate some of the program output into the design data. The order in which to use the programs is up to the designer, reflecting his past experiences and preferences. Many times the designer has to set parameter values for the design that he thinks will lead to a reasonable design, which, with the running of subsequent tools, prove to be inappropriate. The designer may then reguess values for these parameters and repeat tool invocation following the previous sequence, or he may start down a different path to find a suitable design.

Some designer engineers are markedly more successful than others in creating good designs. Because only the latest design data is available in the design database, there was no record of the design steps taken to create it. Two deficiencies result. First, the process that created a design cannot be passed to someone else to use as a guide for a new design. Second, there is no data on which to form hypotheses relating the design process to the quality of the final design.

For this project we used the history capturing/publishing and language mechanisms of n -dim. The original correspondence between the order engineer and the customer is a record kept within n -dim. Being published it becomes immutable and can never be removed. We provided operators that could dissect these memos, allowing the user to use a mouse to highlight any string of characters within a memo and give that string a label. The labeled part became a "string" or, if appropriate, a "number" object in n -dim. n -dim models automatically captured relationships between that string or number object and the memo from which it came, and, if appropriate, between that number and the data field in the design database where it may have been used as part of the original design specification for the product. The system unobtrusively encourages the order engineer to annotate any of these relationships. This mechanism creates a trail from the customer input to the information put in as preliminary design data. We formulated n -dim languages (which themselves are n -dim models) for each of the types of n -dim models in this process.

We also captured in a similar manner every step taken by the design engineer. Whenever he or a tool he invoked accessed the design data, we added pointers and relationships describing this access into an n -dim model. We did this by placing code in front of the database that sorted out such accesses to it and that passed along information so we could construct appropriate n -dim models. Output put back into the database led to more pointers and relationships to describe it. Thus we could track everything done by a designer in the course of creating a design. Again, in as unobtrusive a fashion as we could, we asked for annotations describing the steps he took. We also provided tools to display to the designer the steps he took. He can point anywhere in this process and ask to repeat the design from that point, leading to branches in the design process. His supervisor can also observe the process and offer suggestions.

The designer can revise any of these models at any time to alter the way he organizes and displays any of the captured information. What a designer cannot do is delete earlier versions of any of the data as n -dim captures it as a part of the history of the design and of the design process. His revisions supply information on how to improve what the system should capture and how it might best organize and display this information as these revisions reflect changes the designer finds useful. We believe this evolutionary improvement is crucial for these systems.

Example 3: A Prototype Engineering Design Support System⁴

In this example we shall look in more detail at the actual objects created for a small prototype design support system that we created to illustrate the capabilities of n -dim. Engineers generate and use large amounts of heterogeneous information during the course of designing an artifact. In this n -dim example, we first look at how a designer might organize all of this information using a design folder model. We then explore, in greater detail, one particular aspect of the design folder in order to demonstrate some of the virtues of the n -dim approach and representation. To give a context to this discussion we have chosen a relatively simple domain: the design of hydraulic cylinders.

⁴ Members of n -dim group active on this project are: Eric Gardner, Sean Levy, Ira Monarch, Eswaran Subrahmanian

Since we have modeled this design environment in n -dim, the ability for multiple designers to collaborate on the design is inherently supported. We require that all relevant models be public using the n -dim publishing mechanism.

We have implemented a "Design Folder" modeling language which allows one to organize many different kinds of information used throughout the design process. We describe the components of the Design Folder model below.

- **Physical Model:** The Physical Model represents a hierarchical physical decomposition of the components of the design artifact
- **Design Database:** This n -dim model provides an interface to the relational database in which one stores all of the design specifications, parameters, and other features.
- **Design and Analysis Toolbox:** This model contains references to the various design and analysis tools the designer can utilize. He can tailor the toolbox specifically for the current design problem. The designers can build a library of many different toolboxes, each for a specific engineering domain or task.
- **Design Constraints:** Constraints are an important part of any design environment. There are n -dim models that provide a grammar for defining constraints relevant to the current design artifact. There is also a ML defined for grouping these constraints into logical classes.
- **Design Rules:** The design environment also supports the creation and use of design rules. Rules can be defined to suggest what action might be appropriate when a designer violates a particular design constraint. We concentrate on this part of the Design Folder to elucidate some of the important capabilities designed into the n -dim system.

The rule ML permits one to write rules of the form if <condition> then <action>. We refer to <condition> as the left hand side of the rule (lhs) and <action> as the right hand side (rhs). The ML specifying the grammar of the lhs of rules allows the combination of constraints using the operators "and" and "or." The constraint ML, in turn, allows one to construct algebraic constraints using the design parameters, real and integer constants, and unary and binary operators.

The rhs in the rule ML is currently modeled simply as string. This string is what the user sees when the lhs of the rule evaluates to false. A logical improvement to this current representation would be to define the rhs as a block of code that gets executed rather than a string that gets displayed.

The rule ML defines several useful operations, one for displaying a particular rule's definition and another for evaluating a rule. The first of these is fairly straight forward and warrants little discussion. The latter operation evaluates the rule in the context of the current values of the design parameters. The system displays a dialog box indicating whether or not the parameters values violate the rule.

Figure 3 shows a view of what the designer's screen might look like. The design folder is the model in the upper left. Notice that it contains parts corresponding to all of the elements of the design folder that we discussed above (i.e., Rules, Constraints, Design DB, Physical Model and Design Tools). The window covering the lower left portion of the design folder shows the interface to the design rules.

The dialog box covering part of the lower portion of the rules display was the result of evaluating the "check stress" rule. Notice that the text in the dialog box indicates that the lhs of the rule is not satisfied. It also contains the suggestion that the designer try adjusting the inside diameter or the wall thickness of the cylinder and then re-invoke the tool that calculate the stress to see if the problem is corrected.

The remaining three large windows show the physical model and two of its components. The CAD drawing associated with one of these components is also shown. In addition to having references to CAD models, the component models contain references to the design parameters associated with that part of the artifact. These parameter references point to the same models referenced in the design database model in the design folder.

Notice that there is a second dialog box displayed. This dialog box is the result of the designer invoking the set value operation on the inside diameter parameter. This was in response to the suggestion given by the "check stress" rule.

Example 4 Design and Manufacturing⁵

ACORN (Advanced Collaborative Open Research Network) is an ARPA funded project within the MADE (Manufacturing Automation and Design Engineering) program. ACORN is a testbed for using the World Wide Web to link together geographically dispersed team members who may reside in several different organizations and to link together diverse design and manufacturing resources. We currently are working with five other universities and eleven companies. Examples of services being provided are:

- University of Michigan: synthesize a computer board

5. Members of n -dim group active on this project: S. Konda, E. Subrahmanian, M. Terk, A. Westerberg. Other active participants include: S. Fenves, S. Finger, both in Civil Engineering.

- University of Pennsylvania: acquire the 3-D shape of a physical object
- University of Utah: supply a mechanical engineering parts catalogue
- MIT: advise on assemblability of a mechanical component
- Alcoa: create a part using stereolithography

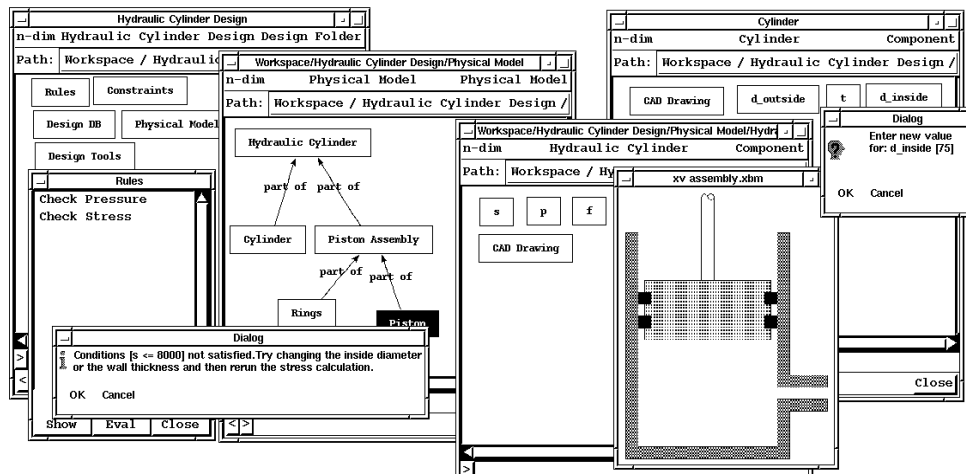


Fig. 3 Engineering Example Screen Shot

We have a problem domain where, as users of this system, we wish to realize many of our n -dim objectives while cooperating with others not in our n -dim world; i.e., in this demonstration we show how we can use n -dim while interacting with external human and computational agents across the WWW. We have encapsulated the "driver" which NCSA Mosaic (a popular WWW browser) uses to exchange pages of information across the web. Our encapsulation identifies and saves WWW pages that pass to and from specific external agents. We have added additional operators to construct n -dim objects from the information in these pages. Thus we are able to record these transactions and convert them to objects we can organize and interrelate within n -dim as we please.

Example 5 Tool integration⁶

We are embedding ASCEND [Piela, et al, 1993] in n -dim. ASCEND is an interactive, equational-based modeling environment. As a tool, it aids its users (called modelers) to create, write, debug, and execute code describing quantitative models based on algebraic and differential equations.

We saw four levels of embedding ASCEND which would be possible: (1) We could put an icon within n -dim which, when double clicked, would open the ASCEND system; control would then pass entirely to ASCEND. (2) We could allow the modeler to create a particular model instance in ASCEND, such as a model of a car. The modeler could create an n -dim "frame" (a data structure containing pairs of attribute names and corresponding values) in which he or she would place names and values for parameters for this model. We could create a translator to convert this to ASCEND input. The modeler could then run ASCEND and generate results which we would translate back into an n -dim frame. (3) We could attempt to support a modeler to create a new model. Input to ASCEND would be code; preliminary output would be diagnostic messages and instances of that code resulting from compiling it. Final output would be debugged code. (4) We could completely rewrite ASCEND to make it a part of n -dim.

The first is hardly a significant form of tool embedding. The second is a reasonable form, but again it is not much of a test of embedding for a complex tool like ASCEND. The last approach is impractical as the ASCEND system involves over one hundred thousand lines of C-code. We, therefore, opted to embed ASCEND for its main purpose: to aid modelers in developing correct equational-based models.

Our original idea on how to do this was to allow a modeler to write ASCEND code visually using the node and link modeling paradigm of n -dim. After trying this approach on several models, we found that one could construct such a model rather easily, but it was virtually impossible for anyone else to understand the result. Even the person creating the code was lost when reviewing it a few hours later. ASCEND code in textual form, on the other hand, is fairly easy to understand. We concluded representing ASCEND code using nodes and links was not the way to embed ASCEND in n -dim, as nice as it

⁶. Members of n -dim active on this project: M. Thomas, A. Westerberg

sounded.

We further noted that *n*-dim is designed to support information handling when designing something, either individually or as a collaborating team. We therefore proposed to use *n*-dim to aid in writing ASCEND code (a design activity) line by line.

ASCEND supports a part/whole approach to modeling. A modeler starts by defining the types of the variables to be used, and creates instances of these types as parts of more complex type definitions. Type definitions themselves are arranged into hierarchies. As an example, a car could be put together by having a car body, an engine, wheels, and so forth, and each of these parts could be built of other parts. Wheels could be organized into *generic_wheels* which are then refined into *spare_wheels* and *normal_wheels*, where the last two each inherit the attributes of *generic_wheels*.

ASCEND also supports deferred binding of types. Deferred binding means a user can include an instance of a type in his or her model and can, at a later time, reach inside that instance and alter the type of one of its parts, provided the alteration is to a more refined type. For example, one could create an array of five *generic_wheels* for the car (including the spare). One *generic_wheel* could be altered to be a *spare_wheel* and the other four to be *normal_wheels*. The consequence is that one may not know the structure of a part simply by looking at its type definition.

To create executable code, one picks a type and asks ASCEND to compile an instance of it.

To support writing and debugging code representing a complex engineering object, the modeler must be able to see inside any of the type definitions in the system. We felt the modeler would be well-served if he or she could also see inside instances of the code he or she is preparing to catch any alterations caused by deferred bindings. Also, the car engine may have four cylinders, and the modeler may want to access attributes for the first one only. This again suggests a need to see inside the instance rather than just inside its type definition. We concluded, therefore, that to support a modeler for coding and debugging, we needed to provide him or her with access to both type definitions and to compiled instances. Our approach was to use the ASCEND system to provide any of these services that it already could, and to supply the rest as operations in *n*-dim.

To use the current embedding of ASCEND within *n*-dim, the modeler writes an ASCEND model by entering a few lines of code. Occasionally, he or she will trigger ASCEND to compile the current version of the model. We have created the ability for the support system to write a test model which contains an instance of the model being created as its only part. This test model is passed to ASCEND to compile, and ASCEND creates a partially compiled data structure (something it does nicely). If the compile step finds it is missing information to create certain instances (for example, the size of an array is unspecified), the support system adds a statement to the test model to prompt the user to put this specification into that model. In addition, *n*-dim models could exist which could contain all the ASCEND types organized into libraries. *n*-dim allows each model to be annotated separately; a modeler could use this annotation to aid in selecting a type.

Whatever part of the data structure the system could compile, the modeler can browse it within ASCEND. The modeler can then use the actual parts in the partially compiled ASCEND models to aid in programming. He or she can open a partially compiled instance to any depth (going into parts of parts of parts etc.) and visually pick a part to be a variable or object for the next line of code in the model being constructed. The system will bring that object name to that line of code, creating the right qualified name for it (from experience, incorrectly qualifying a name is the most common error made by modelers).

We believe this style to be very similar to programming on a spreadsheet---remembering where things are and not what they are called.

The modeler can, in addition, use *n*-dim to manage code revisions, to share models, to link in annotations, and so forth. A complete history of the development of the code is possible. A future addition will be to allow the modeler to use an already existing tool in ASCEND to define frame definitions for a particular model instance, allowing the attachment of ASCEND using the second level of tool integration posed earlier.

Example 6 PhD Notebook⁷

The last example, albeit one we shall only describe very briefly, is for our students to use the *n*-dim environment to keep their PhD notebooks. They are scanning in minutes from meetings, keeping a current bibliography of reference articles, and so forth. The environment allows them to organize the pages of the notebook using a variety of index schemes.

Future Endeavors

There are a number of aspects of information modeling that we have considered or are considering in great depth and intend to incorporate into *n*-dim. Some of these features as they would be manifested in *n*-dim are outlined below.

⁷. Members of *n*-dim group active on this project: D. Cunningham, R. Patrick, and A. Westerberg.

Events and Rules

Information modeling goes beyond static representations of information. While operations (mentioned previously) give users the ability to impose behavior on models, they still fall short of providing a comprehensive environment which users can customize. In order to provide user customization, we intend to incorporate an event/rule mechanism into n -dim. This would most likely be done by using an existing rule-based system such as CLIPS [1994] or RAL [Forgy, 1994]. The basic premise is that a user can establish rules that are invoked when particular events occur in the system. For example, a user may design a rule to have a particular operation invoked when a new issue is added to an issue forest. This gives a user the ability to configure n -dim in a radically different way.

Modeling User Interfaces

The nodes and links representation of n -dim is only one way to view information, and it is not always the most desirable — different views of information structures are often necessary.

In the case of IWEB, for instance, we wrote a separate user interface. We identified two important drawbacks to this exercise: an experienced programmer had to write the interface and someone attempting to use IWEB needs to explicitly load the IWEB user interface code. For this and other reasons we wish to model our user interface toolkit in n -dim such that users can build interfaces in the same manner that they design models. We could associate an interface with a particular model or modeling language. such that, when certain models are opened, the system uses the user-defined interface instead of the default interface.

Modeling Search Criteria

The ability to search for models in an extremely sophisticated matter is a crucial aspect of n -dim. Since one of our main requirements for n -dim is that it be useful in accumulating the information history of an entire organization over time, we anticipate rapidly accumulating a large number of published and public models. Finding specific models or models fitting certain criteria is therefore very critical. In addition, the criteria of a search is in itself useful information that a user may want to keep, publish, revise, etc. Again, the logical solution would be to model search criteria in n -dim.

References

- Bañares-Alcántara, R., "A Process Engineering Design Environment, Based on a Model of the Design Process," invited paper Intelligent Systems for Process Engineering (ISPE'95) - this conference, Snowmass, CO July 9-14, (1995).
- Ballinger, G.H., R. Bañares-Alcántara, D. Costello, E.S. Fraga, J. Krabbe, H. Lababidi, D. M. Laing, R.C. McKinnel, J.W. Ponton, N. Skilling, M.W. Spenceley, "épée: a Process Engineering Software Environment," *Comput. Chem. Engng*, Vol. 18, Suppl., S283-S287 (1994).
- CLIPS Version 5 Reference Manual, COSMIC Project, National Aeronautics and Space Administration (1994).
- Coyne, R. and A. Dutoit, "IWEB (Information WEB): Information Management for Software," EDRC Technical Report EDRC-05-87-94, Engineering Design Research Center, Carnegie Mellon University (1994).
- Coyne, R., A. Dutoit, B. Bruegge and D. Rothenberger, "Teaching More Comprehensive Model-Based Software Engineering: Experience With Objectory's Use Case Approach," *Proc. Conf. Software Engng Education*, New Orleans, Jan. (1995).
- Forgy, C., *RAL Programming Language*, Production Systems Technologies, Pittsburgh, PA (1994).
- Kim, W., *Modern Database Systems*, Addison-Wesley, New York (1995).
- Krieger, J.H., "Process Simulation Seen as Pivotal in Corporate Information Flow," *Chem.&Engng. News*, 50-61, Mar. 27 (1995).
- Levy, S., A. Dutoit, A., "An Overview of the Basic Object System," Unpublished Manuscript, n -dim Group, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA, USA (1994a).
- Levy, S., A. Dutoit, "BOS/Stitch Reference Manual," Unpublished Manuscript, n -dim Group, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA, USA (1994b).
- Oszu, M.T., D. Umeshwar and P. Valduriez, *Distributed Object Management*, Morgan Kaufman, San Mateo, CA (1994).
- Piela, P., R. McKelvey and A. Westerberg, "An Introduction to the ASCEND Modeling System: Its Language and Interactive Environment," *J. Management Information Systems*, vol. 9., 91-121 (1993).
- Robertson, J.L., E. Subrahmanian, M.E. Thomas and A.W. Westerberg, "Management of the Design Process: The Impact of Information Modeling," invited paper at Foundations of Computer Aided Process Design Conference, Snowmass, CO, USA (1994).
- Stonebraker, M., P.M. Aoki, R. Devine, W. Litwin and M. Olson, "Mariposa: A New Architecture for Distributed Data," *Proc. 10th Int. Conf. Data Engng*, Houston, TX, Feb. (1994).
- Turner, J., and R. Kraut (eds), *CSCW '92*, ACM Press, New York (1992)
- Westerberg, A.W., P.C. Piela, E. Subrahmanian, G.W. Podnar and W. Elm, "A Future Computer Environment for Preliminary Design," Keynote lecture. in *Proceedings of FOCAPD*, Snowmass, CO, USA, July 9-14 (1989).
- Yakemovic, K. C. B., and J. Conklin, J., "Report on a Development Project Use of an Issue-Based Information System," *CSCW '90*, pp. 105-118 (1990).