AI GERM: A Logic Programming Front End for GERM

Safaa H. Hashim

AiGerm: A Logic Programming Front End for Germ

Safaa H. Hashim

MCC/STP
Microelectronics and Computer Technology Corporation
Software Technology Program
3500 West balcones Center Drive,
Austin, TX 78759-6509
INTERNET: hashim@mcc.com

Introduction

AiGerm (Artificially Intelligent Graphical Entity Relation Modeler) is a relational database query and programming language front end for MCC/STP's Germ (Graphical Entity Relational Modeling) system. Currently, three versions of AiGerm are in use: Quintus Prolog, BIMprolog, and LDL (MCC's Logical Data Language). AiGerm is intended as an add-on component of the Germ system to be used for navigating very large networks of information, harnessing Prolog or LDL's relational database query capabilities. It can also function as an expert system shell for prototyping knowledge-based systems. AiGerm provides an interface between the programming language and Germ.

When a user starts up AiGerm, the system builds a knowledge base of the currently loaded Germ folio. The knowledge base is a collection of node, link, and aggregate facts. Selecting from the set of commands built in to the AiGerm interface, the user can query the database and run programs that select, create, delete, inspect, and aggregate the nodes and links appearing in the Germ browser.

Aigerm is currently used in MCC/STP's DESIRE system to extract information on the design of code for software systems. Members of the research staff are experimenting with AiGerm in building IBIS-based reasoning and decision support systems for software design and engineering. Rockwell International, an MCC/STP shareholder, is using AiGerm in a simultaneous engineering project.

What is Germ?

Germ (Graphical Entity Relational Modeler) is a graphically-oriented tool for browsing and editing databases. What distinguishes Germ is its conceptual approach in abstracting the elements of a database. Germ uses a few abstractions that we can easily comprehend, remember, and use to create, understand, retrieve, and manipulate database objects. There are two sets of such concepts: basic concepts (also known as object concepts) and interface concepts.

Germ applications are based on an underlying schema file that defines Germ objects and their behavior. The basic object types of the Germ schema are: nodes, links, collections, and aggregates. An application based on a given Germ schema is called a folio; many folios can be based on the same schema. The schema contains the declarations for most of the object concepts in Germ. Embedded in the schema object concepts are properties such as shape, color, attribute types, and so on. Together, the object concepts in a given schema file represent a method for modelling a certain problem, understanding it, and solving it.

Germ's "interface concepts" include a set of window objects: a graphical browser, global view, index window, control panel, inspection window, and editing window, see Figure 1. These objects allow the user to interact with the system to add, delete, update, and retrieve information

represented graphically as nodes and links and to query the database.

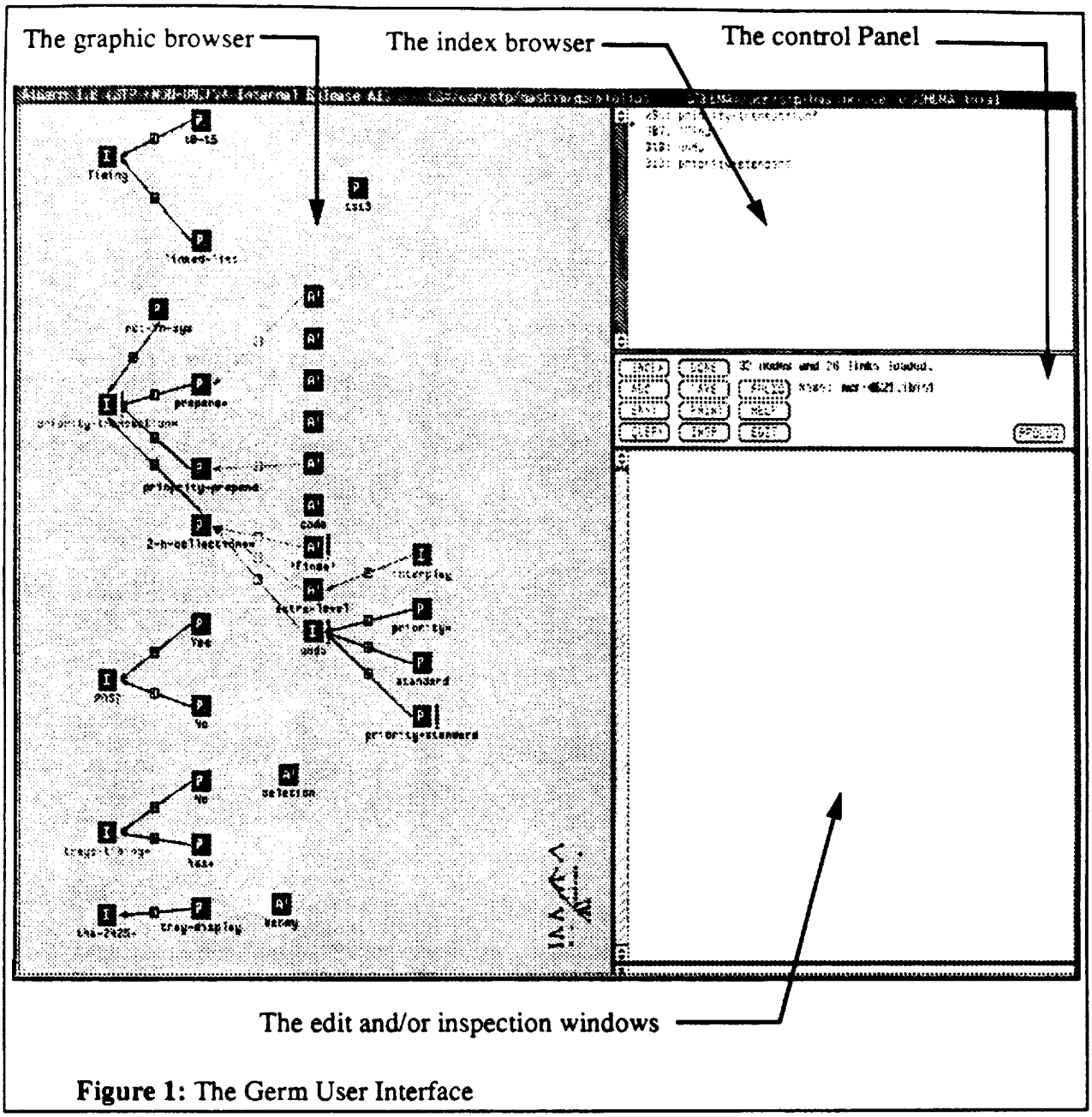

Figure 1: The Germ User Interface

Germ Applications

In its current form, Germ is a generalization of gIBIS. In gIBIS (Conklin & Begeman, 1988), the network of entities represents the argumentation process for understanding and solving a problem using the IBIS methodology. IBIS (Issue Based Information Systems) was introduced by Horst Rittel in the late sixties and early seventies (Rittel & Webber, 1969; Kunz & Rittel, 1970). We have reimplemented gIBIS in Germ just by using a special schema file representation of the IBIS method. The graphic browser in Figure 1 shows a gIBIS network implemented

using a Germ schema file for gIBIS.

Using Germ, researchers and system designers (working individually or in groups) can model systems derived from any method, not just IBIS. In the case of gIBIS, we can have different versions of the gIBIS system that are based on different versions of the gIBIS schema file, each version representing variations in implementing the IBIS method.

Germ can represent both a given model of a method and the database of information on which the method is based. The method can be a design, a problem understanding method, or a problem solving method. Germ will probably be used mostly for building a database representing a problem solving method (problem solving presupposes problem-understanding, and design methods are a special subset of problem solving methods).

Germ is so generalized that it could be considered as a graphical tool that uses geometrical shapes and text to present documents and designs. This is why Some STPer's think of Germ as a "GEometrical Relation Modeler". Germ contains a set of on-line tutorials on Germ usage that were developed using Germ itself. In this case, Germ was used as hypertext-document writing tool (Garrison, Marks and Creemer, 1989). In this article we consider Germ as a modeling tool.

Why AiGerm

Germ has its own query mechanism, which is inflexible for a number of reasons, the two most important ones being:

- Its keyword search combined with regular expression pattern string matching allows only simple queries, like those shown in the preceding section. More importantly, the expressive power of these simple queries is very limited, such that the following simple query is not possible:

  Find the issue with the word "interface" as part of its contents and at least one position responding to it.

  In Prolog, on the other hand, this query would be easily expressed in a single query(goal):

```
|?- issue(I),contents(I,C),substring_of("interface",C),
responds_to(P,I).
```

  Of course, for a practical and real design or an engineering application we would need more complex queries. This requirement, which can be easily met using Prolog, is known as the problem of "structure search" in hypertext (Halasz and Conklin, 1989).

- In addition to richer expressive power in a query mechanism, we need an inference engine, which is a must in the design and engineering tasks of today. The current version of Germ provides no inference engine. With even such a simple one as that in Prolog, we can transform Germ into a powerful knowledge engineering system.

AiGerm is designed to address these two deficiencies in the current query mechanism in Germ. This is why we currently define AiGerm in the following way:

AiGerm = Germ + Logic Programming

A Review of AiGerm

To use AiGerm, the user must have Germ running on a local or a remote machine. Before starting AiGerm, the user must start up Germ and load the desired (hypertext network) folio into the Germ browser. Then, in a shell window, the user would give the command:

```
AiGerm <HOSTNAME>
```

where HOSTNAME is the name of a remote workstation. If no HOSTNAME is given, AiGerm interfaces to the Germ system running on the local workstation. Before actually starting the Prolog process, AiGerm builds a Prolog knowledge base file, see Figure 2.

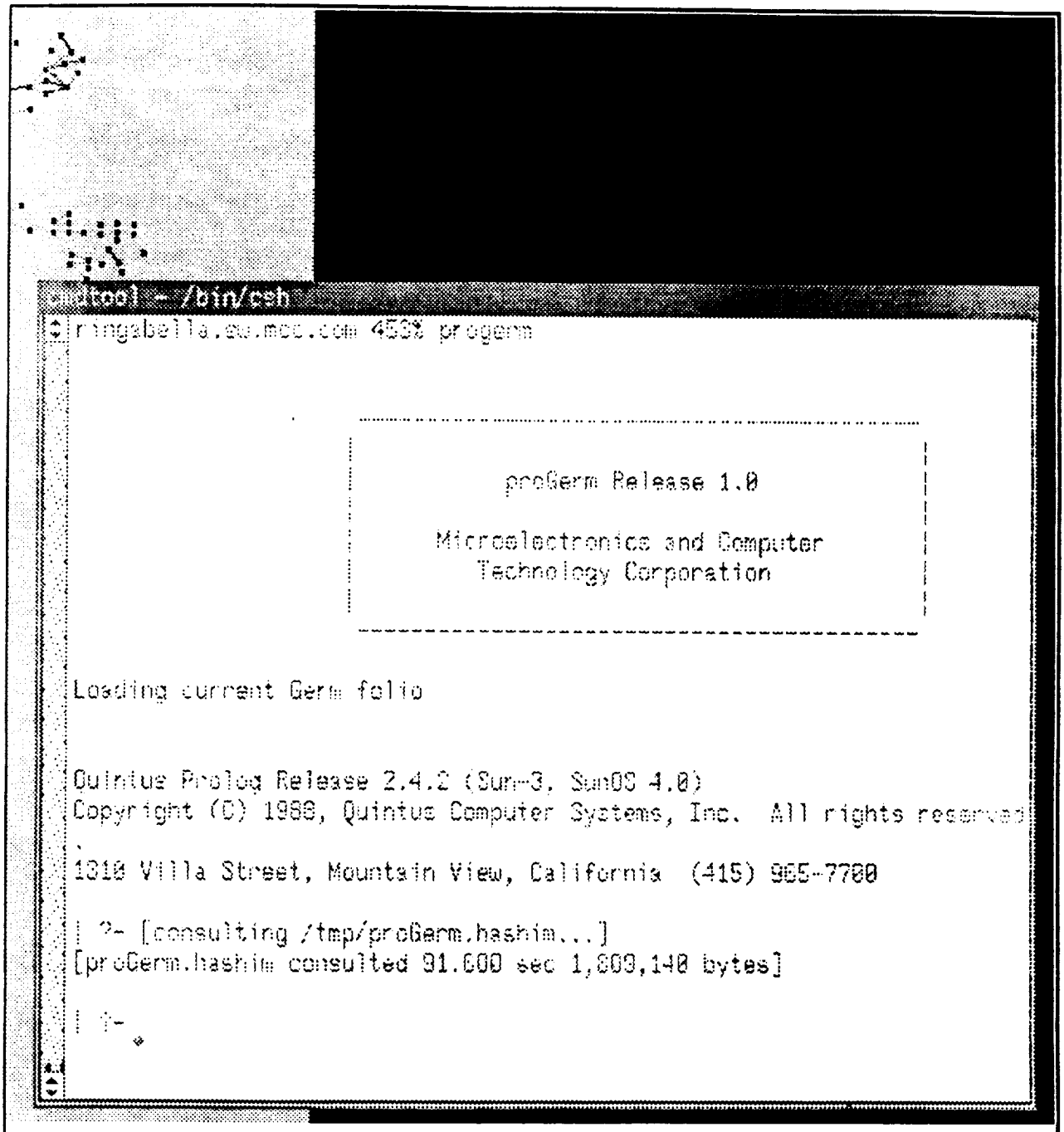

Figure 2: AiGerm: invoking Prolog in a shell window and interfacing it to Germ.

In this knowledge base, for each hypertext entity—i.e. node, link, and aggregate—AiGerm asserts a fact (a prolog clause). Once the knowledge base file is complete, the Prolog process is started and is directed to consult the knowledge base file. When this knowledge base is loaded

into Prolog, nodes, links, and aggregates are represented as Prolog facts, also known as base-relations. The abstract forms of these facts are:

```
node(Eid,[ATTR, ATTR, ...]).
Link(Eid,[EID,EID],[ATTR, ATTR, ...]).
agg(Eid,[EID,EID, ...],[ATTR, ATTR, ...]).
```

As

```
EID = the compound term "eid(INTEGER)"
ATTR = the compound term "attr(TYPE,VALUE)"
```

where TYPE and VALUE are :

```
TYPE = label;author;date;sid,subject;keywords;and so on,
VALUE = STRING;INTEGER
```

Following are examples of a node, a link, and an aggregate, each represented as a fact:

```
node(eid(293),[attr(type,"issue"),
      attr(sid,1),
      attr(date,"Jun 8 10:14 1989"),
      attr(author,"Kemp"),
      attr(label,"Timing"),
      attr(resolved,"yes"),
      attr(contents,"How are timings from multiple trays handled?"),
      attr(x,70), attr(y,36)]).

link(eid(314),[eid(294),eid(293)],[attr(type,"responds-to"),
      attr(sid,-1),
      attr(date,"Jun 8 10:17 1989"),
      attr(author,"Klempay")]).


 aggr(eid(293),[eid(293),eid(295),eid(294)],[attr(type,"AGi")]).
```

Using Prolog to Query Germ Networks

We can query a Germ network directly by issuing goals at the top-level system prompt (|?-). For example, to retrieve nodes one at a time we give the goal:

```
| ?- node(X,List).
```

and Prolog will return the first instance of node that matches this goal, namely:

```
X = eid(7),
List = [attr(type,"issue"), attr(sid,43), attr(date,"May 26 18:23 1989"),
attr(author,"hashim"), attr(label,"theory"), attr('Resolution-due-
date',"Jan 1 1990"), attr('Contents',"^J^JWhat kind of IBIS-theory are we
after?^J")]
```

Retrieving node and link facts is useful but not very interesting. The advantage of Prolog queries over the standard (static) Germ query system becomes apparent when we start giving Prolog sequences of connected subgoals. For example, we can use Germ to model the IBIS method in a way similar to that of the gIBIS system. We would then have a structured hypertext network of issues, positions, and arguments for capturing, say, a group problem-solving or a design meeting session. For real world applications, an IBIS network could have hundreds of nodes and links representing the different issues, positions, and arguments and their relationships. Navigating such large networks is quite difficult if it is done manually. On the other hand, in AiGerm we can use Prolog to query the network for certain nodes and links. For example, we can give this query:

```
| ?- node(X,List), member(attr(type,"issue"),List).
```

meaning that we want to retrieve only nodes that are issues. Moreover, we want to highlight the issue nodes on the browser canvas while retrieving them. To do that we can write this compound goal:

```
| ?- node(X,List), member(attr(type,issue),List), hl_eid(X).
```

`hl_eid` is an add-on (built-in) predicate for interfacing Prolog to Germ. A more interesting goal is to retrieve a more structured set of nodes; for example, to verify that our design discussion satisfies this minimal argumentation subnetwork condition: our IBIS network must have at least one issue with at least one position responding to it, and there must be at least two arguments, one supporting the position and the other objecting to it.

A graph representation of such a subnetwork is shown in Figure 3. Here is the Prolog query for such a structure:

```
| ?- node(X,XNodeAttList),
     member(attr(type,"issue"),XNodeAttList),
     link(L1,[Y,X],LinkAttList1),
     member(attr(type,"responds-to"),LinkAttList1),
     node(Y,YNodeAttList),
     member(attr(type,"position"),YNodeAttList),
     link(L2,[Z,Y],LinkAttList2),
     ( member(attr(type,"supports"),LinkAttList2);
     member(attr(type,"objects-to"),LinkAttList2)),
     node(Z,ZNodeAttList),
     member(attr(type,"argument"),ZNodeAttList),
     hl_eids([X,L1,Y,L2,Z]).
```

The last subgoal, namely the predicate `hl_eids`, takes a list of entity EIDs and highlights (selects) them. Suppose we have a compound goal—that is, a goal made of a sequence of subgoals—that we might need to fire later or use as a subgoal in yet another compound goal. It is worthwhile in such a case to capture a query into a rule (a program) that stands for an executable definition of an "abstraction." This brings us to the subject of abstracting new concepts from existing ones in hypertext networks.

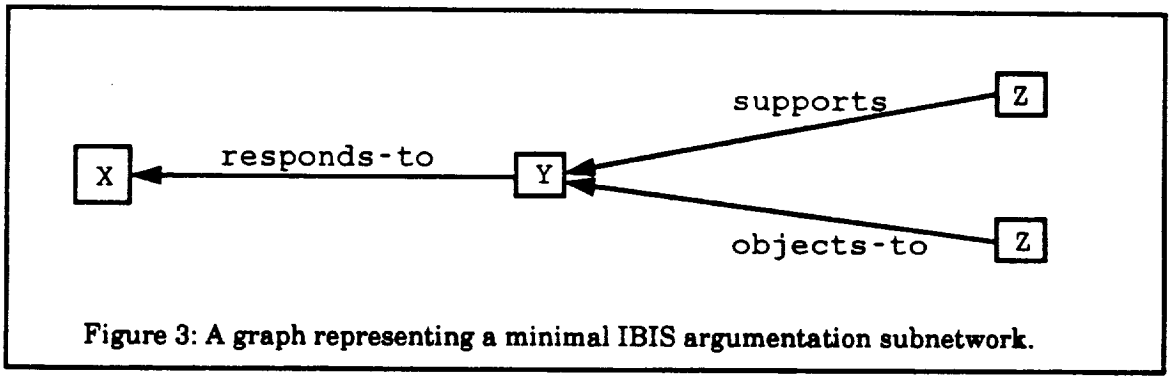

Figure 3: A graph representing a minimal IBIS argumentation subnetwork.

Deriving New Abstraction from Existing Germ Networks and Other Abstractions

Enhancing Germ's hypermedia query and navigation capabilities is not the only advantage of using the logic programming interface in AiGerm. Another advantage is the ability to define new abstractions from the existing pool of base relations and other previously defined abstractions. We say "new abstractions" because Germ itself, through our schema file definition, allows us to have an initial (built-in) set of abstractions on top of the basic node, link, collection,

and aggregate primitives. For example, using a schema file to represent the IBIS method, we usually have abstractions for issues, positions, arguments, and their relationships defined in terms of nodes and links. The knowledge base that AiGerm builds for a Germ network is basically made up of node, link, and aggregate facts. From these facts we can easily define the first level of abstractions as follows:

```
/* ********************** issue ************************** */
/* flow-pattern: (i), (o) */
issue(EID):-
  node(EID,ATTlist),
  member(attr(type,issue),ATTlist).

/* ******************** position ************************** */
/* flow-pattern: (i), (o) */
position(EID):-
  node(EID,ATTlist),
  member(attr(type,position),ATTlist).

/* ******************** argument ************************** */
/* flow-pattern: (i) (o) */
argument(EID):-
  node(EID,ATTlist),
  member(attr(type,position),ATTlist).
```

For the relationships (links) between issue, positions, and arguments, we can define the *responds-to*, *supports*, and *objects-to* relationships in a similar way. For example, here is a definition of the active relationship responds-to between a position and an issue that is supported by Germ:

```
/* ******************** responds_to***************** */
responds_to(P,I):-
  link(_,[P,I],ATTlist),
  member(attr(type, responds-to),ATTlist).
```

What is not supported by Germ is a passive version of responds_to, which we can easily define in Prolog as responded-to-by:

```
/* ***************** responded_to_by *********** */
responded_to_by(I,P):-
  responds_to(P,I).
```

Similarly, we can define *objects-to*, *objected-to-by*, *supports*, and *supported-by* link types.In essence, we can explicate the methodology implicit in a Germ schema file by using such rules. Moreover, we can extend the schema definition in a more flexible way than directly editing and changing the schema file itself. Thus, we can define special modified views of the schema (and thus the methodology represented by the schema) without imposing on other people using the same schema. This ability to modify the representation in such an interactive and dynamic way is a basic aspect of AiGerm.

The abstractions discussed here are just one level above the entity-relation model representation. We can have abstractions that are made up of other abstractions, which themselves are made up of other abstractions, and so on. An example of a system-model using such a multi-level abstracting technique is the following representation of an IBIS-network:

```
% Each IBIS issue must have at least two lines of arg. SL and OL
ibis(I,[SL,OL|REST]):-
     issue(I),
     sup_argLINE(I,SL),      % supporting line of argumentation
     obj_argLINE(I,OL).      % objecting line of argumentation
     ibis1(I,REST).
```

```
ibis1(I,[LINE|REST]):-          % it can have other argumentation lines
     argLINE(I,LINE),
     ibis1(I,REST).
ibis1(_,[]).
```

This definition of an IBIS subnetwork requires that an issue have at least two lines of argumentation, a supporting line and an objecting line. But supporting and objecting lines of argumentation are just special kinds of the argLINE abstraction:

```
argLINE(I,LINE):-
     sup_argLINE(I,LINE).       % a supporting line of argumentation
argLINE(I,LINE):-
     obj_argLINE(I,LINE).       % an objecting line of argumentation
argLINE(I,LINE):-
     cha_argLINE(I,LINE).       % a challenging line of argumentation
```

For the three special lines of argumentation we can have the following definitions:

```
sup_argLINE(I,[P,A|REST]):-
     issue(I),position(P,I),responds_to(P,I),supports(A,P),
     argSEQUENCE([A|Rest]).
obj_argLINE(I,[P,A|REST]):-
     issue(I),position(P),responds_to(P,I),objects_to(A,P),
     argSEQUENCE([A|REST]).
cha_argLINE(I,[I1|REST]):-
     issue(I), issue(I1), suggested_by(I1,I),
     argSEQUENCE([I1|REST]).
```

To complete our sequence of abstractions, we need to define argSEQUENCE, which stands for a sequence of argumentation moves:

```
argSEQUENCE([A,A1|REST]):-
     supports(A1,A),argSEQUENCE([A1|REST]).
argSEQUENCE([A,A1|REST]):-
     objects_to(A1,A),argSEQUENCE([A1|REST]).
argSEQUENCE([A,I|REST]):-
     suggested_by(I,A),argLINE(I,REST).
argSEQUENCE([_]).
```

We believe that such high-level abstractions make navigating Germ networks much easier than navigation with just the basic nodes and links. Also, it makes more sense to talk about related abstractions, such as "a position responding to an issue," than just talking about independent unit abstractions, such as issues, positions, and arguments. For example, issues, positions, and arguments are elements of a discussion or a discourse. Related abstractions form representation structures which we could use to express complex theories and methods. The "ibis" predicate is such a structure that we can use to model the IBIS-based system design process. As a result, we expect that prototypes of system (both software and hardware) engineering applications can be built more efficiently and rapidly using AiGerm's combination of visual modeling in Germ and abstraction-based representation in logic programming. The next section reports on a number of AiGerm-based applications in software engineering and engineering system design.

AiGerm Applications

While we are still in the early stages of experimenting with AiGerm, we feel that it in addition to its use as a relational database query-based hypermedia system, AiGerm could be equally viewed as a general and cost effective tool for prototyping AI-based hypermedia systems. It is this prototyping ability of AiGerm for which we anticipate multiple applications. Currently we are exploring:

1. Reasoning with Issue-Based Design Rationale Networks
2. Analyzing the Structure of Programs
3. The Intelligent Documentation Experiment

Also, researchers at the **Space Systems Division** of **Rockwell International** are currently using AiGerm in developing research prototypes for:

1. QFD Expert System Research
2. Simultaneous Engineering Environment
3. Design Reuse project
4. Design Decision Support prototypes
5. Knowledge Capture
6. Requirement's Analysis (NASP)
7. Payload Mfg. Cost Analysis and Design
8. CAD/CAM Expert system Technology

To illustrate how AiGerm can be used for prototype development, we present two examples in the sections that follow: the "reasoning with IBIS" example and Rockwell's "QFD expert system" example."

EXAMPLE 1: Reasoning with Issue-Based Design Rationale Networks

Although logic programming is based on formal logic, we believe it can also be used for exploring other modes of reasoning, both formal and informal. We have identified four non-mutually-exclusive reasoning methods that we can apply to the IBIS method:

1. a formal reasoning method which builds upon the theory of formal logic and axiomatic (analytic) theory of science
2. an informal reasoning method that builds upon psychology and cognition (J. H. Newman, in Reese, 1980)
3. an informal reasoning method that is based on the theory of informal logic (Blair, 1980)
4. a formal reasoning based on and justified by the theory of dialectical logic, also known as dialogic (Kamlah & Lorenzen, 1984)

Our current work involves formal reasoning of both the first and fourth kind and informal reasoning of the third kind. This paper addresses only the first kind of reasoning—i.e., reasoning in the traditional sense of formal reasoning, and deductive inference in particular. The basis of formal reasoning is logical inference. Inference in general can be deductive, inductive, or abductive. Formal reasoning can be both exact and inexact. Thus, there are exact and inexact rules of logical inference. Here, we consider only exact reasoning. For a formal inexact reasoning approach we have in mind the theory of Fuzzy sets and Fuzzy logic, which deals with inexact or approximate reasoning (Zadeh, 1965, 1979, 1983, and 1985).

In general, IBIS participants raise issues, take positions from the issues, and advance arguments supporting or objecting to the positions. The problem is resolved when the root issue and all other related (major) issues are resolved. Resolving issues involves evaluating (supporting and objecting) arguments to help us find, and thus select, *the most supported and the least objected-to positions.* What we have just said amounts to a *decision-procedure* that we can include in an IBIS- based *decision support system* (DSS). One way to represent such a decision procedure is to use the relational algebraic operation of *quotient*, which we can easily represent in Prolog.

If we have two entities A and B with a respective arity of j and k, expressed as `j > k`, then the quotient, denoted as `A%%B` is a relation with the set of (j-k)-tuples t such that:

```
A%%B = A<1,2,...j-k> -- ((A<1,2,...j-k>) ** B -- A)<1,2,...j-k>
```

The double dash (--) and the double asterisk (**) represent set difference and cartesian product, respectively. To understand "quotient" without the effort of unfolding this complex formula let's use an example. If we have the following relations A and B:

A			
a1	a2	a3	a4
r	s	t	v
r	s	w	x
s	t	w	x
w	v	t	v
w	v	w	x
r	s	vl	w

B	
b1	b2
t	v
w	x

these relations are given in Prolog as the following set of facts:

```
a(r,s,t,v).
a(r,s,w,x).
a(s,t,w,x).
a(w,v,t,v).
a(w,v,w,x).
a(r,s,v,w).
b(t,v).
b(w,x).
```

Then, the quotient expressed in Prolog (a modified version of the one in Li, 1984) is the relation:

```
quotient(A1,A2):-
     group([A1,A2],a(A1,A2,_,_),[A1,A2]),
     setof([AB1,AB2],a(A1,A2,AB1,AB2],Set2),  /* built-in */
     setof([AB1,AB2],b(AB1,AB2),Set1),
     subset(Set1,Set2).
```

`Li defines group` as a *"partitioning relation which conceptually rearranges the relation into groups such that in any one group all tuples have the same value for the grouped attribute."* Thus we can write the following definition:

```
:- dynamic ffound/1.
group(N,G,N):-
  call(G),
  only(N).
only(N) :-
  \+(ffound(N)),
  asserta(ffound(N)).
subset is defined as follows:
subset([H|T],S):-
  member(H,S),
  subset(T,S).
subset([],_).
```

Now, if we try the "quotient" goal, Prolog's response would be:

```
| ?- quotient(X,Y).
```

```
X = r
Y = s;
X = w
Y = v;
no
| ?-
```

Put in a relational form, the result is the relation `a%%b` with two tuples:

a%%b	
ab1	ab2
r	s
w	v

To resolve issues in IBIS, we first need the following relations: responds_to(P,I), *objects_to(A,P)*, *supports(A,P)*, *accepted(A)*, and *rejected(A)*. We have already discussed how to abstract the first three relations in the section on abstractions. Here are the definitions for "*accepted*" and "*rejected*":

```
accepted(Aeid):-
     node(Aeid,AttrList),
     member(attr(acceptance-status,accepted),AttrList).
rejected(Aeid):-
     node(Aeid,AttrList),
     member(attr(acceptance-status,rejected),AttrList).
```

We also define the quotient relations *supports%%accepted*, *supports%%rejected*, *objects_to%%accepted*, *objects_to%%rejected* in the following form:

```
/* positions supported by accepted arguments */
su_quotient_ac(P):-
     retractall(ffound(_)),
     group([P],supports(A,P),[P]),
     setof([A],supports(A,P),Set2),
     setof([A],accepted(A),Set1),
     subset(Set1,Set2).
/* positions supported by rejected arguments */
su_quotient_re(P):-
     retractall(ffound(_)),
     group([P],supports(A,P),[P]),
     setof([A],supports(A,P),Set2),
     setof([A],rejected(A),Set1),
     subset(Set1,Set2).
/* positions objected-to by accepted arguments */
ob_quotient_ac(P):-
     retractall(ffound(_)),
     group([P],objects_to(A,P),[P]),
     setof([A],objects_to(A,P),Set2),
     setof([A],accepted(A),Set1),
     subset(Set1,Set2).
/* positions objected-to by rejected arguments */
ob_quotient_re(P):-
     retractall(ffound(_)),
     group([P],objects_to(A,P),[P]),
     setof([A],objects_to(A,P),Set2),
     setof([A],rejected(A),Set1),
     subset(Set1,Set2).
```

Now we can use these definitions as constraints on selecting a position. A definition that cap-

tures such constrained decision making is the following:

```
selected(P):-
     su_quotient_ac(P),
     \+(su_quotient_re(P)), /* \+ is Quintus-prolog's "not" */
     \+(ob_quotient_ac(P)),
     ob_quotient_re(P).
```

This definition is stated in English as follows:

```
     A position P could be (possibly) selected IF
     it has accepted supporting arguments AND
     none of its supporting arguments are rejected AND
     none of the arguments objecting to it was accepted AND
     it has rejected arguments objecting to it.
```

To try out this definition, we give the following goal:

```
| ?- selected(P).
P = p1;
no
| ?-
```

We can take this definition one step further by considering the possible (or near) resolution of an issue if that issue has at least one selected position:

```
resolved(I):-
     responds_to(P,I),
     selected(P).
```

The above-mentioned decision procedure is only part of an IBIS-based expert system prototype for systems design and analysis. Another part of the system is the IBIS-etiquette adviser shown in Figure 3.

EXAMPLE 2: An Expert System for Implementing the QFD Methods

The *simultaneous engineering* research project at Rockwell International (an MCC shareholder) is an effort to develop tools for supporting the integrated product development process. Simultaneous engineering (SE) is also known as *concurrent engineering* or *integrated product development*. The goal of SE is to model a product development process that results in higher quality and lower cost and that requires shorter time to market than traditional product development systems.

In SE, the different (independent or related) processes of planning, design, manufacturing, testing, and in-service are considered in parallel. The traditional (non-simultaneous) systems engineering approach tackles the different sub-processes sequentially. In many ways, the sequential engineering process has been found to be the main reason for the increase in engineering change orders, the increase in design cycle time, the high manufacturing costs, the increase in scrap and rework situations, and the unnecessary complexity and bad quality of the final product.

The task of SE is to automate the management of planning-to-production processes, taking into consideration the concurrences and cross-functionality of the different processes. Thus, it deals with more than one or two categories or fields of knowledge and expertise. This implies that SE needs more than one method, technology. or instrument to achieve a particular end. These methods or technologies can be alternative, complementary, or independent. In general, we believe that any SE system should allow us to coordinate the competing, or complementing, or interacting methodologies or subsystems.

Quality Functional Deployment (QFD), also known as the *House of Quality* method, is an approach developed by the Japanese to help coordinate the integrated product development pro-

cess (see Hauser, 1988 and Eureka, 1988). The QFD method seeks to diffuse customer-desired qualities (attributes) into a product through the design, specification, parts deployment, process planning, and production planning stages.

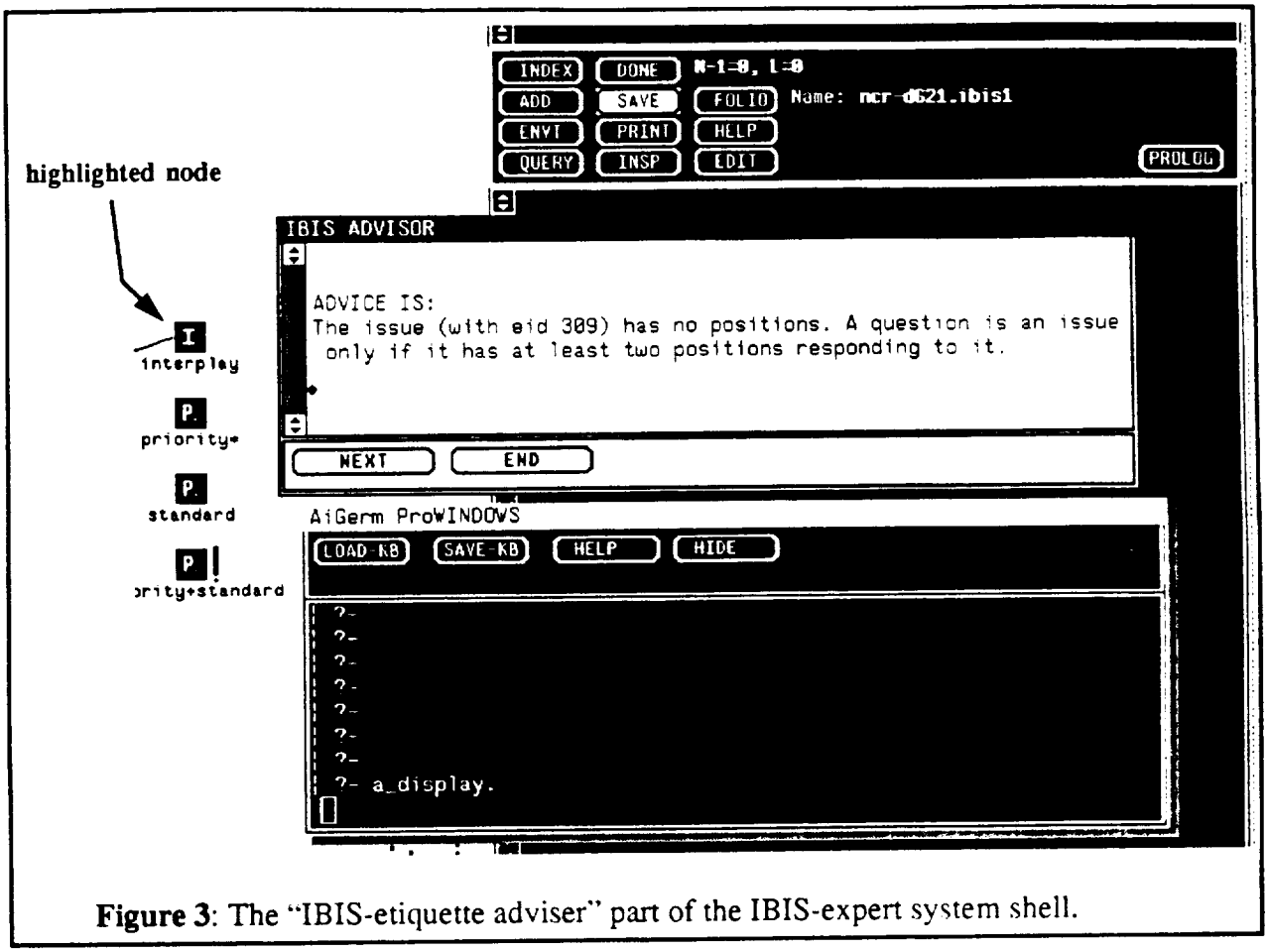

Figure 3: The "IBIS-etiquette adviser" part of the IBIS-expert system shell.

Thus, QFD could serve as a general (and integral) structuring and coordinating part of the SE process. Traditionally, QFD is implemented using linked houses, see Figure 4, with each house being a matrix for relating qualities that convey the customer's voice through to manufacturing. In our case, we want to automate the house-building process and provide decision support for resolving the customer-needs satisfaction issues. One way to look at QFD is to view it as a problem solving process involving a group of participants with different backgrounds—i.e. customers, designers, manufacturing engineers, marketing people, managers, and so on—engaged in a series of discussions trying to resolve different issues. Once we accept such a view, we are tempted to use the IBIS method to represent the QFD-group interactions.

Using AiGerm, we wrote an IBIS-based QFD expert system to help automate the construction of QFD houses. For example, in the case of the first house, the system would elicit needs from customers and help in deriving the engineering characteristics required in the design specifications. Figure 5 shows the network generated in cooperation between the customer and the QFD-expert rules of the system.

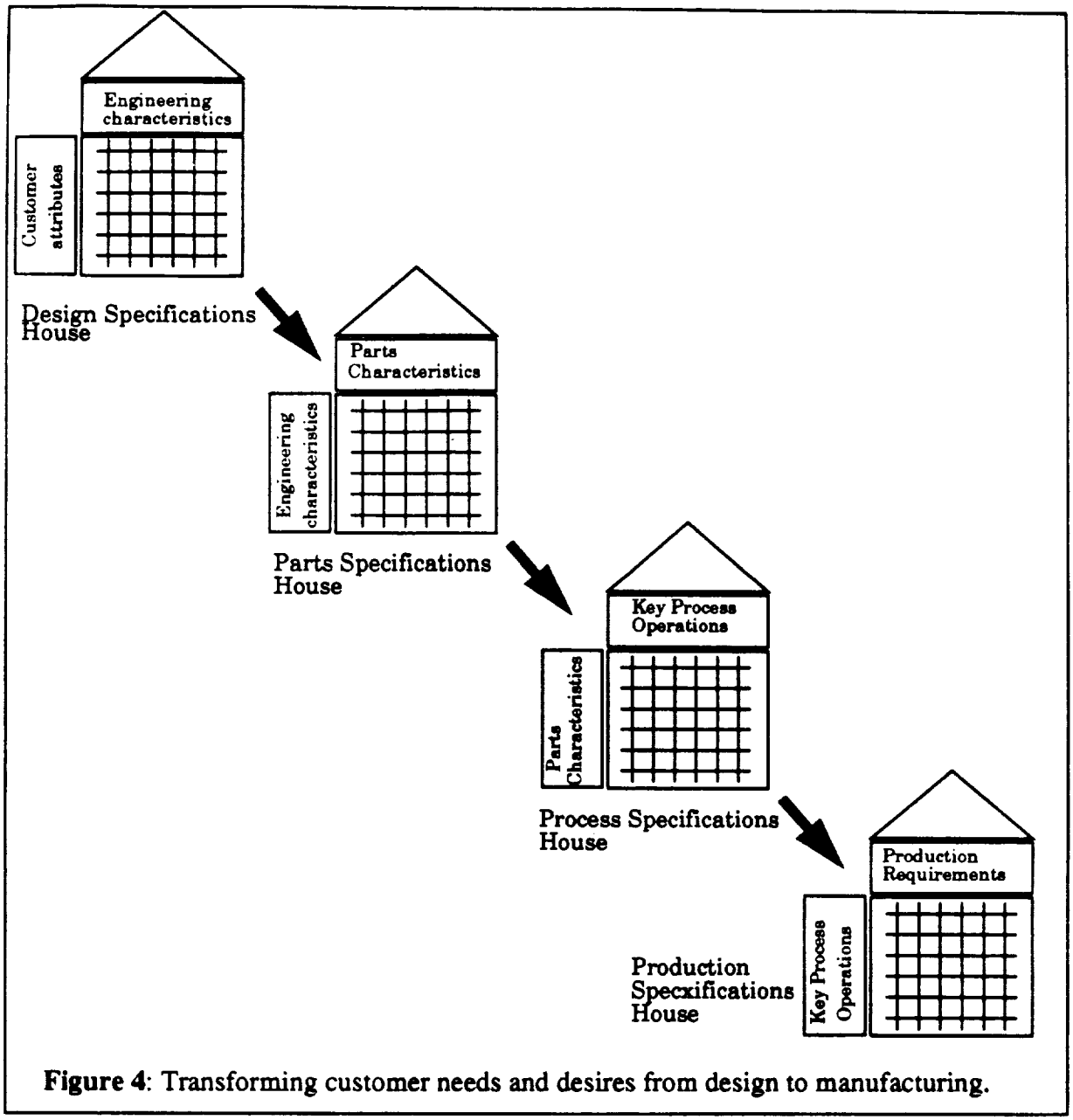

Figure 4: Transforming customer needs and desires from design to manufacturing.

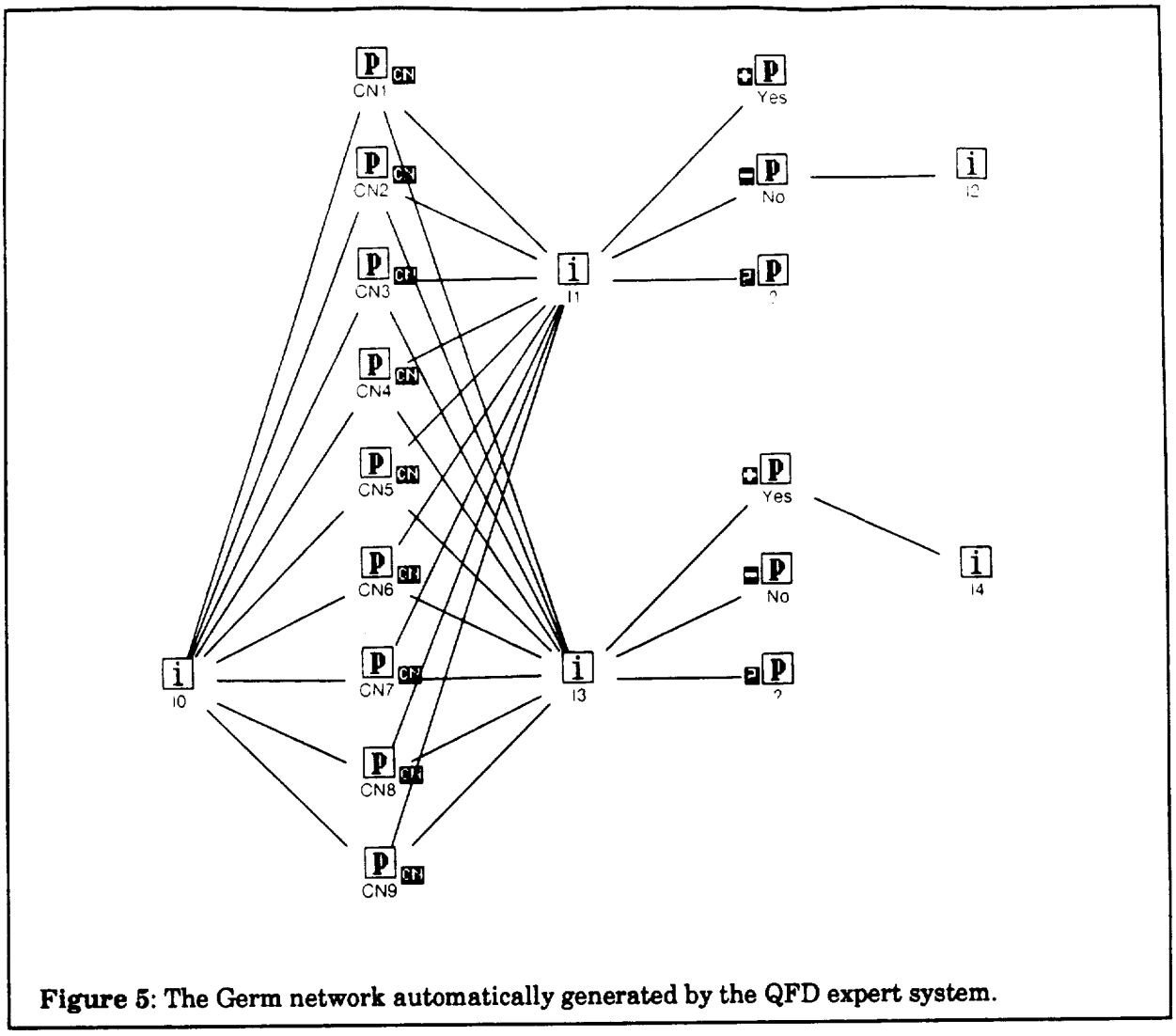

Figure 5: The Germ network automatically generated by the QFD expert system.

Conclusion

AiGerm is MCC's Germ with a logic programming front end. It treats a Germ network as a knowledge base made up of node, link, and aggregate base relations. Users of AiGerm can use Prolog, or MCC's LDL either to navigate Germ networks through queries or to develop prototypes of knowledge-based hypermedia systems. For both applications, we have found that abstractions are the necessary building blocks for any serious use of the system. Currently, AiGerm is used in two major applications, MCC's software design information recovery tool (DESIRE), and Rockwell International's Simultaneous Engineering research project. In conclusion we believe that AiGerm is a cost effective tool for developing and testing systems design prototypes.

Acknowledgments

The author would like to thank Frank Wrabel and the Simultaneous Engineering research team for their many insightful observations during conversations about the research being done at the Space Systems Division of Rockwell International. The author also wishes to thank Noreen Garrison, of STP, for her valuable help in editing the paper.

References

[Blair, 1980] Blair, J. A. and R. H. Johnson. *"Informal Logic: The First International Symposium."* Inverness, California: Edgepress, 1980.

[Biggerstaff, 1988] Biggerstaff, T. J. *"Design Recovery for Maintenance and Reuse."* MCC Technical Report, STP-378-88, November, 1988.

[Bratko, 1986] Bratko, I. *PROLOG Programming for Artificial Intelligence.* Wokingham, England: Addison-Wesley, 1986.

[Clocksin, 1981] Clocksin, W. and C. S. Mellish. *Programming in PROLOG.* Berlin: Springer Verlag.

[Conklin, 1988] Conklin, J. and M. L. Begeman (1988). "gIBIS: A Hypertext Tool for Exploratory Policy Discussion." *ACM Transactions on Office Information Systems*, 6 (October 1988), pp. 303-331.

[Consens, 1989] Consens, M. P. and A. O. Mendelzon. "Expressing Structural Hypertext Queries in GraphLog." *Hypertext '89 Proceedings*, November 5-8, 1989, Pittsburgh, Pennsylvania, pp.269-292.

[Eureka, 1988] Eureka, William E. and N. E. Ryan. *"The Customer Driven Company: Managerial Perspectives on QFD."* Deaborn, Michigan: ASI Press, 1988.

[Halasz, 1989] Halasz F. and J. Conklin. *"Issues in the Design and Application of Hypermedia Systems."* SIGCHI 89, 1989.

[Hashim, 1990a] Hashim, S. H. *"MicroIBIS: A Micro Issue Based Information System."* In: Exploring Hypertext Programming: Writing Knowledge Representation and Problem Solving Programs, Part III." Blue Ridge Summit, PA: Windcrest Books, imprint of TAB BOOKS, 1990.

[Hashim, 1990b] Hashim, S. H. *"WHAT: Writing with a Hypertext-based Argumentative Tool."* MCC/ STP Technical report, No. STP-270-90, 1990.

[Hashim, 1990c] Hashim, S. H. and Mahesh Zurale. *"AiGerm: An Intelligent Graphical Entity relational Modeler."* MCC/STP Technical Report, No. STP-096-90, 1990.

[Hauser, 1988] Hauser, J. R. "The House of Quality." *Harvard Business Review,* May-June, 1988.

[Kamlah, 1984] Kamlah, W. and P. Lorenzen. *Logical Propaedeutic: Pre-School of Reasonable Discourse.* Lanham, MD: University Press of America, Inc., 1984.

[Kunz, 1970] Kunz, W. / Rittel, H. W.J. *"Issues as Elements of Information Systems."* Institute for Urban and Regional Development, University of California, Berkeley, No.131, 1970; also: Institut fuer Grundlagen der Planung, Universiteat Stuttgart S-78-2.

[Li, 1984] Li D. *A Prolog Database System.* England: Research Studies Press, 1984.

[Reese, 1980] Reese, W.L. *Dictionary of Philosophy and Religion, Eastern and Western Thought.* Atlantic Highlands, N.J.: Humanties Press Inc, 1980.

[Rittel, 1980] Rittel, H. W. J. *"APIS - A Concept for an Argumentative Planning Information System."* Institute of Urban and Regional Development, University of California, Berkeley, Working Paper 324, 1980. Also Institut fuer Grundlagen der Planung, Universiteat Stuttgart S-80-2.

[Zadeh, 1980] Zadeh, L. A.. "Inference in Fuzzy Logic." *Proceedings of the International Symposium on Multiple-Valued Logic*, Northwestern University (1980), pp. 124-131.

[Zadeh, 1979] Zadeh, L. A.. "A Theory of Approximate Reasoning." *Machine Intelligence*, ed. D. Michie, American Elsevier, 1979, pp.149-194 .

[Zadeh, 1965] Zadeh, L. A. "Fuzzy Sets." *Information and Control*, 8 (1965), pp.338-353.