BRIDGE LAWS IN HYPERTEXT: A LOGIC MODELING APPROACH

by

Michael Bieber
Computer Science Department
Fulton 430
Boston College
Chestnut Hill, Massachusetts 02167-3808

and

Tomás Isakowitz
Information Systems Department
Leonard N. Stern School of Business
New York University
New York, NY 10003

July 1991

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series

STERN IS-91-17

ABSTRACT

Increasingly, computerized systems tend to delegate certain portions of their functionality to other systems. This is routinely done by systems that use Data Base Management Systems (DBMS) to manage their data. The DBMS is in charge of all data related operations. A similar phenomena is emerging in the area of graphical user-interfaces. As more of these *delegation* phenomena occur, the establishment of flexible communication channels for the different applications becomes increasingly important. We propose to achieve this communication by establishing a set of relationships between the applications. These relationships will be specified by *bridge laws*, i.e. laws that establish bridges between different domains.

We concentrate on a particular example: coupling arbitrary applications to a hypertext user interface. In terms of the discussion above, one of the systems in consideration is fixed. We study the elements that are needed in order to establish effective bridge laws. We do this by defining a general framework and providing two examples. The first example deals with a Data Base Management System, and the second one with a model management system. The examples show that in order to achieve effective interaction between a system and a hypertext interface, some meta-knowledge is required. We extrapolate from our experiments to conclude the type of general properties of bridge laws that are necessary to achieve this high level type of process communication.

Contents

1 Introduction

Increasingly, computerized systems delegate certain portions of their operations to other systems. This is routinely done by systems that use Data Base Management Systems (DBMS) to manage their data. The DBMS is in charge of all data related operations. A similar phenomena is emerging in the area of graphical user interfaces. As more of these *delegation* phenomena occur, the establishment of flexible communication channels for the different applications becomes increasingly important. It is important to develop a communications language protocol between applications. More fundamental, however, is a method for interpreting the objects each application sends to the other using this protocol. We propose to facilitate this interpretation by establishing a set of relationships between applications. These relationships will be specified by *bridge laws*, i.e. laws that establish bridges between different domains. The purpose of this paper is to explore the concept of bridge laws and present a logic model for them.

In this paper we concentrate on a particular example: coupling arbitrary applications to a hypertext user interface. In our future research we shall be extending this coupling to complete "front-end subsystems" besides hypertext interfaces. Implementing a hypertext-based interface is often a complex endeavor. We shall argue that by using *bridge laws* it is possible to establish a general framework that will enable a *reasonably well structured* computerized information system to use a *reasonably well structured* hypertext interface. This framework will include the elements necessary to establish effective bridge laws to support the expression of high level relationships between entities in the application environment and entities in the hypertext environment as well as between activities specific to the application and hypertext activities such as node creation and link traversal.

Bridge laws were briefly introduced in [BK90], in which we presented a logic model for *generalized hypertext*—a dynamic model of hypertext necessary for information systems such as those in this paper's examples. The emphasis of the present paper is not on the model of generalized hypertext, but rather on the concept of bridge laws and how they facilitate the coupling of independent information systems. As such, this paper differs from [BK90] in two major ways. First, we shall be presenting a detailed discussion of bridge laws and their properties. Second, we have simplified the model of hypertext so as not to complicate the discussion and examples needlessly, and then coupled it using bridge laws to an expansion of [BK90]'s example and to another information system domain—database. The bridge laws described in this paper easily could be expanded to incorporate the advanced hypertext generalized features of filtering, and user-supplied comments and links.

This paper is organized as follows. We begin in section 2 with an overview of the hypertext concepts we shall be using in our examples. With this background we ask the question, "Why would the builder of an information system want to couple his or her application to a hypertext interface?" We give examples of usage of bridge laws to incorporate hypertext interfaces to data base management systems in section 3 and to model management systems in section 4. We discuss the bridge law's mechanism and we give evidence to its power and generality in section 5 and we conclude with summarizing remarks and directions for further research in section 6.

2 Back-End Applications and the Front-End Hypertext Interface

Increasingly, designers are turning to *hypertext* as a model for information presentation and user navigation in information systems. Hypertext [Con87, SK89, Nie90] is the concept of linking pieces of information. At its most basic level, a hypertext interface comprises document *nodes* which are interconnected by *links*. Links are marked by *buttons*, portions of text highlighted to indicate a connection to relevant items of information. *Traversing* a link originating in one node is tantamount to generating and displaying the information at its other end.

We refer the reader to [Nie90] for a comprehensive survey of hypertext, to [BI91, Bie90] for discussions of incorporating hypertext in dynamic information system environments and to [BK90, Bie90] for logic models of *dynamic* or *generalized hypertext*.

Why would information system builders wish to couple their systems to a hypertext interface? Hypertext offers information systems a natural and comprehensive means of navigating among its components. Standard features of hypertext systems include browsing and searching mechanisms, *backtracking* to previously-viewed stages in the user's session, user-commenting and other types of annotation via links. Hypertext already has been incorporated into the interfaces of a variety of specialized processes. These include argumentation systems [CB89, SBF$^+$87, GFM89], software engineering systems [GS89, Del86, BR87], legal applications [You89, Wil90], and on-line help functions [AMY88], among others.

Our goal is to develop a way for information system builders to acquire a hypertext interface relatively easily, and to do so by *augmenting* their existing code with bridge laws instead of modifying it. Our approach is to view the hypertext interface and the information system application as two independent modules, the former being the *front-end* and the latter being the *back-end*. The two modules use a communications language to pass information. The application would pass analysis reports, user-input query templates, command sets and the like, to the interface. The interface would pass user responses and requests to the application. Bridge laws are the mechanism that allows the interface to interpret the application elements properly. For example, bridge laws map application commands to hypertext links.

3 Mapping a Relational DBMS to a Hypertext Interface

Let us a consider a hypertext user interface to a relational DBMS. In order to provide a useful interface it is important to consider not only the data base but the entity-relationship (E-R) diagram as well. In the relational model the *entities* and *relationships* that appear in the E-R diagram are represented uniformly by relations. Both entities and relationships present in the E-R diagram are mapped to relations in the relational DATA BASE model. Since there is a conceptual difference between entities and relationships which is important for the user's understanding of the DATA BASE, and since a hypertext front end interacts with the user, it becomes important to retain the E-R diagram information. The interface is set up in such a way that each entity in the E-R diagram is represented by a node in the hypertext subsystem. Whenever there is a relationship between entities, there is a hypertext link between the corresponding nodes in the hypertext front-end.

For example in a database about student housing on a campus there is data about apartments, students, maintenance personnel, etc. For each of these entities a node exists in the hypertext front-end. Each node can represent some (even all) of the tuples in the relation. The contents of each node appear on screen one tuple per line. Relationships among entities are represented by virtual links, as we explain next. Consider the "occupied by" relationship from apartments to students, corresponding to a hypertext link labeled by the same name. The link is *virtual* because the actual *destination* tuples will not be instantiated until the link is traversed at which point a database query is invoked. Each apartment in the *apartments* node is a (virtual) hypertext button connected to an "occupied by" link which when traversed triggers a database query for all students living in the apartment. The query results in a set of students. The hypertext interface processes the set, displaying each element as a button on a separate line of a report.

3.1 Relational Algebra

We will use relational algebra as the formal language to express data base operations since any implementation of a relational data base has to support these in some way or other. Consider, for example, a hypertext interface to a relational data base with information about students, dorm apartments and about the relationship "occupied by" between dorm apartments and students. There are three relations: *studs*, *apts* and *apts-stud* as given in figures 1, 2 and 3. The last one represents the relationship between students and apartments.

Our discussion will focus mainly on querying Data Bases as opposed to constructing and maintaining it. Therefore, we shall only use the projection (π), selection (σ) and join ($\bowtie$) relational algebra operators in the following. For example, in order to find the names of all occupants of the apartment located

studs		
SS	**Name**	**DOB**
546-98-3058	Alberto Cortez	11/15/91
547-54-9807	Richard Gulligan	6/29/71
679-07-6323	Mary Smith	8/30/73
995-67-3211	Jonathan Pressman	7/14/68
...		

Figure 1: The *studs* relation

apts		
Addr	**Type**	**Rent**
2 WSV. # 13-O, New York, NY 10012	2 BDR	730
4717 Broadway# 3F , New York, NY 10004	2 1/2 BDR	560
...		

Figure 2: The *apts* relation

apts−studs	
Addr	**SS**
2 WSV. # 13-O, New York, NY 10012	547-54-9807
2 WSV. # 13-O, New York, NY 10012	995-67-3211
4717 Bway# 3F , New York, NY 10004	679-07-6323
...	

Figure 3: The *apts-studs* relation

at 2 WSV. # 13-O, New York, NY 10012 we evaluate the following expression:

$$\pi_{Name}(\sigma_{Addr=2WSV.\#13-O,NewYork,NY10012}(apts) \bowtie apts\text{-}studs \bowtie studs).$$

3.2 Knowledge about the DBMS

The knowledge about the DBMS needed for the hypertext interface is given here in terms of logic predicates. Predicate names will be in lowercase as will constant and function names. Variables will be uppercased and an appearance of _ in an argument position is a "don't care variable" meaning that it is irrelevant to the point and is therefore left unnamed.

The DBMS supplies the functions *display_value*() and *position_of*() which the hypertext uses to map the data base to the nodes. The first one provides the actual text to be displayed and the second one specifies the location of tuples and values within a relation (it is used to "anchor" buttons). The following are examples of predicates in the context of the *students* and *apartments* data base.

- $entity(students, studs)$, the entity $students$ is represented by the relation $studs$;
- $relation(studs,$ 'Students on campus'$)$, the name of the $studs$ relation is 'Students on campus';
- $attr(ss, studs)$, ss (social security) is an attribute of $studs$;
- $p\_key(ss, studs)$ the primary key for the relation $studs$ is ss;
- $tuple(t_1, studs)$ states that t_1 is a tuple of the $studs$ relation;
- $value($Alberto Cortez$, name, t_1, studs)$, the value the $name$ field of t_1 is "Alberto Cortez".

The value of $display\_value(studs)$ is a string of ascii characters (with line feeds) that contains all tuples in the $studs$ relation, similar to figure 1. The value of $position\_of(t_1, studs)$ in $studs$ is computed as $< 0, 41 >$.

3.2.1 Hypertext objects

The knowledge about hypertext reduces itself to nodes, buttons and links, and it is for these hypertext constructs that the database builder must provide bridge laws. In this section we describe these hypertext constructs and in the next section we present the corresponding bridge laws. A nodes has an *id*, a *name*, a *type* and a *content*. A node is given by the following predicate: $node(Node\_id, Node\_name, Type)$.

Buttons are areas within nodes where links can be accessed. These buttons can be invisible, contain a single letter, a word, a sentence or even the whole node. However, they represent a contiguous portion of a node. Buttons have a unique identifier and information about their location in a node: $button(ID, node\_location(Node\_id, Position))$.

There are two types of links: *static links* and *virtual links*. In the static ones, of the form $link(link\_id, link\_name, source\_button\_id, destination\_node\_id)$, the destination is explicitly specified in the declaration. *Virtual links*, which of the form $v\_link(link\_id, link\_name, source\_button\_id, operation)$, specify a delayed computation of the destination. Instead of a node for the destination of the link, a back-end operation is provided. This operation is performed by the back-end at traversal time to determine the destination of the link. In this paper we shall use only v_link predicates as it is only practical to specify classes of links in a large information system.

3.2.2 The bridge laws

We now specify a set of *bridge laws* that will relate a hypertext interface to the relational DBMS. All variables appearing in the following formulae are assumed to be universally quantified. The first law establishes a node for the relation together with its contents. It also defines the whole relation as a button. Clicking anywhere on the relation will activate a link traversal operation. The nature of this traversal will depend upon the actual links defined.

Bridge law I

$$\begin{aligned}\text{relation(Rel, Name)} \supset\ & node(d\text{-}h(Rel), Name, \textbf{relation}) \wedge \\ & contents(d\text{-}h(Rel), display\_value(Rel)) \wedge \\ & button(d\text{-}h(Rel), node\_location(d\text{-}h(Rel, < 0, size(Rel) >))\end{aligned}$$

The use of *d-h* emphasized a **d**ata base to **h**ypertext translation. It has only mnemonic value and it is meant to remind the reader about the nature of the objects being discussed. The next law is about tuples. These are defined as buttons within nodes. Their display value is obtained from a data base retrieval, their location is obtained from the data base by issuing a call to *position_in*.

Bridge law II

$$\begin{aligned}& \text{tuple(T, Rel)} \wedge \\ & p\_key(Key, Rel) \wedge \\ & key\_val(K, T, Rel) \supset button(\quad d\text{-}h(T, Rel), \\ & \qquad\qquad node\_location(d\text{-}h(Rel), position\_in(T, Rel)))\end{aligned}$$

In order to be able to span across relationships by clicking on the appropriate buttons, we define virtual links for every relationship as follows:

Bridge law III

relationship(E_1, E_2, Rel, Name)$\wedge$

$entity(E_1, R_1)\wedge$

$entity(E_2, R_2)\wedge$

$relation(Rel, \_)\wedge$

$p\_key(R_1, Key_1)\wedge$

$tuple(T_1, R_1)\wedge$

$key\_val(K_1, T_1, R_1)\wedge$

$$\begin{aligned} attributes(R_2, Att_2) \supset \quad v\_link(\quad & rel(E_1, E_2), \\ & Name, \\ & d\text{-}h(T_1, R_1), \\ & operation(\pi_{Att_2}(\sigma_{R_1.key=K_1}(R_1) \bowtie Rel \bowtie R_2))) \end{aligned}$$

The semantics of link traversal relies upon the data base to obtain the actual answer to such an *interactive* query. When traversing a *v_link*, the formula is evaluated which causes a set of tuples to be *shown*. This mechanism is explained at the end of section 4.

Continuing with our example of *students* and *apartments*, the three bridge laws just presented will infer the following nodes and buttons for the students entity:

```
node(       d-h(studs),
            'Students on campus',
            relation)

contents(   d-h(studs),
            "546-98-3058 Alberto Cortez 11/15/91 547-54-9807 Richard Gulligan...")

button(     d-h(studs),
            node_location(d-h(studs), < 0, size of studs relation >))
```

The students in the *studs* relation are represented by tuples $t_1, t_2 \ldots$ which are made into virtual buttons by the second bridge law. Thus, for the first tuple:

```
button(  d-h(t1, studs),
         node_location(d-h(studs), < 0, 41 >))
```

Similarly we have a node and a button for the *apartments* relation and buttons for each tuple therein.

Our example has only one relationship: 'occupied by' which relates apartments to students. This is realized via a virtual link according to the third bridge law. We obtain one such link per tuple in the *apts* relation. One such link is:

$v\_link($ $rel(apartments, students),$

'occupied by',

$d\text{-}h(t_1, apts),$

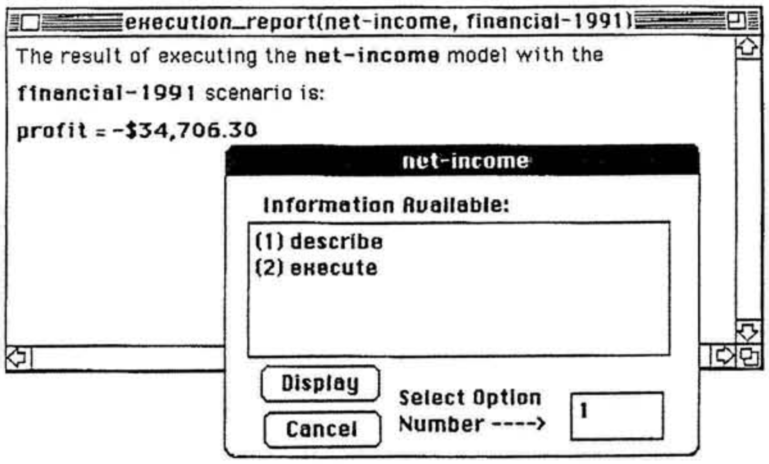
$operation(\pi_{\{ss,name,dob\}}(\sigma_{apts.addr=\text{“2 WSV...”}}(apts) \bowtie apts\text{-}studs \bowtie studs)))$

Traversal of such links is triggered by clicking on the desired tuple in the *apartments* node and selecting the 'occupied by' link. The outcome of this traversal is a new window where the tuples for the students living in the apartment are portrayed.

The hypertext interface we now have is quite powerful. For example **any** relationship will automatically be mapped onto a link, yet it only required three bridge laws to establish. We can easily add more options to the interface such as explaining the meaning of a given field, retrieving the apartment where a student lives, obtaining system data about a relation such as author, date of last update, etc. These can be realized with relative ease with additional bridge laws.

4 Mapping a Model Management System to a Hypertext Interface

In this section we describe the mapping between a model management system back-end and a hypertext front-end interface showing how link traversal is trig-



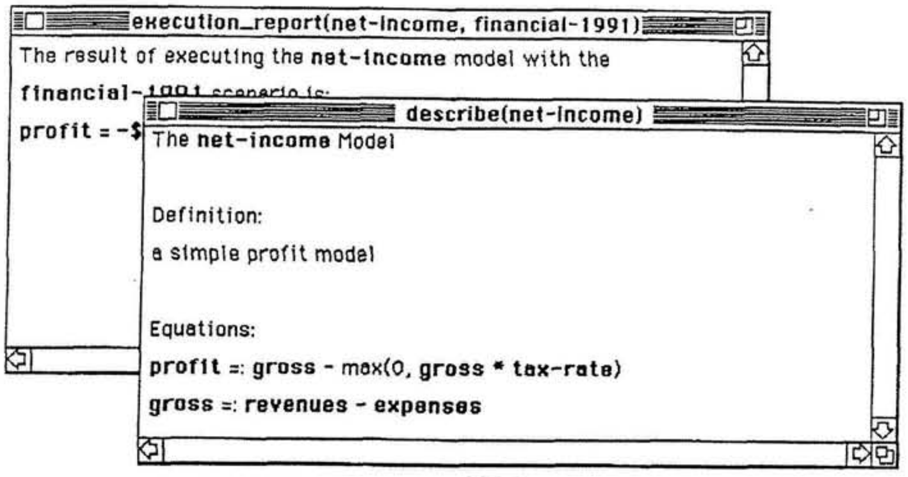
```
execution_report(net-income, financial-1991)
The result of executing the net-income model with the
financial-1991 scenario is:
profit = -$34,706.30
```

```
net-income
Information Available:
(1) describe
(2) execute
Select Option
Number ---->  1
```

Figure 4: Selecting the *describe* link

gered and executed. The model management back-end is based loosely on TEFA [Bha90], which is part of the Max system [KPBB90]. It has a *knowledge base* that contains mathematical models together with their variables, equations, and data scenarios containing datum values for the exogenous variables.

Before formally declaring our environment, in Figure 4 we show a hypothetical screen from the hypertext model management system dealing with a simple *net income* mathematical model. This model has two data scenarios for a company: last year's actual financial figures and this year's projected figures. This screen resulted from the user executing the *net-income* model with the *financial-1991* data scenario, resulting in Figure 4. The boldface text strings represent mouse-sensitive hypertext buttons. Assume the user selects the first button, *net-income*. We see that two possible links are associated with the button, to describe the model or re-execute it. Figure 5 shows the resulting display from describing the model.


```
execution_report(net-income, financial-1991)
The result of executing the net-income model with the
financial-1991 scenario is:
profit = -$
```

```
describe(net-income)
The net-income Model

Definition:
a simple profit model

Equations:
profit =: gross - max(0, gross * tax-rate)
gross =: revenues - expenses
```

Figure 5: Result of traversing the *describe* link

4.1 Basic Model Management System Elements

Just as the data base system had DBMS-specific elements and operations that it had to map to hypertext nodes, links and buttons, the model management system has its own components that the hypertext "front-end" must access. Our model management system stores its elements in a knowledge base of predicates of which the following are examples:

- *model*(*net-income*, *definition*('a simple profit model'))
- *variable*(*profit*, *definition*('annual net income'), *quiddity*(*currency*))
- *variable*(*revenues*, *definition*('annual revenue'), *quiddity*(*currency*))
- *model-equations*(*net-income*, *equations*(

  [$profit =: gross - max(0, gross * tax\text{-}rate)$,

  $gross =: revenues - expenses$]))
- *scenario*(*financial-1991*, *definition*('actual financial figures for 1991'))

- $scenario(financial\text{-}1992, definition($'estimated financial figures for 1992'$))$
- $datum(revenues, scenario(financial\text{-}1991), value(1586731.27))$
- $datum(revenues, scenario(financial\text{-}1992), value(1750000.00))$

As we see from Figure 4 and the discussion below, some of the possible model management system operations are ***describe, execute*** and ***explain***. We shall not declare these here as they are quite complex and vary according to the type of model management system object that the user selects.

4.2 The Model Management System Bridge Laws

The model management builder declares the following bridge laws to map its objects—the mathematical models, variables, scenarios and execution results—to hypertext nodes.

$model(M, definition(Name)) \supset node(M, Name, \mathbf{model})$

$variable(V, definition(Name), Format) \supset node(V, Name, \mathbf{variable})$

$scenario(S, definition(Name)) \supset node(S, Name), \mathbf{scenario})$

$execution(M, V, S, \_) \supset node(exec(M, V, S),$ 'model execution result', $\mathbf{execution})$

These virtual nodes represent the back-end application's "objects of interest" in the front-end interface subsystem.

The interface also treats every document as a node, especially the reports that the model management back-end passes for display. Here are the virtual declarations that generated the two documents in Figure 5.

$$
\begin{array}{ll}
\mathit{model\_equations}(M,E)\wedge & \\
\mathit{first\_variable}(E,V)\wedge & \\
\mathit{execution}(M,V,S,\mathit{Result}) \supset & \mathit{node}(\mathit{execution\_report}(M,S), \text{'execution analysis report'}, \\
& \textbf{document})\wedge \\
& \mathit{contents}(\mathit{execution\_report}(M,S), \\
& [\text{"The result of executing the"}, M, \text{"model with the"}, \\
& S, \text{"scenario is:}\backslash n\text{ "}, V, \text{"="}, \mathit{Result}])
\end{array}
$$

$$
\begin{array}{ll}
\mathit{model}(M,\mathit{definition}(\mathit{Def}))\wedge & \\
\mathit{model\_equations}(M,E) \supset & \mathit{node}(\mathit{describe}(M), \text{'model description report'}, \textbf{document})\wedge \\
& \mathit{contents}(\mathit{describe}(M), \\
& [\text{"The"}, M, \text{"Model}\backslash n\ \backslash n\text{ Definition:}\backslash n\text{ "}, \mathit{Def}, \text{"}\backslash n\text{ Equations:"}, E])
\end{array}
$$

We also have button declarations corresponding to each of the highlighted objects within these documents shown in Figures 4 and 5.

Here are the bridge laws the interface uses to access link operations and determine their traversals. The first law *describes* a model management object. It is valid for all button types. The second *executes* mathematical models when the user selects a button representing one. Implicit is a scenario selection mechanism. The third produces an *explanation* of an execution result highlighted as a button, such as $-\$34,706.30$ in Figure 5.

$button(B, L) \supset$

$v\_link(descr(B), \text{'describe'}, B, operation(\mathbf{describe(B)}))$

$button(m(B), L) \supset$

$v\_link(exec(B), \text{'execute'}, B, operation(\mathbf{execute(B)}))$

$button(exec(M, V, S), L) \supset$

$v\_link(exec(M, V, S), \text{'explanation'}, exec(M, V, S), operation(\mathbf{explain(exec(M, V, S))}))$

Notice that the buttons' location arguments L are left unspecified so that these virtual link declarations apply to all appropriate button instances at any position in any document.

4.3 Link Traversal

How does the hypertext interface associate the button a user selects to the desired virtual link, forward the proper operation label to the back-end for processing and receive the destination document to display? These actions are controlled by the hypertext interfaces *traverse* predicate. The hypertext interface invokes $traverse(button(B))$, when the user selects button B.

$$
\begin{array}{l}
button(B,\_)\wedge \\
collect\_links(B, link\_list(LL))\wedge \\
choose\_link(link\_list(LL), link(L))\wedge \\
perform\_operation(B, OP, Result)\wedge \\
create\_display(Result) \supset \\
\quad traverse(button(B))
\end{array}
$$

The predicate *collect_links*/2 determines all possible links from button B. It "returns" these in the list LL. The predicate *choose_link* generates the user request shown in Figure 4 so the user may choose the link L to traverse. The actual call to the back-end to execute an operation is taken care by the $perform\_operation(B, OP, Result$ predicate. The back-end instantiates the variable *Result*, to the result of performing the operation. The hypertext interface predicate *create_display*/1 transforms the back-end result into the document windows the user sees on the screen, such as those in this paper's figures.

5 Bridge Laws

What are the steps we had to go through in each of the previous examples in order to establish a hypertext interface? First, we identified the key concepts or "objects of interest" in the application domain with which the user may interact, e.g., entities, tuples, relations, mathematical models, variables, scenarios; and then we mapped these into hypertext concepts, i.e., nodes, links and buttons. These two steps are tantamount to declaring a *theory* of what is *important* to the application [BK90]. In a hypertext interface, the important elements are those that the user should be able to access, comment upon and relate to others.

The first step is about discovering the knowledge embedded within a back-end system. It is asking the question, *"What does this system know about?"*. Usually we are not interested in individual data instances, rather about the overall structure of the application, which can be determined by examining its specifications.

The second step is about relating hypertext concepts to application-dependent concepts. For instance, in the DBMS example of section 3 we related schemas to nodes, tuples to lines in nodes, links to relationships and link traversal to database querying. Part of this second step is to determine how the various elements should be displayed to the user. By this we do not mean the visual medium such as windows, etc., but the format of the content (e.g., headers and dimensions) and which buttons it should contain.

Of high importance is the proper establishment of the correspondences between links and the actions in the application domain. This is because the computational power of hypertext is represented by its link traversal operation, hence the only way to represent a *computation* in hypertext is via link traversal.

The beauty of bridge laws is that they take advantage of an application's structure. Instead of an application builder specifying each link among, say, the individual tuples in the database (most of which are added by users after the database system is defined), he or she can exploit ***logical quantification*** and declare a single *general* bridge law for each facet of the application's structure to be represented in the interface.

Bridge laws represent mappings between elements in different domains: in particular the application domain and the hypertext system. In a way, what these laws achieve is to provide a *semantics* for the different objects and operations prevalent in the application domain. A computer system usually provides such meaning via the operational semantics via a menagerie of compilers, linkers, etc. For the purposes of the interface, we provide here a different meaning to the objects of the system, namely the *form* of their expression. There has been considerable debate about the appropriateness of the distinction between form and content [Bro89, Tho90]. We show here the power that is to be obtained by concentrating upon the form.

6 Discussion and Future Research

We have shown in this paper that the complex coordination task of interfacing applications with a hypertext front-end can be achieved by providing knowledge about the relevant objects in the application system based upon the application's internal structure. This knowledge is used to establish a mapping between objects and operations in the application domain, and objects and operations in the hypertext interface. In a similar manner we could couple a hypertext interface to other applications such as expert systems or intelligent tutoring systems. We are investigating how to utilize these concepts to support inter-process communication between arbitrary processes.

Bridge laws allow an application builder to connect two or more distinct environments so that each can take advantage of the other's unique features. As long as each environment makes public the nature of its objects and accessibility paths, skillful individuals should be able to declare bridge laws that will seamlessly provide stable communication between the processes.

References

[AMY88] Robert .M. Akscyn, D. L. McCracken, and E.A. Yoder. KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. *Communications of the ACM*, 31(7):820–835, 1988.

[Bha90] Hemant Bhargava. *A Logic Model for Model Management: An Embedded Languages Approach.* PhD thesis, Wharton School, University of Pennsylvania, December 1990.

[BI91] Michael P. Bieber and Tomás Isakowitz. Valuation Links: Formally Extending the Computational Power of Hypertext. Technical report, CRIS, New York University, 1991.

[Bie90] Michael P. Bieber. *Generalized Hypertext in a Knowledge-based DSS Shell Environment.* PhD thesis, Decision Sciences Department, University of Pennsylvania, Philadelphia, PA 19104, December 1990.

[Bie91] Michael P. Bieber. Issues in Modeling a *Dynamic* Hypertext Interface. Technical report, Boston College, 1991.

[BK90] Michael P. Bieber and Steven O. Kimbrough. On Generalizing the Logic of Hypertext. In *Proceedings of the 23rd Hawaii International Conference on System Sciences*, January 1990.

[BR87] James Bigelow and Victor Riley. Manipulating Source Code in DynamicDesign. In *Hypertext '87 Proceedings*, pages 397–408, New York, NY, November 1987. ACM, ACM Press.

[Bro89] Heather Brown. Standards for Structured Documents. *The Computer Journal*, 32(6):505–514, 1989.

[CB89] Jeff Conklin and Michael L. Begeman. gIBIS: A Tool for All Reasons. *Journal of the American Society for Information Science*, 20(3):200–213, 1989.

[Con87] Jeff Conklin. Hypertext: An Introduction and Survey. *IEEE Computer*, 20(9):17–41, September 1987.

[Del86] N. Delisle. Neptune: A Hypertext System for CAD Applications. In *Proceedings of ACM SIGMOD International Conference on Man-*

agement of Data, Washington, D.C., pages 132–143 (Also available as SIGMOD Record Vol 15, No. 2. June 1986), 1986.

[GFM89] Raymond McCall Gerhard Fischer and Anders Morch. JANUS: Integrating Hypertext with a Knowledge-based Design Environment. In *Hypertext '89 Proceedings*, pages 105–117, New York, NY, November 1989. ACM, ACM Press.

[GS89] Pankaj K. Garg and Walt Scacchi. Ishys: Designing an Intelligent Software Hypertext System. *IEEE Expert*, 4(3):52–64, Fall 1989.

[KPBB90] Sreven Kimbrough, Clark Prichett, Micahel Bieber, and Hemant Bhargava. The Coast Guard's KSS Project. *Interfaces*, 20(6):5–16, November/December 1990.

[Mey89] Norman Meyrowitz. The Missing Link: Why we're all doing hypertext wrong. In Edward Barrett, editor, *The Society of Text*, pages 107–14. MIT Press, Cambridge, MA, 1989.

[Nie90] Jakob Nielsen. *HyperText & HyperMedia*. Academic Press, 1990. A very readable introduction to the field.

[SBF+87] Paul Smolensky, Brigham Bell, Barbara Fox, Roger King, and Clayton Lewis. Constraint-based Hypertext for Argumentation. In *Hypertext '87 Proceedings*, pages 215–246, New York, NY, November 1987. ACM, ACM Press.

[SF89] P. David Stotts and Richard Furuta. Petri net based Hypertext: Document Structure with Browsing Semantics. *ACM Transactions on Information Systems*, 7(1), January 1989.

[SK89] Ben Shneiderman and Greg Kearsley. *Hypertext Hands-On! An Introduction to a New Way of Organizing and Accessing Information.* Addison-Wesley, 1989.

[Tho90] Craig W. Thompson. Strawman Reference Model for Hypermedia Systems. In Judi Moline, Dan Beningni, and Jean Baronas, editors, *Proceedings of the Hypertext Standartization Workshop*, pages 223–246, Gaithersburg, MD 20899, March 1990. National Institute of Standards and Technology, NIST special publication 500-178.

[Wil90] Eve Wilson. Links and Structures in Hypertext Databases for Law. In A. Rizk, N. Streitz, and J. André, editors, *Proceedings of the European Conference on Hypertext*, pages 194–211, France, November 1990. INRIA, Cambridge University Press.

[You89] L. De Young. Hypertext Challenges in the Auditing Domain. In *HyperText-89 Proceedings*, pages 169–180, November 1989.