Using an ER Query and Update Interface for Rapid Prototyping of Hypertext Systems

Bogdan Czejdo
Computer Science Department
University of Houston
Houston, TX 77004

Abstract

In this paper we discuss a method for rapid prototyping of hypertext systems using an Entity-Relationship (ER) query and update interface. Prototyping of a hypertext system for the literate programming is shown as an example. The approach is based on utilizing a relational database management system as a major component of the prototyped system. First, an ER query and update interface is described. Such an interface allows an interactive manipulation of ER database schemas. The sample interactions between a user and the system are given. During the process of manipulation of ER diagrams the information needed to generate an equivalent SQL query is accumulated. Next, a hypertext system for the literate programming is described. The architecture of a prototype of such system is presented. It is shown how the components of a prototype are generated by a simple ER query and update interface.

1. Introduction

A variety of interactive graphical query systems were described in the literature [4, 5, 6]. Several systems have been based on the relational [2] and the Entity-Relationship (ER) data model [1, 3, 7]. They can provide valuable assistance in formulating database queries and updates.

In this paper we discuss a method for rapid prototyping of hypertext systems using a database query and update interface. Prototyping of a hypertext system for the literate programming [11] is shown as an example. The approach is based on utilizing a relational database management system as a major component of the prototyped system. First, an ER query and update interface is described. Such an interface allows an interactive manipulation of ER database schemas. The sample interactions between a user and the system are given. During the process of manipulation of ER diagrams the information needed to generate an equivalent SQL query is accumulated. Next, a hypertext system for the literate programming is described. The architecture of a prototype of such system is presented. It is shown how the components of a prototype are generated by a simple ER query and update interface.

2. An ER Query and Update Interface

Some systems or their components can be conveniently represented as ER diagrams. As an example, let us consider a system for retrieval and update of a University Database. Such database can be described by three entity types FACULTY, STUDENT, and COURSE and three relationship types Advises, IsTaking, and Teaches. It can be represented by a diagram shown in Figure 1.

There are no commonly accepted specifications for hypertext systems [12]. Hypertext systems are good examples of software systems for which experiments with the variety of working models are generally suggested. In this situation a rapid prototyping can be very helpful in the software development process. Rapid prototyping of the working models can be used for requirements validation and behavioral specification of such systems [6].

The paper is organized as follows. In the next section, an ER query and update interface is described. The sample interactions between a user and the system are given. In Section 3 the underlying relational database is discussed and the equivalent SQL expressions are presented. In Section 4 an ER database modeling of a hypertext system is shown. An architecture of a prototype of the hypertext system is presented in Section 5. Also in this section it is shown how the components of a prototype are generated by the ER query and update interface. The Summary presents some conclusions and directions for further work.
When it is applied to a relationship type it removes from the screen the icons corresponding to the specified object and all its attributes. When it is applied to an entity type, it removes from the screen the icons corresponding to the specified object, all its attributes and all relationship types defined on the specified entity type.

2.2 Display

The Display operator displays on the terminal screen the icons corresponding to the attributes of the specified objects, that were hidden previously. The object can be an entity type or a relationship type.

2.3 Move

The Move operator re-positions the icon of a specified object. The object can be an entity or relationship type. The argument list for this operator consists of two elements: the object and a new position of the object.

2.4 Rename

The Rename operator allows a user to change the displayed name of a specified object. The object can be an attribute, an entity type or a relationship type. The argument list for this operator consists of an object and a string describing the new name.

For example, Figure 2 shows the result of applying the operator Hide to some of the attributes shown in Figure 1, invoking the operator Move to the entity type 'STUDENT', invoking the operator Move to the relationship types 'IsTaking' and 'Advises' and renaming 'IsTaking' to 'Takes'.

The second group of operators to specify queries is described below.

2.5 Select

The Select operator allows the user to identify the icons corresponding to the objects that are used in the process of data retrieval. This operator can be applied to an entity type, a relationship type or an attribute. When it is applied to an entity type, the icon corresponding to a selected object is darken on the terminal screen. When it is applied to a relationship type, the icons corresponding to a selected object and all entity types participating in the specified relationship type are darken on the terminal screen. When it is applied to an attribute, the icon corresponding to selected attribute is darken on the terminal screen and the Select operator is invoked recursively for the object the attribute belongs to.

2.6 Restrict_Values

The Restrict_Values operator allows a user to construct a condition that is used in the process of data retrieval. The argument list for this operator consists of three elements: an attribute, a boolean-valued operator and a constant from the domain of the attribute. When this operator is applied, the new condition descriptor is created and displayed on the terminal screen. In addition, the Select operator is invoked for the object the attribute belongs to.

2.7 Duplicate

The Duplicate operator allows to create a duplicate of an icon corresponding to the specified object. This operator can be applied to a relationship type or an entity type.
When it is applied to a relationship type it creates a duplicate of the icon corresponding to the specified object. When it is applied to an entity type it creates a duplicate of the icon corresponding to the specified object and all displayed relationship types defined on the specified entity type.

A user invokes the above described operations by using a pointing device to choose an operator from the menu including Hide, Display, Move, Rename, Select and Restrict value and Duplicate operators. After choosing an operator the user either points to an element of the diagram or enters text in the message area. If the operator is to be applied two or more times in succession, it need not be repeatedly chosen. For example, several entity and relationship types can be removed by just choosing the operator Hide and then pointing successively at the entity and relationship type icons to be removed from the diagram.

As an example let us consider the query: Get the names of all faculty members. Assuming that the diagram of Figure 1 is initially displayed, the query can be specified by the following sequence of user's actions:

1. Point at the operator 'Select'.
2. Point at the attribute 'name' of the entity type 'FACULTY'. As a result of the above actions the icons corresponding to the attribute 'name' and the entity type 'FACULTY' are darkened on the terminal screen.

The resulting diagram on the terminal screen is shown in Figure 3. Then, after invoking the operator 'Execute', the query results are displayed.

Let us consider another query: List the names of faculty members teaching course 'Lasers'. Assuming again that the diagram of Figure 1 is initially displayed, the query can be specified by the following sequence of user's actions:

1. Point at the operator 'Select'.
2. Point at the attribute 'name' of the entity type 'FACULTY'. As a result of the above actions the icons corresponding to the attribute 'name' and the entity type 'FACULTY' are darkened on the terminal screen.
3. Point at the relationship type 'Teaches' ('Select' is still active). As a result of the above action the icons corresponding to the relationship type 'Teaches' and the entity type 'COURSE' are darkened on the terminal screen.
4. Point at the operator 'R_Value'.
5. Point at the attribute 'csname' of the entity type 'COURSE'.
6. Enter the remaining part of the condition (= 'Lasers'). As a result of the above actions the icon corresponding to the condition descriptor is displayed on the terminal screen adjacent to the attribute 'csname' icon.

The resulting diagram on the terminal screen is shown in Figure 4.
The group of operators to specify updates is described below.

2.8 Delete

The Delete operator allows the user to specify the object which tuples are to be deleted in the process of database update. This operator can be applied to an entity type or a relationship type. When it is applied to one of these objects, the icon corresponding to the selected object is redrawn with the double line. In addition, the Select operator is invoked for the selected object.

2.9 Update

The Update operator allows the user to specify an attribute which values are to be modified in the process of database update. The argument list for this operator consists of two elements: an attribute and a constant from the domain of the attribute. When this operator is applied, the new update descriptor is created and displayed on the terminal screen. The icon corresponding to the object the attribute belongs to is redrawn with the blue line. In addition to that, the Select operator is invoked for the object the attribute belongs to.

2.10 Insert

The Insert operator allows the user to construct an 'inserted_value' descriptor that is used in the process of database update. This operator can be applied to an entity type or a relationship type. When it is applied to an entity type, new values for all attributes of the object are requested from the user. Each new value for an attribute can be specified by pointing at an attribute icon and typing the new value. For each attribute of the object the new value descriptor is created and displayed on the terminal screen. For non-key attributes the new value can be null.

When it is applied to a relationship type, new values for all attributes of the relationship type are requested from the user. The new values for the key attributes of the entity types participating in the relationship type are also necessary to identify a relationship. Each new value for an attribute can be specified by pointing at an attribute icon and typing the new value. For each attribute the new value descriptor is created and displayed on the terminal screen.

In both cases the icon corresponding to a selected object is redrawn with the green line. In addition, the Select operator is invoked for the chosen object.

A user invokes the above described update operations similarly to the query operators. If the Delete or Update operation is chosen, then very often the query needs to be specified to determine tuples to be modified or deleted from the database. The discussed previously query operators are used for this purpose.

As an example of an update operation, let us consider the request: Cancel all courses that are taught by the faculty with the name 'Jones'. Assuming again that the diagram of Figure 1 is initially displayed, the update can be specified by the following sequence of user's actions:

1. Point at the operator 'R_Value'
2. Point at the attribute 'name' of the entity type 'FACULTY'.
3. Enter the remaining part of the condition (= 'Jones'). As a result of the above actions the icon corresponding to the condition descriptor is displayed on the terminal screen adjacent to the attribute 'name' icon. In addition to that, the icon corresponding to the entity type 'FACULTY' is darkened on the terminal screen.
4. Point at the operator 'Delete'.
5. Point at the entity type 'COURSE'. As a result of the above action the icon corresponding to the entity type 'COURSE' is redrawn with the double line and darkened on the terminal screen.
6. Point at the operator 'Select'.
7. Point at the relationship type 'Teaches'. As a result of the above action the icon corresponding to the relationship type 'Teaches' is darkened on the terminal screen.

The resulting diagram on the terminal screen is shown in Figure 5. Then, after invoking the operator 'Execute', the update is actually performed.
3. Generation of SQL Query Expressions

The Execute operation is invoked to see the results of the user’s query or to perform an update. In order to describe this operator a mapping of ER schema into an underlying relational schema should be specified [1]. Here, we assume that each entity and relationship type corresponds to a single relation scheme. The relation scheme corresponding to an entity type has the same attributes. The relation scheme corresponding to a relationship type has several (usually two) attributes in addition to those in the relationship type. These additional attributes are foreign keys, namely the primary keys of the relation schemes corresponding to entity types on which the relationship type is defined. As an example let us consider University Database of Figure 1. The ER schema used in this section, can be mapped into the following relational schema:

Faculty (ssno,name,salary)
Student(idnum,sname,addr)
Course(csnum,csname,cshrs)
Advises(ssno, idnum)
IsTaking(idnum, csnum)
Teaches(ssno, csnum)

For the given mapping of an ER schema into a relational schema, it should be possible to generate SQL expressions from any state of the ER diagram. Let us consider the example queries and updates from Section 2 and translate them into SQL expressions for the underlying relational database. The SQL expression generated for the query in Figure 3 is:

SELECT name
FROM Faculty;

The SQL expression generated for the query in Figure 4 is:

SELECT name
FROM Faculty, Teaches, Course
WHERE Faculty.ssno = Teaches.ssno
AND Course.csnum = Teaches.csnum
AND csname = 'Lasers';

The SQL expression generated for the update specification given in Figure 5 is:

DELETE
FROM Course
WHERE csnum IN
(SELECT csnum
FROM Faculty, Teaches, Course
WHERE Faculty.ssno = Teaches.ssno
AND Teaches.csnum = Course.csnum
AND Faculty.name = 'Jones')

The presented above ER query and update interface can generate SQL expressions for any system that can be described by an ER database. Hypertext systems are good examples of such systems.

4. Database Design for a Hypertext System

One current research topic in the area of information systems is concentrated on devising mechanisms to extend the traditional notion of ‘flat’ text to a hypertext presentation [12]. As a result of this extension the text is treated as a structure with complex organizational links allowing direct references from one part of the text to another. Many texts have such links as a natural part of their presentation e.g. computer source programs have a structure which often allows for relatively straightforward division into fragments with connecting semantic links. In particular, the WEB [11] approach to literate programming requires extensive usage of semantic links. An experimental environment for browsing and editing WEB computer source programs has been proposed [9].

An ER diagram modelling the WEB hypertext is presented in Figure 6. This diagram can be easily generalized to accommodate variety of other hypertext systems.

WEB text is divided into modules. Modules are self-contained units, having explicit links to other modules. Modules are the most often used or referenced components of the source program therefore Module is the central entity set in the Figure 6. Each module must have a unique identifier. Most modules have a name. Hence module_number and module_name are attributes of the entity type Module. Each module begins at some particular offset in a file and is usually contained entirely and completely in that one file therefore file_name and displacement are other attributes of the Module entity set. A section can group several modules and therefore section_name is also included in the Module attribute set.

Modules can be parents or children of other modules. This is reflected in the parent/child relationship. A single module may have several modules which are children, and a single child module may have several parent modules. Thus the parent/child relationship is of the type many-to-many. Modules with the same name are related by the relationship type same_name. There can be several modules with the same name, therefore the same_name relationship type is of the type many-to-many.

Modules can be grouped in sections by the same_section relationship type. Each section can contain several modules therefore the same_section relationship is of the type many-to-many.
Identifier is an entity set with the single attribute identifier-name. An identifier can be used several different ways in each module that it is in. It can be defined, referenced or modified. Each of this functions correspond to different relationship type in Figure 6. An identifier can perform the same function in several modules and each module can contain several identifiers and therefore each relationship between identifier and module is of the type many-to-many. line_number is an attribute of each relationship. An identifier can be a variable name or a procedure name.

Topic is an entity type with the single attribute topic-name which is a primary key. The same topic may be in several modules and each module can contain several topics and therefore the relationship type covered_in is of the type many-to-many.

The underlying relational database, called in this paper the ERR database, can be obtained using a standard algorithm [10]. The schema of the ERR database is shown below.

| Module(module_number, file_name, displacement, module_name, section_name) |
| Parent/child(parent_number, child_number, line_number) |
| Same_name(module1_number, module2_number) |
| Same_section(module1_number, module2_number) |

```
Identifier(identifier_value)
Defined_(identifier_value, module_number, line_number)
Referenced(identifier_name, module_number, line_number)
Modified(identifier_name, module_number, line_number)
Topic(topic_name)
Covered_in(topic_name, module_number, line_number)
```

5 Rapid Prototyping of Hypertext Systems

The approach to rapid prototyping of hypertext systems presented in this paper requires three phases. In the first phase, an ER diagram modelling the system is specified and the ERR database is created. In the second phase, the semantics of system operations are given in terms of ER diagram manipulations what is translated automatically by an ER query and update interface into SQL expressions. In the last phase, these SQL expressions (SQL templates) are included as a central component in the architecture of the prototyped system that is shown in Figure 7.
Let us describe each component and the typical flow of information and control among the various components as shown in Figure 7. **ERR Hypertext Database** contains information about all semantic links of the stored texts in the form of relations. **DBMS** is a relational database management system accepting requests (database queries) in SQL language and returning the results in the form of a table. The **Module Browser** performs the most important functions of the hypertext. It accepts browsing requests, converts them into database queries according to SQL templates, receives the information about position of the module (and possibly position within the module) and displays the appropriate text on the terminal screen. The **Module Editor** contained within the **Module Browser** allows for standard screen text editing and upon termination of the edit mode generates appropriate database updates. **ER Interface** allows the user to define new browsing requests and to access directly **ERR Hypertext Database**.

Under the proposed architecture, new semantic links between the elements of the text can be also easily supported by adding new functions defined in terms of ER diagram manipulations. Direct access to **ERR Hypertext Database** allows the user to create module, identifier and topic indexes alleviating the 'lost in hypertext space' problem.

As a case study of this approach let us consider prototyping the WEB hypertext described in Section 4. An ER diagram modelling the system is specified in Section 4. Operations should allow simple navigation of the hypertext space. These operations are associated with the different types of semantic links and allow the user to retrieve and display referenced modules. According to the operation the referenced module can contain the next usage of the selected variable, the definition of the selected variable, the next occurrence of the selected topic etc. These operations, called functions to distinguish them from ER operations, are described below. The semantics of each function is defined by showing an equivalent ER database query.

5.1 Show Module With Selected Name

This function allows the user to display new modules on the basis of the module name. The function 'show module with selected name' has as an argument the module name $M$ (e.g. 'Global Variables'). Assuming that the diagram of Figure 6 is initially displayed, the semantics of this function can be specified by the following sequence of actions:

1. Point at the operator 'Hide'.
2. Point at the icons corresponding to the entity types 'Identifier' and 'Topic' and to the relationship 'parent', 'same_name' and 'same section'. As a result of the above actions, the icons corresponding to the entity types with the names 'Identifier', and 'Topic' and the relationship types with the names 'covered_in', 'defined', 'referenced', 'modified', 'parent', 'same_name' and 'same section' are removed from the terminal screen.

3. Point at the operator 'Select'.

4. Point at the attributes 'file_name' and 'displacement' of the entity type 'Module'. As a result of the above actions the icons corresponding to the attributes 'file_name' and 'displacement' are darkened on the terminal screen.

5. Point at the operator 'R_Value'.

6. Point at the attribute 'module_name' of the entity type 'Module'.

7. Enter the remaining part of the condition (= $M). As a result of the above actions the icon corresponding to the condition descriptor is displayed on the terminal screen adjacent to the attribute 'module_name' icon.

The resulting diagram on the terminal screen is shown in Figure 8.

The equivalent SQL expression is as follows:

SELECT file_name, displacement
FROM Module
WHERE module_name = $M

The result of this query contains file_name and displacement and therefore provides all the necessary information for the execution of this function.

This function (and several to follow) can return a table of several entries, though only the first entry is immediately used. Successive entries in the table can be accessed by use of the 'Repeat' function described later in this section.

It is possible to define some other functions selecting a module on the basis of the module name such as 'show next module with selected name'. The function 'show next module with selected name' has for arguments the current module number $N and the selected name $M. It can be specified by a sequence of actions resulting in the diagram of Figure 9.

Figure 9. The screen for the query performed when the function 'show next module with selected name' is invoked for a module with the number $N and the name $M.

The equivalent SQL expression is as follows:

SELECT file_name, displacement
FROM Module
WHERE module_number > $N
AND module_name = $M

5.2 Show Module with Selected Number.

Some operations relate to modules on the basis of number. The function 'show module with selected number' has as an argument the module number $N. Assuming that the diagram of Figure 6 is initially displayed, the semantics of this function can be specified by a sequence of actions similar to the sequence of actions for the 'show module with selected name' function. These actions are reflected in the diagram shown in Figure 10.

Figure 10. The screen for the query performed when the function 'show module with selected number' is invoked for a module with the number $N.

The equivalent SQL expression is as follows:

SELECT file_name, displacement
FROM Module
WHERE module_number = $N

The result of this query contains file_name and displacement and therefore provides all the necessary information for the execution of this function.

This function (and several to follow) can return a
The equivalent SQL expression is as follows:

SELECT file_name, displacement
FROM Module
WHERE module_number = $N;

It is possible to define some other functions selecting a module on the basis of the module number such as the function 'show next numbered module'. The semantics of these functions can be defined similarly.

5.3 Show Parent Module of the Specified Module

The user of the hypertext can request to move from the specified module to related modules in the parent/child hierarchy. The 'show parent module of the specified module' function allows the user to display a parent module. This function has as an argument the module number $N$. Assuming that the diagram of Figure 6 is initially displayed, the semantics of the function 'show parent module of the specified module' can be specified by the following sequence of actions:

1. Point at the operator 'Hide'.
2. Point at the icons corresponding to the entity types 'Identifier' and 'Topic' and to the relationship types same_name and same_section. As a result of the above actions, the icons corresponding to the entity types with the names 'Identifier' and 'Topic' and the relationship types with the names 'covered-in', defined, referenced, modified, same_name and same section' are removed from the terminal screen.
3. Point at the operator 'Duplicate'.
4. Point at the icon corresponding to the entity type 'Module'. As a result of the above actions, the new icon corresponding to the entity type with the name Module1 is displayed on the screen. In addition to that, two new icons for the relationship types 'parent1' and 'parent11' are also displayed.
5. Point at the operator 'Hide'.
6. Point at the icons corresponding to the relationship types 'parent' and 'parent1'. As a result of the above actions, the icons corresponding to the relationship types 'parent' and 'parent1' are removed from the screen.
7. Point at the operator 'Select'.
8. Point at the attributes 'file_name' and 'displacement' of the entity type Module1 and line_number of the relationship type 'parent1'. As a result of the above actions the icons corresponding to the attributes 'file_name', 'displacement' and 'line_number' are darkened on the terminal screen.
9. Point at the operator 'R_Value'.
10. Point at the attribute 'module_number' of the entity type Module.
11. Enter the remaining part of the condition (= $N). As a result of the above actions the icon corresponding to the condition descriptor is displayed on the terminal screen adjacent to the attribute 'module_number' icon.

Figure 11. The screen for the query performed when the function 'show parent module of the specified module' is invoked for a module with the number $N$

The equivalent SQL expression is as follows:

SELECT Module1.file_name, Module1.displacement, line_number
FROM Module Module1, Parent, Module
WHERE Module1.module_number = Parent.parent_number
AND Parent.child_number = Module.module_number
AND Module.module_number = $N

The result of this query contains file_name, displacement and line_number and therefore provides all the necessary information for the execution of this function. By including the line_number it is also possible to display the place in the parent module where the module $N$ is referenced.

The semantics of the function 'show child module of the specified module' can be defined similarly.

5.4 Show Module Where Selected Identifier is Used.

The programmer can be interested in displaying modules which use some identifier. The function 'show module where selected identifier is defined' has as an argument the identifier name $I$. Assuming that the diagram of Figure 6 is initially displayed, the semantics of this function can be defined by a sequence of actions resulting in the diagram of Figure 12.
Figure 12. The screen for the query performed when the function 'show module where selected identifier is defined' is invoked.

The equivalent SQL expression is as follows:

```sql
SELECT file_name, displacement, line_number
FROM Identifier, Defined, Module
WHERE Module.module_number = Defined.module_number
AND Defined.identifier_name = Identifier.identifier_name
AND Identifier.identifier_name = $I
```

5.5 Show Module Where Selected Topic is Used.

The programmer might want to display modules under the same topic by using the 'show module where selected topic is used'. The semantics for this function can be defined similarly.

5.6 Repeat

Some of the functions listed above return more than one result. An operation to provide the second, third, etc. of those results can be defined as 'Repeat'.

5.7 Define New Link Type

The 'define new link type' operation allows the user to define a new semantic link type and invoke it later by 'user defined operation'. The specification of the new function is done similarly to specifications described in Section 5.1 - 5.4. Once the new function is defined it can be invoked by the 'user defined operation' function.

5.8 Edit

When a module is displayed, it is not initially available for editing. However, the edit mode can be specified, and when editing is finished, the changes can be written to the selected module. The updated text is processed and the ERR database is updated accordingly.

6. Summary

In this paper we have presented a method for rapid prototyping with an ER graphical query and update interface. Some of the prototyped system components are generated by manipulating ER schema diagrams displayed on the terminal screen. Diagrams are transformed until they represent a desired query or update. The user can specify actions in any order. The graphical query interface provides a convenient and dynamically changing frame of reference. Immediate feedback is provided whenever an operator is invalid in the current context. Assistance in both formulating and understanding the query is provided at a higher level of abstraction.

The proposed architecture of a hypertext system allows use of a relational database management system. Mapping between hypertext queries and relational database queries is provided. The system allows the user to define new hypertext operations by providing a corresponding ER diagram. The prototype of WEB and other hypertext systems has been implemented on an IBM PC microcomputer. The current research is concentrated on the design of a data definition language for the ER model and incorporating it in the rapid prototyping process.

References