INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. · Suite 600 · Berkeley, California 94704-1198 · (510) 666-2900 · FAX (510) 666-2956

SchemaDB – An Extensible Schema Database System Using ECG Representation

Manli Li

TR-04-001

September 2003

Abstract

How are our language, concepts and thoughts formed? Schemas as the most primitive conceptual units contribute in forming languages and thoughts. Schemas are studied by linguists, cognitive scientists, psychologists and computer scientists on various emphases. However, there is no existing systematic collection of schemas in a formalized representation. As part of MetaNet Project at ICSI, SchemaDB is an extensible database that aims at not only collecting all existing schemas through a user-friendly, web-based interface, but is also intended for formalizing schema using ECG (Embodied Construction Grammar). SchemaDB is to be used in cataloging, examining, computing metaphor and in many other language and cognitive science studies. The goal of the SchemaDB project is to create a user-friendly web based application in order to collect as many cross-cultural, cross-language schemas as possible in a complete, widely accessible, and human/machine readable manner. Using client/server architecture in addition with PHP (Hypertext Processor) scripting language and the relational database, MySQL, SchemaDB system enables secure interactions between users and the database server.

1. Introduction

The word "Schema" is used differently by different people. Much of the previous work on schemas by other researchers has focused on what are often called Image schemas. These schemas are usually spatial and largely visually based. However, in ECG we also include Motor-Control schemas (or X-schemas), and force-dynamic schemas. The inventory of ECG schemas will also include some Experiencer schemas, though no one has done much work on these.

For ECG, "schema" is the name given to all the basic structured representations of embodied meaning. These schemas are written as a set of role names and various constraints. The roles are intended to represent the linguistically relevant parameters of each schema. Because meaning is assumed to be embodied, it is ultimately grounded in how we experience the world. This means that the hypothesized schemas are intended to be representations of the innate regularities in our perceptual, motor and cognitive systems that structure our experiences and interactions with the world.

The research on schemas certainly does not form a large cohesive or clear picture. This is partly due to different emphases taken by different people. For example, linguists often study the image schemas of a particular word or set of words in a given language. Others study spatial relations in many different languages, looking both at regularities and differences between the languages, but do not necessarily discuss these in terms of schemas. And because there is not really a single clear definition of what an image schema is, research involving schemas may sometimes inconsistent because people are really talking about different things.

---

2. Background

There has been similar work (MetaDB) done by Michael Meisel[3] in relation to the MetaNet project at ICSI. The MetaDB is a database of metaphors, including information about source domain, target domain and the specific role mappings between the two domains. The structure of schemas is based on ECG and is similar to what is in the SchemaDB. MetaDB is able to input schema role names and role types and to have sub-case-of (similar to parent in SchemaDB) relations between schemas. However the evoke relations between schemas or bindings between the schemas' roles are not yet implemented. Consequently, the important relations between schemas cannot be captured.

In the following sections, we are going to put emphasis on the internal structures of the SchemaDB system, the user interface, security management, system core implementation and system software requirements.

---

3. Internal Structures

3.1 System Architecture:

In this section we will describe the internal structure of the SchemaDB application. First of all, in order to design a web application, server/client architecture becomes a necessity. In the SchemaDB system, the server/client model is designed such that any internet enabled client (a browser) can issue request to the web server and the database server for information. The following System Architecture figure shows a complete loop of requesting and retrieving information from the database system.

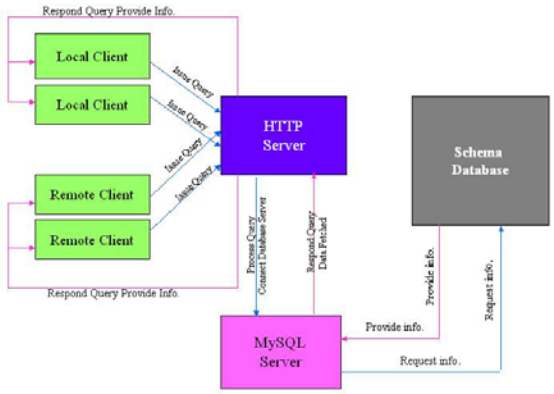


Figure 3.1 System Architecture

3.2 System Data Structure:

The data structure design is based upon the structure of ECG representation of schemas. There are six tables that are related to schema; the one table used for user management will be described in detail in a later section on security and user management. The overview is shown in the following figure:

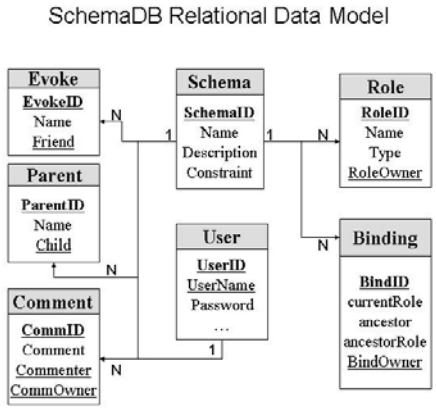


Figure 3.2 SchemaDB Model

3.2.1 Schema Table and Role Table

In the data model, we have one table called "Schema" which contains basic information about a schema, including its id, name, a brief description and its constraints; the schema id and name serve as references to facilitate linking with other tables. For each schema, the ECG representation consists a set of *Roles*, roles are the elements required to define a schema. Each role has a certain *Type,* which restricts what kind of object/entity a role could be. Therefore, we have another table called "role", which consists of the *id* of a role, its *type* and its *owner*. The *roleOwner* refers to the *id* of the schema in the "schema" table. This role belongs to the schema whose *id* is shown in the "role" table. Thus Schema and Role has a 1 to N relationship. Internally, the role table looks like the following figure:

id	rName	rVal	roleOwner
39	Region1	region	26
40	Region2	region	26
44	transition1		10
45	state		10
56	dim1extent	distance	34
58	dim3extent	distance	34
57	dim2extent	distance	34
52	regionA	region	6
53	regionB	region	6
59	Region2	boundedRegion	37
60	Region1	boundedRegion	37
54	boundary	line	6
63	region2	boundedRegion	40
64	inside	region	38
65	outside	region	38
66	inside	substance	41
67	surface	substance	41
68	boundedEntity	entity	41
70	mainAxis		42
71	secondaryAxis		42
72	tertiaryAxis		42
79	tertiaryAxis		46
83	sides	substance	7

Role table

3.2.2 Parent table and Evoke table

Every schema except the top-level schema are naturally inherited from some other schema, meaning that each schema will have one or more "Parents"; this leads to the creation of table *Parent*. Similarly, when a schema is defined using part of another schema, we call that schema an evoked schema, so we create a table named "*Evoke*". The relationships between Schema and Parent or Schema and Evoke are both 1 to N relationships. The following figures show the internals of table Parent and table Evoke.

The *child* column in table Parent refers to the *id* from table Schema. For example, schema *boundedState* has the child schema *initiallyBounded,* which possesses the id #10. To explain the necessity of such structures, we have to evoke a more precise example. Let's

take *closedBoundary* schema as an example; the schema *closedBoundary* has to be defined

id	pName	child
21	space	33
2	boundedState	10
9	boundary	12
19	space	26
22	space	34
30	dimensionalEntity	46
11	entity	7
25	boundary	38
18	entity	21
20	space	28
23	space	36
24	space	37
28	space	5

Parent Table

id	eName	friend
25	boundedRegion	40
24	encircle	38
3	division/whole	5
4	figure/ground	5
5	force-dynamic	7
6	space	7
11	experiencer	12
10	boundedRegion	26
37	boundedEntity	7
12	time	29
13	time	8
14	state	9
16	transition	9
18	distance	34
19	region	6
20	boundedRegion	37
22	space	6
26	space	40
27	closedBoundary	5
29	boundedRegion	41
30	entity	41

Evoke Table

using schema *Boundary*, but partially defined using *Encircle*. The reason is rather simple—a *closedBoundary* is one specific kind of *Boundary*, without the definition of *Boundary*, we cannot define a specific type of *Boundary*. Thus, the existence of *Boundary* determines the existence of *closedBoundary*, which is analogous of a parent-child relationship. However, we do not need the complete set of characteristics of *Encircle* to define the schema *closedBoundary*. It does not need the existence of *Encircle* to be properly defined, yet *closedBoundary* does retain certain characteristics of schema *Encircle*. Hence we say *Encircle* is an evoked schema of *closedBounday*.

3.2.3 Binding table

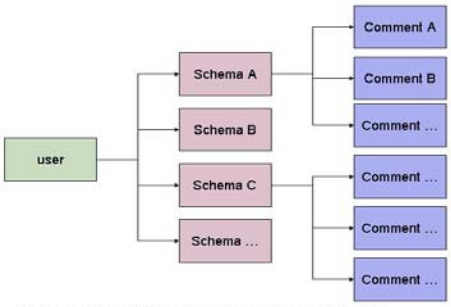
The subtle difference between a parent schema and an evoked schema leads to the issue of binding. Binding is another essential element of ECG representation. Binding means the roles of a schema are linked to those of its ancestors, and have the *possibility* to be linked in evoked schemas. Our example *closedBoundary* is a schema that is described as "the boundary is a closed curve or surface, and it encloses one of the regions". Schema *Encircle* is described as "a topological relation between two bounded regions in which one of the regions forms a closed curve or closed surface, and the other region is within the bounded region defined by this closed curve/surface. The two regions are not coincident." Obviously, schema *closedBoundary* carries partial properties of schema *Encircle*, namely one of the roles of *Encircle*, *region1* of *Type closedRegion*. From the description, it is easy to see that the role of *closedBoundary* "inside" is really the same as the role "*region1"* of *Encircle.* Therefore, we can bind these two roles. We use the symbol "<==>" as operator to represent the binding relationship and "*schema.role*" notation to represent the operands on the two ends (on the user interface). In this case: *closedBoundary.inside<==>Encircle.region1*

id	currentRole	ancesstorRole	bind0wner	ancesstor
9	objWhole	figure	boundedRegion	figure/ground
11	objWhole	whole	boundedRegion	division/whole
66	edge	boundary	boundedRegion	closedBoundary
72	edge	division2	boundedRegion	division/whole
73	background	ground	boundedRegion	figure/ground
74	background	outside	boundedRegion	closedBoundary
60	boundary	boundary	closedBoundary	boundary
82	inside	region1	closedBoundary	encircle
81	outside	regionB	closedBoundary	boundary
80	inside	regionA	closedBoundary	boundary

Binding Table

3.2.4 Comment table

Besides the core tables used in ECG representation we described above, another table named "comment" is established for the purpose of users' convenience. It consists of the *id* of a comment, the content of the comment (which is restricted to a length of 255 characters), and the username of the commenter. Any user comments on any schemas will be inserted into this table with the reference of schema id as the *commentOwner*. Therefore, there is a 1 to N relationship between *schema* and *comment.* Similarly, each user can provide multiple comments on many different schemas; therefore, the relationship between *user* and *comment* is also 1 to N. It can be illustrated by the following figure:



Relational representation between *user*, *schema* and *comment*

---

4. SchemaDB User Interface

4.1 Adding/Delete Info.

4.1.1 Add schema:

You can add a schema's name in two ways:

Method 1. In the page Add/Delete/Edit Schema as shown in the figure 4.1, you can enter a schema name and its description into the textboxes. It is illegal to enter a description without entering a schema name.

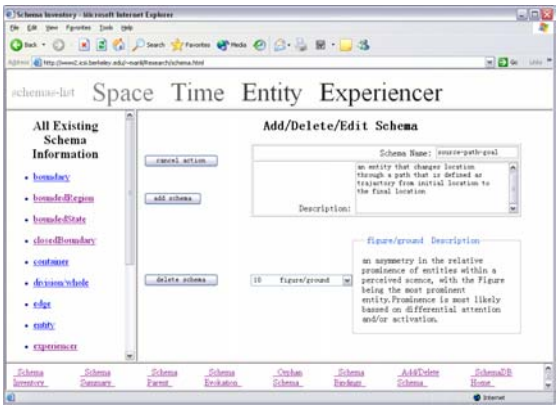


Figure 4.1 Add/Delete Schema

Method 2. In the page of ECG representation as show in figure 4.2, if the "parent" or "evoke" field of the current schema is not an existing schema, then it will be automatically added to the schema list as an "undefined" schema with a mark '*' followed by the schema name.

4.1.2. Add ECG information:

ECG information can be added at the ECG Form page by entering the information and then clicking the Add Info button. A previously undefined schema can be defined in this way also. All existing and added information, with the exception of Constraint and Comment, will appear in boxes at the top of the page for the user's convenience. The following figure represents the ECG Form page, which is the core of the SchemaDB system.

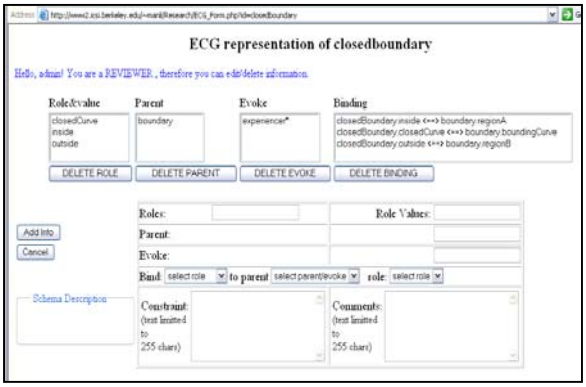


Figure 4.2 ECG Form

4.1.2.1

Add Role/Role Value: Role and Role value can be added one (pair) at a time. A Role can be added without its corresponding Role Value. A Role Value can be added later to an existing Role by typing in the existing Role name in the Role input box and then typing the value that is to be added in the "Role Value" input box. However, without a role name, no role value can be added.

4.1.2.2

Add Parent/Evoke: Parent and Evoke schema names can be added one at a time. If the Parent/Evoke is an undefined schema, then it will be added into schema database with a * mark by it. These undefined schemas can be defined at anytime (refer to 1.1 and 1.2).

4.1.2.3

Add Binding: Bindings are designed in the way that the current schema's roles can only bind with the immediate parents and evoke

schemas' roles. The first pull down menu consists of all roles of the current schema, of which the user can select one at a time. The second pull down menu consists of a list of all the schemas which are parents of or are evoked by the current schema. Once a parent/evoke in the second column is selected, the roles of that parent/evoke schema will appear in the third pull down menu. The user then selects one of the roles from this third pull down menu to bind with the selected role from the current schema.

4.1.2.4

Add Constraint: A schema can only have one constraint. The newly added constraint information will overwrite the old one.

4.1.2.5

Add Comment: Each schema can have multiple comments. Comments can be added through ECG Form page. Each comment is restricted to a maximum length of 255 characters. The user name of the person making the comment will also be added into the database so that later users can see *who* has entered *what* comment on a particular schema. Once a comment is added it remains permanent, and will not be removed until the whole database system is deleted.

4.1.3 Delete ECG information:

Delete ECG information is also done through the ECG Form page shown in figure 4.2.

4.1.3.1

Delete Role: Roles can be deleted one at a time by first clicking on the role you wish to delete and then clicking the DELETE ROLE button. The consequence is that any binding involving this role will also be deleted. If the current schema is the parent of or evoked by other schemas, this will result in the deletion of bindings to this role that have been declared within other schemas. Role values can be deleted by editing the role information. Type in the role name in the "Role" box, leave the "Role Values" box empty, and then click the "Add Info" button.

4.1.3.2

Delete Parent: Parents can be deleted one at a time by first clicking on the parent schema you wish to delete and then clicking the DELETE PARENT button. The consequence is that any binding involving this parent will be deleted.

4.1.3.3

Delete Evoke: Evoked schemas can be deleted one at a time by first clicking on the Evoke schema you wish to delete and then clicking the DELETE EVOKE button. The consequence is that any binding involving this evoked schema will be deleted.

4.1.3.4

Delete Binding: Bindings can be deleted one at a time by first clicking on the binding you wish to delete and then clicking the DELETE BINDING button.

4.1.4Delete schema:

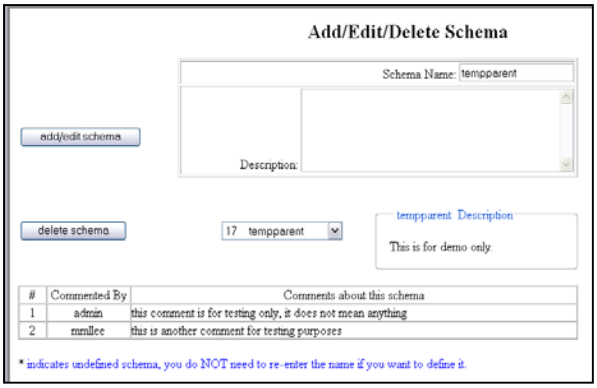


Figure 4.3 Add/Delete Schema Page

There is only one place you can delete a schema, which is at Add/Delete schema page (figure 4.3). It is easy to delete a schema by clicking the "delete schema" button, however you have to consider the consequences, such as 1) If the current schema is a parent of another schema or is evoked by another schema, you have to accept the fact that by deleting current schema, another schema might become an "orphan"; 2) All bindings involving the current schema will be deleted.

4.1.5 Edit Schema Description:

A schema's description can be added or edited through Add/Delete schema page (figure 4.3). If a schema's description exists, the new input will overwrite the old information.

4.1.6 Edit ECG information:

The ECG information of a schema can be edited using steps of 1.4 and 1.2.

4.2 Browse Info. (Accessible for any registered user)

4.2.1 Categorized schema lists

There are five categories of schemas. They are Space, Time, Entity, Experiencer and Relationship. By clicking on these words, the left column of the page will show all schemas in this category.

4.2.2 Schema Inventory information:

Schema Inventory is a list that has the information of a schema's name and its corresponding constraint and description. This is displayed in a new browser window, allowing the user to print it easily.

4.2.3 Schema ECG Summary

Schema ECG Summary is a complete list that consists of schema's name and its corresponding roles and values, parents, evokes and bindings.

4.2.3 Parent/Evoke Schema summary

This is a list that indicates the schemas and their corresponding parents and evokes schemas. The user can order the list according to the name of the schemas or the name of the parents/evokes. This is to make sure the user is able to see all parents of one schema or all children of one parent.

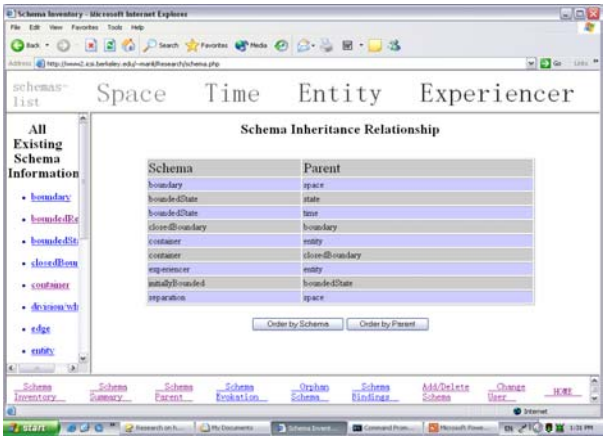


Figure 4.4 Browse Info.

4.2.4 Orphan Schema summary

This is a list that indicates the schemas and their corresponding descriptions. These schemas either have no parent or no evoke. By clicking the appropriate button, the user can obtain either the list of schemas that have no parent or the list of schemas that have no evoke.

4.3 System Security Management

4.3.1 User Registration:

A user can access SchemaDB by registering as a user through the User Registration Page (figure 4.5). All user names are unique, the system will complain if you entered a duplicated user name, or leave blank on the required field, or type in two different passwords. Once you are registered, you can browse information of the SchemaDB. However, you have no privileges of adding, editing and deleting information from the SchemaDB. You need to email the User Administrator to request such privileges. Upon reviewing your registration information, the User Administrator will set your add, edit and delete rights. You also need to send a request to the User Administrator in order to change your email and phone number, etc.

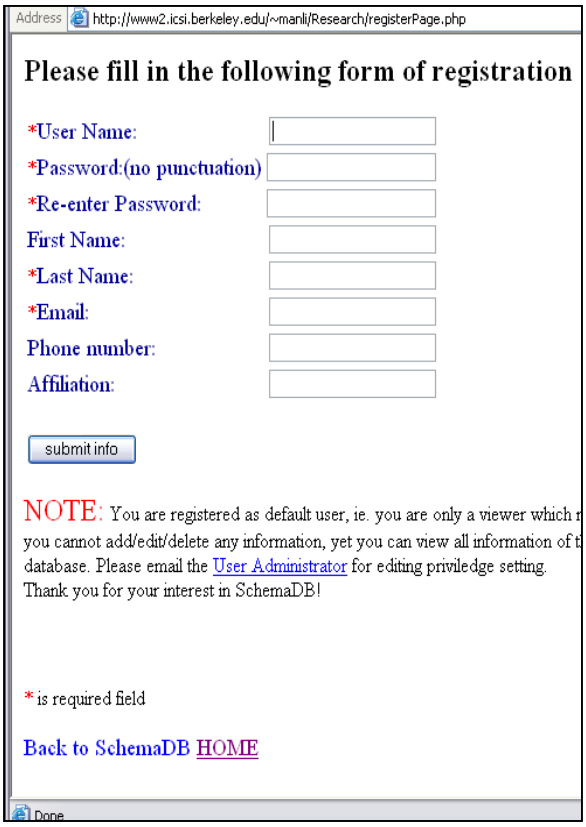


Figure 4.5 User Registration

4.3.2 User Login/Change Password:

A user needs to login in order to enter the SchemaDB system. Upon login, you can change your password or directly enter SchemaDB. After changing your password, you have to re-login immediately using the new password. You can go back the SchemaDB home main page at anytime in the process.

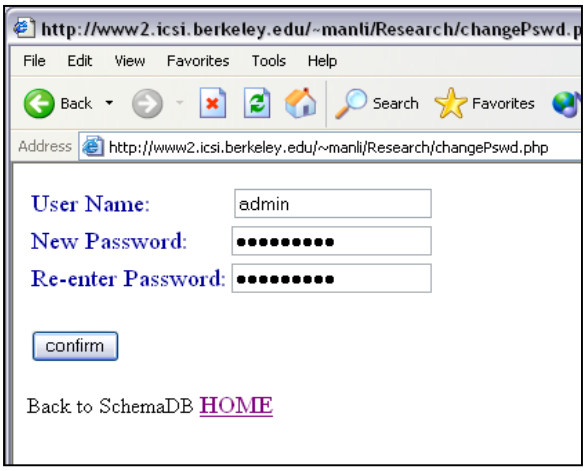


Figure 4.6 User Login

4.3.3 User Management:

This feature is designed for the User Administrator only. The User Administrator can change email address, phone number and Affiliation information for all users and most importantly can set the add/edit/delete privileges for the users by checking the radio buttons on the page form (figure 4.7). The system is designed to protect the critical information of all users; therefore even the User Administrator cannot change the user names and cannot see the passwords. If a user forgets his/her password, the user has to click on the "forget password" link on the login page, which allows the system programmers who have access to the user database through MySQL command line to reset a new password for the user. In addition, the User Administrator can deactivate a user. For the time being, "deactivate" means delete, in the future, "deactivate" could mean that the user name will be maintained in the database but the user will be unable to log in to SchemaDB. The purpose of this potential feature is to maintain a repository of all users of the SchemaDB.

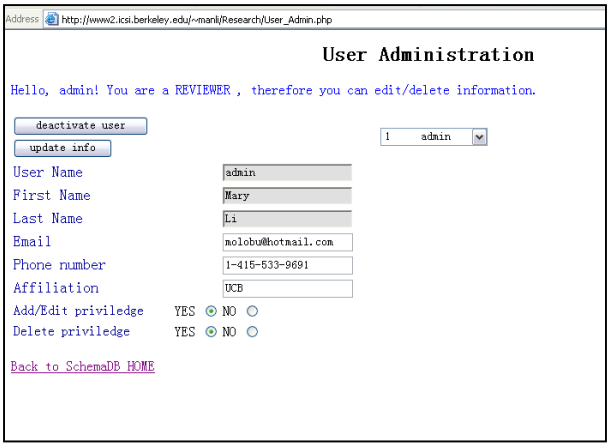


Figure 4. 7 User Administration

4.3.4 System Security:

Since SchemaDB is an application that involves operation on databases, security is a major concern. The system is designed such that all pages are protected. Without user login, no add/edit/delete operations can be done on databases. The system uses Cookie techniques to ensure security by storing necessary user information on the browser. This stored information enables the browser to differentiate from user to user; hence action buttons are displayed to the users accordingly. One important note is that users should NOT disable Cookie settings on browsers in order to have the SchemaDB application function properly.

---

5. System Implementation

In this section we will talk in great detail about some important parts of the program implementation. Altering information in the schema database system mainly takes place in two files, namely Add_Schema.php and ECG_Form.php. On the other hand, the whole system security check is implemented in the file named Authenticate.inc, which is an *include* file that is required to be executed before other files. Finally, a file structure diagram will be shown to present the flow of execution during the process.

5.1 File Add_Schema.php

This is the page allows a user to add/delete a schema and add/edit a schema's description. First of all, this page is protected using Cookies; the "Authenticate.inc" file (refer to section 5.3) is included to check the cookie information. Therefore, a browser only displays the page without any action buttons if the page is not authenticated, i.e. without proper user login information or if the user had logged in as "viewer" only. Technically, in order to display the description and its comments of selected schema on the same page, some "hidden" POST_VARS and JavaScript techniques have to be used to pass information between page calls.

In the case of inserting a piece of new information, there are three possible outcomes. The system is designed to first check if the input schema name is an existing schema; if so, it then to check if the existing schema is a previously undefined schema or defined schema. The three results are: a) if defined, only the description will be updated; b) if the schema is an undefined existing schema, both the schema description and name will be updated, the schema name is updated by taking away the symbol of undefined schema, '*'; c) if the schema name does not exist in the database, the new schema name is inserted into the "schema" table with the entered description (could be blank if nothing entered).

The schema's name will be updated only of the original schema is undefined. An undefined schema name is usually entered in the ECG_Form page (refer to section 5.2), and is stored in the "schema" table with a special mark '*' on the right side of the name as an indication both to the user and to the system itself. To check such undefined schema, the system uses "substr(string, int, int)" function provided by PHP. PHP Code segment is shown as follows:

```
$sql = "SELECT sName FROM schema WHERE
sName=\"$inputName\";";
$rslt = mysql_query($sql,$db);

if ($rslt)
{
        if ($row=mysql_fetch_row($rslt))
        {
        $testsuffix = substr($row[0], -1);
        if ( $testsuffix == "*")
        {
        # it's previous undefined schema, now
update the name by taking away the '*' mark
```

```
$sName = substr($row[0], 0, -1);
}
```

As a consequence of updating a schema's name, the "parent" table and "evoke" table also have to be updated with the new name. The motivation of adding '*' mark is to differentiate an undefined schema from all other schemas, this design is based upon users' visual intuitiveness. However, this set of operations will create overhead and result in system performance penalty. With the small set of data (about 40 schemas) in the database at current testing phase, the difference is not yet observed.

When deleting a schema, it has to be taken into consideration that information will be deleted accordingly in connected tables such as "role". This is because we have to make sure that the "role" table does not contain any roles of the deleted schema. The same reasoning applies to the tables "parent", "evoke", and "binding"--it does not make sense that a schema is deleted but it can be evoked by other schemas or it could be counted as a parent of other schemas; bindings involving the deleted schema should also no longer exist. Therefore all information related to the subject schema should be deleted from these tables, i.e. the whole database will no longer contain information about the subject schema.

5.2 File ECG_Form.php

This is the essential part of the system. ECG stands for Embodied Construction Grammar; it is a formal representation of schema. All actions upon ECG representation take place here. The following sections describe the implementation of the page form.

5.2.1 Form Variables

In order to describe the program in detail, we need to introduce some important form variables, which are used extensively in the process of database information altering:

```
$sName=the current schema's name, also serves as id in
URL, this is not a POST_VARS but a GET_VARS variable
$pid = the parent id of current schema, it is the "id" field in
the "schema" table (local variable)
$rName = role name that is from the input box
$rVal = role value (or type) that is from the form input box
$pName = parent name that is from the input box
$eName = evoke name that is from the input box
$currentRole = role name that is from the selection box
$ancestor = parent/evoke name that is from the selection box
$ancesstorRole = parent/evoke role name that is from the
selection box
```

5.2.1 Displaying the Page

5.2.1.1 *URL Variable:* In order to display information on a chosen schema, a schema "id" (schema name is used here) is passed to the ECG_Form.php through URL by including the file "Authenticate.inc". By extracting from the GET_VARS array, the schema id is identified, thereafter using MySQL statements to extract the corresponding role information from the "role" table, parent and evoke information from the "parent" and "evoke" tables and binding information from the "binding" table. All information is displayed in a selection box, where the user can select one at a time to delete.

5.2.1.2 *Integrate JavaScript:* Operation buttons such as "Add Info" and "Delete xxx" buttons are set to be displayed only if the logged in user has the rights to add/edit/delete information from SchemaDB. In order to add information to the database, the form uses POST_VARS technique particularly with the hidden variable "DoIt" and JavaScript change() that allow the page to display properly. Initially, the value of the hidden form variable "DoIt" is set to be NULL, upon user click on the selection box, the JavaScript "change()" instantly change the variable value to "notnull" thus evokes the action according to the variable value.

Here we illustrate the technique using the example of the process of schema binding. In order to insert binding information to the database, the user will have to select three

pieces of information, namely, a) the role of current schema represented by the variable $currentRole; b) the parent/evoke schema which the schema needs to bind to, which represented by the variable $ancestor; c) the role of the chosen parent/evoke schema, to which the current schema role needs to bind, represented by variable $ancesstorRole. Since information in c) depends on information on b), we need to submit the information in b) to the system in order to extract information from the database, therefore the function "change()" serves as a "submit button" in the form. The simple JavaScript is shown here:

```
<SCRIPT LANGUAGE = "javascript">
function change()
{
        window.document.ECG.DoIt.value = "notnull";
        window.document.ECG.submit();
}
</SCRIPT>
```

Upon receiving the submit information, the system takes proper action according to the form variable values, and information is added into the binding table.

5.2.2 Adding ECG information

In the ECG form, the *Role*, *Value(Type), Parent* and *Evoke* fields are text input boxes. Each of these four variables is set to be a 20-30 character long string. The system first checks if an input string exists in the database, then decides to either ignore the input if it exists or insert input to the database otherwise. Since *Role* and *Value(Type)* are paired in the ECG, users are allowed to input a *Role* name without *Value(Type).* However, to edit an existing *Role's Value(Type),* the input *Role* name has to match the existing one and then the *Value*(*Type*) of the role will be UPDATED in the table "role" using the input Role Value(Type). A similar technique is used to enter information for the field of *Parent* and *Evoke*. However, if an input *Parent* or *Evoke* schema did not exist before, it will be added to the table "schema" as an "undefined" schema (refer to section 5.1) with an '*' indicator for search purposes in later use. Binding is the most complex field in the form. All three selection boxes are not only for displaying information, but also serve as input of the form once information is selected. The first selection box displays roles extracted from the "role" table for the current schema; the second selection box displays the parents and evokes of the current schema from the table "parent". Once the parent/evoke schema is selected, a "pid" is set by the system (on line 78), its roles can then be displayed in the third selection box accordingly. To add a piece of binding information, refer to the example given in section 5.2.1.2.

5.2.3 Deleting ECG information

Any deletion decision made on this page is permanent, there is no "undo" process to reverse the action. However, a prompt will be given to the user to remind them of the serious consequences of deleting such information. There is no backup information of data deleted; the information appearing on the page are extracted from the database directly and deletion is also executed on the system database directly. Therefore, any deletion decision should be made with care.

5.2.4 Security

By using cookie techniques and appended URL id, the page will be displayed as a plain form without content and operation buttons if no user login has occurred and no schema has been selected. For a more detailed explanation, refer to the following section 5.3 (Authenticate.inc).

5.3 File Authenticate.inc

In order to retain system security, we need to check user information. All PHP files that perform security checks and database operations include Authenticate.inc. These files are *Add_Schema.php, bottom.php, changePwd.php, ECG_Form.php and*

User_Admin.php. In the file *Authenticatae.inc,* it first connects to the database ("schemadb"), then checks the cookie settings on the browser, sees if the user has logged in, and in what role a user is. It checks the "*user*" table and retrieves the Add/Edit/Delete rights for user. It also checks the URL appended id, using GET_VARS to extract the information. Thus the system will later be able to display the appropriate HTML page for the particular user. In addition, Authenticate.inc checks the PHP version in order to use the arrayKeyExists () function properly.

5.4 System File structure and execution flow

The following chart shows systematically how each page is related to the other. The rectangular boxes are files that are executed on the runtime but not shown as pages on browsers. The rest are pages either individually shown on the browser or in a frame of a page.

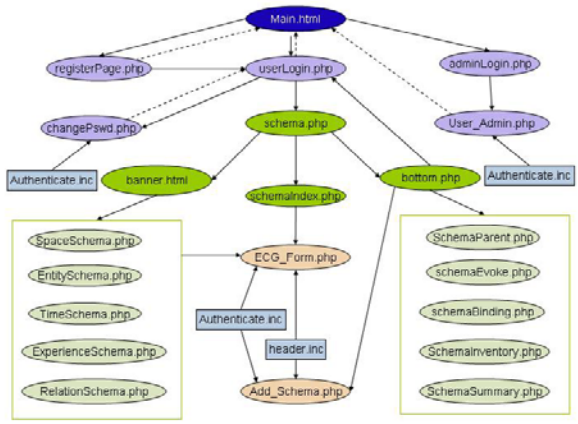


Figure 5.1 Execution Flow Chart

The direction of the arrows indicates the sequence of execution. The page attached to the tail of an arrow executes before the page attached to the head of the arrow.

---

6. System Software Requirement

6.1 Client

A client of SchemaDB system can be run on browsers that support JavaScript such as Internet Explorer 5.0 and Netscape 6.0 or above. It has not yet been tested on any other browsers. Cookies should be enabled in browsers.

6.2 Server

6.2.1 ICSI Experimental Server
Refer to the following link:
http://www.icsi.berkeley.edu/sysdocs/

6.2.2 Using Personal computer as a Server
The server of the system consists of HTTP server, database server, and PHP interpreter.

a) An Apache HTTP Server 2.0.46 needs to be installed. It can be downloaded free from the following link: http://httpd.apache.org/ . However, the server needs to be configured in order to accommodate the PHP interpreter.
The following two lines of code should be added at the end of the httpd.conf configuration file of Apache Server to cooperate with PHP interpreter:

LoadModule php4_module "c:/php/sapi/php4apache2.dll"
AddType application/x-httpd-php .php

The document root should also be configured so that all files stored under the root directory can be executed. For example, if your document root is set to be C:/Username, then the files under this directory can be browsed on a local browser by type in the following address: http://localhost/filename ; the files can be browsed by a remote browser by type in http://ip-address-of-your-server/filename .

b) MySQL 3.23 can be downloaded free from
http://www.mysql.com/downloads/mysql-3.23.html

c) PHP 4.3.2 can be downloaded free from
http://www.php.net/downloads.php

---

7. Acknowledgements

Jerome Feldman:
Professor of EECS Dept., UC Berkeley
Leader of AI research group at ICSI (International Computer Science Institute)
Srini Narayanan:
Senior Research Scientist at ICSI
George Lakoff:
Professor of Linguistics Dept., UC Berkeley
Author of numerous texts including *Philosophy in the Flesh*, *Women, Fire, and Dangerous Things*
Ellen Dodge:
Graduate student, Linguistics Dept. UC Berkeley
Michael Meisel:
Graduate student, CS Dept. UCLA

---

8. References

[1] Michael Meisel. "Metaphor Representation using a Relational Database System" 2003, Project Report

[2] Benjamin K. Bergen and Nancy C. Chang. "Embodied Construction Grammar in Simulation-Based Language Understanding". 2002, ICSI TR-02-004

[3] Jerome Feldman and Srini Narayanan. "Embodied Meaning in a Neural Theory of Language" 2003, Brain and Language (accepted for publication).

[4] George Lakoff and Mark Johnson. *Metaphors We Live By*. 1980. Chicago: University of Chicago Press[i]

---

[i] B.S. Computer Science, San Francisco State University. Currently is a PHD student at UCSB, Dept. of ECE.