Design for new VERCON/RADCON Components

Part I

Prepared By

Samhaa El-Beltagy

October, 2005

Table of contents

1 Introduction

In a previously prepared report "*VERCON Changes and Enhancements: A Follow up report based on the Information Needs Assessment of VERCON stakeholders*", some components identified by the information needs assessment of VERCON stakeholders were found to require further design. In this report, the design of three of these components is presented. Specifically, this report details the design the following components:

1. A Component for Linking the Farmer's Problem to Extension Brochures
2. An Alert System For Frequently Consulted Expert System Problems
3. An Agricultural Services Guide

While the first two components, extend already existing ones, the third is a brand new system that might be better suited in the new RADCON information system. In each of the following sections, the design of one of the above listed components is detailed. Before the design is elaborated upon, a task description is provided along with references to the original documents from which this component is derived. The report on the **Information Needs Assessment of VERCON stakeholders** will hence forth be referred to as RAD/05/2 while the report titled **VERCON Changes and Enhancements: A Follow up report based on the Information Needs Assessment of VERCON stakeholders** will be referred to as RAD/05/3.

2 Linking the Farmer's Problem Component to the Extension Brochures System

2.1 Task Description

What is required is the development of a method "to link the farmer problems tracking system with the extension bulletin system to support the solution by more information if needed (For more information click here)". There is no straight forward way of doing this automatically expect, by providing descriptors for the solution and seeing whether a corresponding item can be found in the database.

So, what is needed is the implementation of a component that attaches meta-tags to a problem using solution descriptors provided by the researcher that solves the problem. These descriptors should be the ones employed by the brochures system in order to be able to link to that system. The attached descriptors can then serve as a query to the brochures system and result in the return of associated links.

Cross reference(s)
RAD/05/2,Section 3.1.1

RAD/05/3,Section 3.1, recommendation 3

2.2 Database changes

Introduce a new table < Problem_Descriptors> in the database that has the following fields:

a. The id of the encountered problem (P_ID). This is a foreign key which refers to P_ID in table <Problems>
b. The id of the plant (PLANT_ID). Both the problem tracking system and the brochures system use the same table <PLANTS> to get a reference to plants.
c. The id of the operation under which the problem is classified OP_ID (if applicable). This is a foreign key which refers to CODE in table <Operations>
d. The id of the disease around which the problem revolves Disease_ID. This is a foreign key which refers to CODE in table <Diseases>
e. The id of the insect around which the problem revolves insect_ID. This is a foreign key which refers to CODE in table < Insects>

A check has to be in the global vercon database to ensure that both the problem tracking system and the brochures system are using the same tables to refer to the same things(mainly check that they're both using the same plants table).

2.3 Interface changes

The interface for posting a solution will have to be changed (in case of production type problems). It will have to allow solution producers to add descriptors to the entered solutions. The interface presented to the end user will also have to change to allow for related links to be added after the solution.

In the case of the interface for solution producers, the interface should incorporate a dropdown list of all possible operations, another for diseases, and finally one for insects. The solution producer should select all of these that apply to the problem/solution. The crop need not be place here for selection, as the user might have possibly included it in his/her description of the problem. If selected however, it should be made available as a session variable or a parameter so that its value can be set in the database. After the user enters the solution, problem related descriptors should be stored in table Problem_Descriptors. If the solution needs to be approved by someone else before being published, then the selected descriptors should be shown to the approver and s/he must be able to change them. In case of such a modification, the saved entry must be updated in the database.

Regarding the end user's interface, when viewing the solution for a problem, the descriptors for that problem in table Problem_Descriptors should be matched against descriptors in table Doc_Index. If any records match, then the title(s) of the matched sections should be displayed to the user. These will be found in the

CAPTION field of the matched records. A very simple way for creating a link for these records, is through the use of their ID and a page that has already been developed to view brochure sections. To create such a link, the developer of this system simply needs to concatenate the ID of the retrieved record <R_ID> to the following string:
http://www.vercon.sci.eg/VerconProject/HTMLView2.asp?ID=
which will result in the following link
http://www.vercon.sci.eg/VerconProject/HTMLView2.asp?ID=*R_ID*

3 The Expert System Frequently Consulted Problems Alert System

3.1 Task Description

What is required is the “development of a mechanism to follow-up repeatedly consulted expert system problems; that can be used to warn other users about possible events”.

Since what is required is related to problems, then it can be thought of as only related to the diagnostic part of the expert system. Having said that, the design being presented is independent of the expert system being targeted. To detect frequently consulted problems, The frequency of the repetition of the problem within some pre-defined time range can serve as an indicator. So, logging the occurrence of problems and their dates is essential to the successful operation of this system.

In the proposed design, there are two ways in which event alerts can be created and conveyed to the user. In the first, an experts defines the conditions for raising an alert and interested users subscribe to being notified of such alerts or events. These type of alerts will be hence forth referred to as system based alerts. Every time some given problem is encountered, any conditions related to that problem will be checked, and if met, a message created by the expert who specified this event will be sent to all users who subscribed to system based events. The second way in which an alert can be created, is when a user is interested in being alerted when some given problem occurs with a specific frequency in a given region (or in all regions). In such a case, the user defines the conditions under which s/he wants to be alerted, and as soon as these conditions are encountered, s/he and only s/he gets notified. In the following subsections, a detailed design of the changes and components needed to achieve these tasks, is provided.

Cross reference(s)
RAD/05/2,Section 3.1.4
RAD/05/3,Section 3.2, recommendation 1

3.2 Database changes

1. Introduce a new table < Problem_Log> in the database that has the following fields:
    f. The id of the encountered problem (PID)
    g. The date this problem was diagnosed (D_Diagnosed)
    h. The id of the region in which this problem was reported. (RID)
2. Introduce a second table <System_Alert_Events> to define an event that requires an alert. This table should have the following fields

i. The ID of the problem for which an alert is to be automatically raised. (PID)
j. The Id of the administrator who created this entry. (admin_ID)
k. The number of times this problem has to be raised to signal an alert. (Freq)
l. The time interval during which the frequency (specified in the previous point) occurs in order to signify a problem. This should be denoted in months. (T_Int)
m. Text to send to the user in case this alert is raised. (alert_text)
n. A date to specify when this event was last raised. (last_raised)
o. Time that needs to elapse before raising this alarm again (in months). (T_elapsed)
p. The date on which this entry was created. (date_created)

3. Introduce a third table <User_Defined_Alerts> to define an event in which some given user is interested. This table should have the following fields:
   q. The ID of the problem for which an alert is to be raised. (PID)
   r. The Id of the user who created this entry. (user_ID)
   s. The number of times this problem has been raised to signal an alert. (Freq)
   t. The time interval during which the frequency (specified in the previous point) occurs in order to signify a problem. This should be denoted in months. (T_Int)
   u. The ID of the region in which this problem has to occur in order to raise an alert. 0 in this field should mean any region. (RID)
   v. The date on which this entry was created. (date_created)
4. Introduce a fourth and final table Alert_ Subscriptions to hold the ids of subscribers of system raised alerts. This table should have the following fields:
   w. The Id a user who is interested in being notified of system raised alert. (user_ID).

3.3 Expert System changes

Every time a problem is encountered, a check has to be made to see whether or not an alert is going to be raised because of the occurrence of this problem. The algorithm for doing this is described as follows:

Every time a problem X occurs

- Add the problem's Id, the current date and the region in which this problem occurred (obtainable from the working memory) to the <Problem_Log> table.
- Check to see if the Id of problem X is in table <System_Alert_Event>. If it is not, then run Algorithm 2, else continue
- If this alert was raised before (this can be found by checking the last_raised field, then see whether the time required to raise this alarm again has passed or not. This can be done by:
  1. Obtaining the current date

2. Adding the number of months in T_elapsed to the date last_raised
3. Comparing the resulting date to the current date. If it is smaller then run Algorithm 2, else continue

- Check the problem log to obtain a listing of all the entries of this problem. Filter this result by:
  1. Subtracting the number of months specified in field T_Int from the previously obtained current date to obtain a lower bound date.
  2. Eliminating all obtained entries that have occurred before the lower bound entry.
- Check the number of remaining entries. If these are equal to or greater than the number specified in field frequency, then raise the event alert. This can be accomplished by the following steps:
  1. Setting the last_raised field to the current date
  2. Obtaining emails for all users who've subscribed to this event from the FORUM_MEMBERS table (M_EMAIL field).
  3. Sending the message in the alert_text field to all subscribed users via email.
- Run Algorithm 2

Algorithm 2

- Check to see if the Id of problem X is in table <User_Defined_Alerts>. If it is not, then exit, else continue
- Obtain the current date
- For each entry of problem X in table User_Defined_Alerts do the following:
- Check the problem log to obtain a listing of all the entries of this problem that match the region specified in the entry's RID. Filter these by:
  1. Subtracting the number of months specified in field User_Defined_Alerts .T_Int from the current date to obtain a lower bound date.
  2. Eliminating all obtained entries that have occurred before the lower bound entry.
- Check the number of remaining entries. If these are equal to or greater than the number specified in field User_Defined_Alerts.frequency, then raise the event alert. This can be accomplished by the following steps:
  1. Obtaining email of the user who created this event from the FORUM_MEMBERS table (M_EMAIL field).
  2. Deleting the entry from the table

3.4 User Interface Changes

Several types of user interface changes need to be carried out. Firstly, new interfaces for defining system raised and user defined alerts as well as deleting

old system defined entries should be developed. Secondly, access for these interfaces has to be given through modifications in the currently available ones.

3.4.1 The interface for defining system raised alerts

This interface should only be accessible to authorized users. Such users will be provided with a url for creating alerts. After entering this url in their browsers, a login screen should appear. Successful entry of user and password information should provide the user with the means to carry out the following tasks:

- Create a new alert
- Delete old alert entries
- Logout

Logging out is a straight forward task. The other tasks are described in details in the following subsections.

3.4.1.1 Create a new alert

Selecting this option should result in a window containing a form for defining the alert, to appear. The form should contain the following fields:

1. A problem field. This should be a drop down list from which the user should be able to choose the problem for which to create an alert. This field should map to System_Alert_Events.PID
2. A frequency field. This is an input box where the user should specify the number of times this problem has to be raised to signal an alert. This field should map to System_Alert_Events.Freq
3. A Time interval field. This is an input box where the user should specify a time interval in months during which the indicated problem occurs with the specified frequency in order to signify a problem. This field should map to System_Alert_Events.T_Int.
4. An Alert text field. The is a text field where the user should enter the message to send to the user in case this alert is raised. This field should map to System_Alert_Events.T_ alert_text
5. A Time after which to raise this alarm again. This is an input box where the user should specify the number of months that need to elapse from when this alarm was last raised, before it can be raised again. This field should map to System_Alert_Events.T_elapsed

Upon pressing a submit button, these fields need to be validated. Successful validation, should result in the formulation of an SQL string to add an entry to the database. Besides the information entered by the user, the string should also contain the id of the user who created this entry (this can be made accessible through the use of a session variable or a cookie) which should map to System_Alert_Events.admin_ID, and the current date, which should map to System_Alert_Events. date_created. The database field System_Alert_Events.last_raised should be left empty.

3.4.1.2 Delete old alert entries

This option should be used to clean the database from entries that are no longer valid. Usually, these will be entries that at the time of deletion, are out of date. To clean the database of these entries, the user should be given the choice to delete all database entries before a date that s/he specifies. So, what is required is provide him/her with an interface in which s/he can enter a date before which all entries should be deleted. Before the deletion operation actually takes place, the user should be told of the number of records that are to be deleted and asked whether or not they are sure you want to carry out the operation.

3.4.2 The interface for creating user defined alerts

A link to this interface should be provided to any user who is using the Expert System component below the system's main links. Following a link to this interface should result in the display of a form for defining an alert. This form should have the following fields:

1. A problem field. This should be a drop down list from which the user can choose the problem for which to create an alert. This field should map User_Defined_Alerts.PID
2. A frequency field. This is an input box where the user should specify the number of times this problem has to be raised to signal an alert. This field should map to User_Defined_Alerts.Freq
3. A Time interval field. This is an input box where the user should specify a time interval in months during which the indicated problem occurs with the specified frequency in order to signify a problem. This field should map to User_Defined_Alerts.T_Int.
4. A region field. This should be a drop down list from which the user can choose region for which to create an alert. This field should map to User_Defined_Alerts.RID
5. A Time after which to raise this alarm again. This is an input box where the user should specify the number of months that need to elapse from when this alarm was last raised, before it can be raised again. This field should map to User_Defined_Alerts.T_elapsed

Upon pressing a submit button, these fields need to be validated. Successful validation, should result in the formulation of an SQL string to add an entry to the database. Besides the information entered by the user, the string should also contain the id of the user who created this entry which should map to User_Defined_Alerts.user_ID, and the current date, which should map to User_Defined_Alerts. date_created.

3.4.3 The interface for subscribing/unsubscribing to system defined alerts

The interface for subscribing or unsubscribing to system defined alerts need not be an elaborate one. Subscribing or unsubscribing to these kinds of events can be simply achieved by providing a link from the expert system's main menu. Following a "Subscribe to alerts" link, should result in a confirmation message to

be displayed asking the user whether or not s/he is sure s/he wants to subscribe to this service. If confirmation is obtained, then an entry in table **Alert_Subscriptions** should be created with the currently logged in user ID. A check that this entry is not already in the database should be performed first to prevent duplicate entries. Similarly, following the “Unsubscribe to alerts" link, should result in a confirmation message to be displayed asking the user whether or not s/he is sure s/he wants to unsubscribe to this service. If confirmation is obtained, then the currently logged user’s ID should be deleted from the **Alert_Subscriptions** table (if it was there in the first place). In both cases a message should be displayed to the user telling him/her, that his/her desired action has been successfully completed (or not).

4 Agricultural services guide

4.1 Task Description

What is required is the development of a system that would aid farmers to find the selling places and prices of commodities, and the contact points of various services.

Cross reference(s)

RAD/05/2,Section 4.3

RAD/05/3,Section 3.3, recommendation 4

For this guide to be useful, it has to provide an easy to use front end. It also has to take into consideration the fact that the search criteria may exceed that mentioned in document RAD/05/2,Section 4.3. The criteria listed in the document is as follows:

1. Soil analysis labs.
2. Sales points of agricultural production.
3. Sales points of input supplies for poultry breeding.
4. Sales points of bees and bee supplies.
5. Sales points of veterinary supplies.
6. Exporters of agricultural production (companies and individuals).
7. Importers of agricultural input supplies and machinery.
8. Sales points of fertilizers.
9. Sales points of trusted pesticides.
10. Sales points of seeds and seedlings.
11. Sales points of agricultural machinery.
12. Sales points of silkworm supplies and marketing places.
13. Nurseries.
14. Ammonium injection units.
15. Poultry and rabbits production stations.
16. Dairy collection centers.
17. Sales points/breeding stations of cattle.
18. Agricultural and rural training centers.
19. NGO working in rural and agricultural domain and its services.
20. Researchers, consultants and experts of agriculture and their specializations and contact address.
21. Guidebook for exportation production criteria.

Regarding item 20, a whole project (the NARIMS project) is dedicated to this point. As such, it should not be included here. Instead a link to the NARIMS system can be provided in the system's home page. As for item 21, it is dissimilar to other items (all other items are physical entities) and as such can not be entered in the database in a manner similar to others and should not be included in this system. By analyzing the rest of the items it can be seen that

these can be broken down to categories and subcategories. Identified categories include:

- Labs
  - Soil analysis
- Sales points
  - Agricultural production.
  - Input supplies for poultry breeding.
  - Bees and bee supplies.
  - Veterinary supplies.
  - Fertilizers.
  - Trusted pesticides.
  - Seeds and seedlings.
  - Agricultural machinery.
  - Silkworm supplies.
  - Cattle.
- Nurseries
- NGOs
  - rural and agricultural domain and its services
- Exporters
  - Agricultural production
- Importers
  - Agricultural input supplies
  - Agricultural machinery
- Injection units
  - Ammonium
- Production stations
  - Poultry and rabbits
  - Dairy collection centers
- Training centers
  - Agricultural and rural

To achieve the above mentioned goals, the system should have two types of users: backend authorized users who enter required information into the database and front end users (VERCON/RADCON users) who search this information. Backend users are further divided into two types of users: administrators, and information providers. Each user, will have a different type of interface, details of which are provided in subsections 4.3 and 4.4 . A database will be utilized in both cases. The design for this component is described in the following subsection.

4.2 Database Design

The database component will serve as a repository for services and their related information. Backend users will have the ability to store and modify information while front end users will have the ability to retrieve information within it .The database to be used within this system should have at least the 5 tables shown in Figure 1.

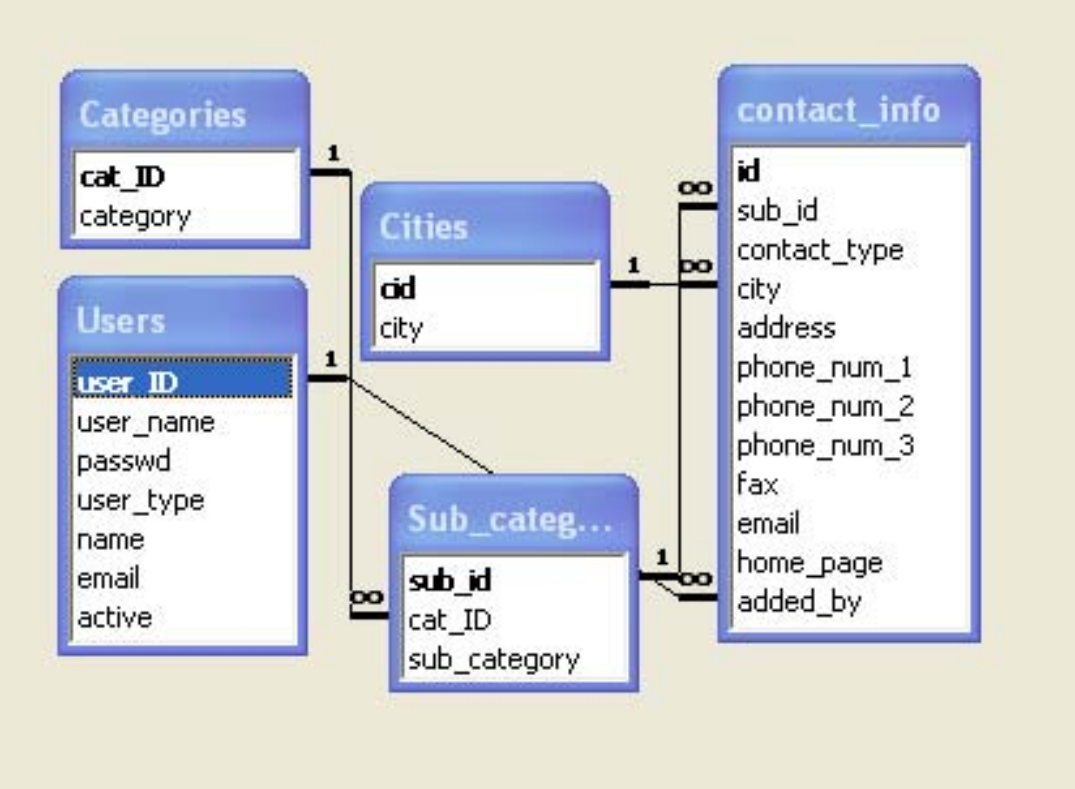

Figure 1: Database Tables

Table **Users** will define system users. Each user will be defined in terms of the following properties:

- *user_ID*: a number denoting the ID of the user the as well as being the primary key
- *user_name*: login name
- *passwd*
- *user_type:* this is a number that will specify the type of some given backend user. 1 will indicate an administrator user, 2 will indicate an information provider user. Future introduction of user types, should be reflected here.
- *name*: the user's real name
- *email*: the user's email.
- *Active*: this is a numerical field to be used to indicate whether a user is active or suspended. A suspended user is a user whom an administrator has deleted but which has previously entered valid service provider information. 1 will be used to indicate an active user will 0 will be used to indicate a suspended one. This field should be set to one by default.

Table **Categories** will be used to store various general categories under which system providers can be classified. It will have the following fields:

- cat_ID: a number denoting the ID of the category as well as being the primary key.
- category: a string denoting the category

Table **Sub_categories** will be used to store various specific categories under which system providers can be classified. It will have the following fields:

- sub_ID: a number denoting the ID of the sub_category as well as being the primary key.
- cat _ID: a foreign key representing the ID of the general category of which a given sub_category is a specification or an instance.
- sub_category: a string representing the subcategory

Table **Cities** will be used to store a list of Egyptian cities. It will have the following fields:

- cid: a number denoting the ID of the city as well as being the primary key.
- city: a string representing the name of a city

Table **contact_info** will be used to store information about various service providers. This is the system's main table. It will have the following fields:

- id: a number denoting the ID of a service provider as well as being the primary key
- sub_ID: a foreign key representing the ID of the specific category under which some given service provider falls.
- contact_type: a number used to reflect the type of the entered service provider whether (individual, company, organization, etc.)
- city: a foreign key representing the ID of the city in which a given provider is located.
- address: The address of the service provider (excluding the city)
- phone_num1: The main phone number of the provider
- phone_num2: An alternative phone number for the provider
- phone_num3: same as above
- fax: The fax number of the provider
- email: The email of a contact person
- home_page: the url of the provider's web page
- added_by: a foreign key representing the ID of the user who made this entry

4.3 The Backend User Interface

Administrators should have the ability to add, or delete users as well as modify existing user information. Administrators should also have the ability to carry out all tasks that can be carried out by information providers. Information provider users should have the ability to add new categories and sub-categories, delete existing ones, and add/update and delete service provider information. They should also be able to change their own passwords. In order to carry out any of these tasks, an end user (administrator or otherwise) should first login to the

system. To do so a log-in screen should be provided. Suspended users should not be able to log-in to the system. Upon successful login, a screen with the following functionality should appear to the user:

Welcome <user name>
The Agricultural Services Guide Management System

System's Main Menu

User related functions (except for the last option, these options should only appear to a system administrator)

- Add a new user
- Delete an existing user
- Update an existing user 's data
- Change my password (to appear to all users)

System related functions

- Add a new Category
- Delete an Existing Category
- Add a new Subcategory
- Delete an Existing Subcategory
- Add new service provider Information
- Update information about an existing service provider
- Delete an existing service provider

Logout

4.3.1 User Related Functions

4.3.1.1 Add a new user

- Display a popup window whereby the user can enter data for a new user. This window should contain a form for adding user information as well as an ***Add User*** button. User information will be defined in terms of the following properties:
    - Name <optional> : This is a string the stores the user's real name.
    - Email <optional>.
    - User name <mandatory>: this is a single word that represents the user's login name.
    - Password <mandatory>
    - Type: Initially, there will be two types of users: administrators and information providers.
- Carry out customary validation on input fields, including asking the user to re-enter the password in order to confirm that a typing mistake in password entry has not occurred.

- Verify that the entered user name does not already exist in the database.
    - If not, add user info to the database
    - If it does, then display an appropriate message to the user

4.3.1.2 Delete an existing user

Display a page that contains a list of all usernames with a checkbox next to each. If the number of users exceeds a developer defined number *n*, then usernames should be paged on multiple page each of which is of size *n,* to be navigated by the administrator through customary means (previous, next, page 1, 2, ..m). Each page should have a delete button. Pressing the delete button should result in a confirmatory message to appear to the administrator asking him/her whether s/he is sure s/he wants to carry out the delete operation. Should the administrator confirm the delete operation, a check should be carried out to see whether any of the deleted users has previously entered service provider information in the database. This check can be easily done by comparing the user's id against the added_by field in the contact_info table. For each user for whom an entry does exist in the table, set his/her active status to suspended. Otherwise, delete the user from the database.

4.3.1.3 Update an existing user's data

In case of an administrator user, offer the user the choice to enter the name of the user he/she wants to update. In this case search for this name in the DB. If it is found then, offer the user a popup window containing the following editable fields, the user's

- real name
- email
- password
- Active status (Active/ suspended)

Also offer a save changes button, the pressing of which should result in any changes to be saved to the database.

On the other hand, if the search fails, then display a message to this effect. Alternatively, offer the user a list of all users sorted alphabetically with a link to update each of the user's data. Following any of these links should result to the appearance of a window similar to the one describe above. If the number of users exceeds a developer defined number *n*, then users should be paged on multiple pages each of size *n,* to be navigated by the user through customary means.

4.3.1.4 Change my password

For a user who wants to change his/her own password, following a change my password link should prompt him/her to enter the new password twice. The two entered passwords should be compared against each other before the database is updated to ensure that no typing mistake has occurred.

4.3.2 System Related Functions

4.3.2.1 Add a new Category

Adding a new category should result in an update in the database table Categories. A user who selects to carry out this action should simply be prompted to enter the name of the new category. Before adding this category to the database, a check should be made to ensure that this category does not already exist.

4.3.2.2 Delete an existing category

When a user selects to delete an existing category, a list of all categories should be offered for the user to select from, as well as a delete button. Deletion of an existing category should never be performed unless it has no sub-categories. To check whether or not a sub-category exists for some given category, the field *cat_ID* in the Sub_ categories table should be checked against the id of the category to be deleted. If any entry that has the category to be deleted is found, then the delete operation should not proceed and a message informing the user of why the delete operation has failed should be displayed.

4.3.2.3 Add a new subcategory

To add a new subcategory a popup window should be displayed to the user. In this window a list of all existing categories should be shown to the user in the form of a drop down list for him/her to select from. An input box for entering the new subcategory should also be displayed along with a button to carry out the addition action. The required database check for this kind of entry is that it is not a duplicate entry.

4.3.2.4 Delete an existing subcategory

This operation is very similar to the deletion of a category. When a user selects to delete an existing sub-category, a list of all categories should be offered for the user to select one from. Then, a list of all sub-categories for the selected category should be shown. Deletion of an existing subcategory should never be performed unless no service provider has it as a descriptor. To check whether or not a this is the case, the field sub_id in the contact_Info table should be checked against the id of the subcategory to be deleted. If any entry has the subcategory to be deleted, then the delete operation should not proceed and a message informing the user of why the delete operation has failed, should be displayed.

4.3.2.5 Add new service provider Information

Upon selection of this option, a form should appear to the user. The form should include the following fields:

- Category: in this field all entered categories should appear in the form of a drop down list. This is a mandatory field.

- Subcategory: this filed should be initially empty. Upon selecting a category from the Category field, this field should then contain of the selected category's sub-categories. This is a mandatory field.
- Contact Type: in this field all contact types should be listed.
- Name: an input field where the name of the individual or company is to be entered. This is a mandatory field
- City: This field should display a list of all Egyptian cities in the form of a drop down list. These are to be found in the cities table. This is a mandatory field
- Address: an input box where the address of the service provider is to be entered. This is a mandatory field.
- Phone number: this input box should be left empty for the user to enter the phone number of the entered entity. This is a mandatory field.
- Second phone number: this input box should be left empty for the user to enter the 2nd number of the entered entity. This is an optional field.
- Third phone number: this input box should be left empty for the user to enter the 3rd number of the entered entity. This is an optional field.
- Fax: this input box should be left empty for the user to enter the fax number of the entered entity. This is an optional field.
- email: this input box should be left empty for the user to enter the email number of the entered entity. This is an optional field.
- Home page: this input box should be left empty for the user to enter the address of the entered entity's web site. This is an optional field.
- A Submit button.

Usual validation on the fields should be carried out and after validation entered information should be saved to the contact_Info Table. The system should automatically add the id of the user in the field contact_info.added_by based on login information.

4.3.2.6 Update information about an existing service provider

Updating service provider data should only be carried out by the user who initially entered the data or by an administrator. Such a user should be offered a list of all service providers sorted alphabetically with a link to update each of the provider's information. If the number of providers exceeds a developer defined number *n*, then the providers should be divided on multiple pages each of which is of size *n,* to be navigated by the user through customary means. Following a link associated with any provider should result in the appearance of a window containing the following editable fields initialized with their previously set values:

- Category
- Subcategory
- City
- Address
- Phone number
- Second phone number

- Third phone number
- Fax
- email
- Home page

A ‘Save Changes’ button should also be offered. Pressing this button should result in any changes to be saved to the database.

4.3.2.7 Delete an existing service provider

Deleting service provider data should only be carried out by the user who initially entered the data or by an administrator. Such a user should be offered a list of all service providers (entered by him/her) sorted alphabetically with a remove checkbox next to each. If the number of providers exceeds a developer defined number *n*, then the providers should be divided on multiple pages each of size *n.* Each page should have a delete button. Pressing the delete button should result in a confirmatory message to appear to the user asking him/her whether s/he is sure s/he wants to carry out the delete operation. Should the user confirm the delete operation, all checked service providers should be deleted from the contact_info table.

4.4 The search front end

The search front end is to be used by ordinary VERCON/RADCON users to search for some given service. The front end should be provided as a form with multiple selection criteria. These are to include:

- Name: an empty input search box, contents of which are to be entered by the user
- Category: A drop down list to be obtained from the database and from which the user can make a selection
- Subcategory: a drop down list to be obtained from the database depending on the selected category and from which the user can make a selection.
- City: A drop down list to be obtained from the database and from which the user can make a selection

A user should be able to specify his/her query using one or more of the above listed criteria. For example, by only selecting a category and pressing the submit button, the user should obtain all service providers that fit that category regardless of their subcategory. The more of these search criteria the user fills in, the more targeted the search will be. The search result should appear in the form of a web page listing the names of all service providers that match the entered criteria. A link to the names of these providers, should provide full information about each in a popup window.