# Design for new VERCON/RADCON Components

**Part II**

**Prepared By**

*Samhaa El-Beltagy*

**November, 2005**

# Table of contents

## 1 Introduction

The report titled "*Design for new VERCON/RADCON Components: Part 1*" presented the design of three components that were identified as requiring further design in the report: "*VERCON Changes and Enhancements: A Follow up report based on the Information Needs Assessment of VERCON stakeholders*". This report continues on to present the design of two more components which are the:

- Farmers Problems Solution Monitoring Module
- Extension of Farmers' Problems Descriptors Module

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 Farmers Problems Solution Monitoring Module

### *2.1 Task Description*

Design and develop a component to keep track of when a problem was posted and to automatically notify any concerned persons of the delay. The component will allow research Institutes to monitor posting of problem solutions, and to take over answering of a problem in case one is not available at the research station, or if the solution is not posted for more than three days.

To achieve the desired functionality, entered problems should be time stamped. Whenever a solution is not posted for more than three days, then an email should be sent to appropriate personal. Two types of users should be introduced to define users of this system. The first is a super user for this module. Such a user gets notified of all problems that fit the above mentioned criteria. He/she can also view problem details and post problem solutions. /the other type of user will be defined by problem type and/or by region such that if a problem is entered for some given problem type, in some given region the system can then notify all users that fit that type whenever action is needed.

**Cross reference(s)**
RAD/05/2,Section 3.1.1
RAD/05/3,Section 3.1, recommendation 1

## *2.2 Database Changes*

The table <***Problems***> in the Global VERCON database currently contains all details of any input problem including a timestamp denoting the date on which the problem was reported. This will be used extensively ..

1. Introduce a new table <PendingAndUnable_to_Solve>. This table will store any problems to which a research station is unable to find a solution or which a solution has been delayed for more than *n* days. The table should have the following fields:
    a) PID : The id of the encountered problem. This will be a foreign key mapping to the field P_ID in table <Problems>
    b) uid: The id of the user who indicated that the research station can not solve this problem. The will be a foreign key mapping MEMBER_ID in table < FORUM_MEMBERS>
2. Create a new table <Delayed_Problem_Users>. This table will be used to store information about whom to notify when the solution to a problem is delayed or unavailable. The table should have the following fields:
    a) uid: The id of the user who indicated that the research station can not solve this problem. The will be a foreign key mapping MEMBER_ID in table < FORUM_MEMBERS>
    b) DIR_ID: The id of the directorate for which this user is assigned. 0 indicates any directorate
    c) GOV_ID: The id of the governorate for which this user is assigned. 0 indicates any governorate

## *2.3 System changes*

Create a script that checks the database on a daily basis and does the following:

1. Obtain a list of all unsolved problems. This can be found by querying the <Problems> table for all records who have an empty or null P_SOLUTION_DATE.
2. For each of the returned records, check to see if *n* days have elapsed since the reporting of this problem where n is a developer defined number that will initially be set to 3. This can be done by carrying out the following:
    a) Obtaining the current date
    b) Subtracting the reported date which can be found in the field P_DATE in the retrieved record, from the current date.

    If the result is smaller than *n*, then process next record else
3. 3 days or more would have passed since the reporting of this record. The reason an equivalency check is not immediately made is to overcome any case in which there is a failure to run the script on some given day. If the script is guaranteed to run on a daily basis, then an equivalency check should be made. Store the id of the problem in the <PendingAndUnable_to_Solve> table with a uid of -1 to indicate that the time since posting this problem has surpassed *n* days as opposed to the research station indicating that it could not be solved.

4. Obtain a list of user IDs that should be notified by email of this problem. The IDs of these users can be obtained from the table <Delayed_Problem_Users> by checking for:
   a) all users that have a similar directorate and governorate as in the retrieved record (same DIR_ID, GOV_ID)
   b) all users that have a 0 in DIR_ID
   c) all users that have a directorate similar to that in DIR_ID and a 0 in the GOV_ID field
5. Email each of the users of problem details. The email for the users can be retrieved from the < FORUM_MEMBERS> using the users' Ids.

### *2.4 Interface Changes*

1. The first change is regarding giving authorized staff accessibility to delayed and unsolved problems. To achieve this, an extra link should be provided to allow appropriate staff to view and publish solutions to delayed problems. Following this link should result in the display of a page containing all problems that fall under the authority of a logged in user. To this end, a script for checking and retrieving the problems that the logged in user can view, should be developed. Further more, following a link of any of the displayed problems should result in problem details being displayed as well as a text box for entering a solution. These interface changes can be done by simply customizing and extending the current interface used for viewing and publishing problem solutions. Once a solution has been published, it should be deleted from the <PendingAndUnable_to_Solve> table. The solution, and the date of the solution was posted, should be stored In the problem's record in the <Problems> table. The flag *P_publish* flag should also be set to true.
2. The second change is related to providing the means whereby a user can indicate that a solution to some given problem is unavailable. To do so, the interface for viewing problem details and publishing a solution for research station users, should be modified so as to include a button for stating that a solution for the displayed problem is unavailable. If the user presses this button, then a confirmatory message asking the user whether s/he is sure a solution is unavailable for the displayed problem. If the user's answer is yes, then the problem's ID should be placed in the <PendingAndUnable_to_Solve> along with the ID of the logged in user. The problem should then no longer be made visible to the logged in user.

# 3 Extension of Farmers' Problems Descriptors

## *3.1 Task Description*

An entered farmer's problem is currently represented in terms of a general category, a sub category, a crop, a variety, the location of the problem and a problem description expressed in free text. The general category is used to describe the type of the problem (Administrative, Production, Marketing) and the sub-category is used to further elaborate on what the selected problem is in relation to (for example, if production is the general category, then pest control, fertilization, or irrigation, among others, can be the sub category). The goal of this task is to extend the currently existing descriptors listed above in order to define a problem on a finer level of detail. This will enhance the search service provided by the system. Specifically, we're looking to extend on the descriptors that fall under the general and specific sub-categories.

The original suggestion was to aim to use expert system options as the basis for this modification. The design presented herein is independent of the actual descriptors that will be used. These should be specified by examining the set of already entered problems and abstracting descriptors from those, as well as examining the applicability of expert system diagnostic descriptors to problems.

To carry out this task, the database needs to be modified in a way that would allow for the addition of new descriptors in a flexible manner that would not disrupt the current design. The following interfaces will also need to be changed:

1. Extension workers problem entry screen
2. Solution providers problem viewing and publishing screen
3. End users search interface.

Another interface through which administrators can define descriptors also has to be introduced.

In the following subsections, details related to these changes are provided.

**Cross reference(s)**
RAD/05/2,Section 3.1.1
RAD/05/3,Section 3.1, recommendation 2

### *3.2 Database Changes*

A new table <Detailed _Descriptors> should be introduced. This table should have the following fields:

- Did: A unique id for the introduced descriptor that also serves as a primary key.

- S_ID: A foreign key mapping to the ID of the sub category under which this descriptor falls. Specifically, this key should map to field PROBLEM_SUBCAT _ID in table <ProblemSubCategories> .
- Parent_Did: In case this descriptor is a child of another descriptor entered in this table, then this field will be used to point to it's parent descriptor, otherwise, it should be left empty.
- Level: This is a number that indicates that depth of this category after some given sub category. So, if a descriptor is entered as the immediate child of some given subcategory, it will be given a level of 1, its child will have a level of 2, and so on.
- Cat_Description: Text representing this descriptor

Another new table <Detailed_Problem_Descriptors> should be defined. This table can be thought of as an extension to table <Problems>. The table should have the following fields:

- PID: The ID of the problem being described. This is a foreign key mapping to field P_ID in table <Problems>
- DID: The ID of the very specific descriptor, used to describe the problem whose id is specified in field PID

A single problem can have more than descriptor in this table.

### *3.3 Interface Changes*

#### 3.3.1 Extension Worker's Problem Entry Screen

As stated before, the current entry screen allows extension worker to specify and general and specific subcategory for the problem being entered. This should remain the same, and a single problem should only have one general category and one specific subcategory. However, should extension workers wish to elaborate on the problem even further then s/he should be able to do so using , any existing descriptors that fall under his/her selected category and subcategory. To enable the extension worker from doing this, a link or a button with the text "enter more details" should appear in a prominent place underneath the currently available problem details data entry menus. Upon pressing this link or button, the following should take place:

1. The database table <Detailed *Descriptors> should be searched for all entries that have the selected sub*category as its S_ID and a level of 1.
2. If no entries are found, then a message "No further details can be entered for this problem" should be displayed and the script should terminate.
3. In case entries are found, then these should be displayed in a dropdown list the location of which is best determined by the developer of this system.
4. If the user makes a selection from this list, then the table <Detailed *Descriptors*> should be searched for all entries that have the selected descriptor as their Parent_Did.
5. Steps 3, and 4 should be repeated until:
    a. No further descriptor can be found in the database

   b. The user submits the problem description
6. When the user submits the problem description, the id of each of the selected descriptors should be stored in the table <Detailed_Problem_Descriptors> along with the problem's ID.

### 3.3.2 Solution Providers Problem Viewing and Publishing screen

Solution providers should be provided with the means whereby to add descriptors to entered problems for the following 2 reasons:

1. The descriptors entered by the extension workers do not really represent the problem as specified by its free text description. In this case, solution publishers should be able to modify these to actually reflect the problem and solution's content.
2. The extension workers did not provide any descriptors when they could have been added. In this case, the solution publishers may want to do so to make it easier for end users searching the database, to locate their problems.

To achieve this,  the solution publishing screen should now display all the descriptors entered by the extension worker, along with the means for modifying them. This can be done using the following steps:

1. When displaying a problem for publishing, use its id to retrieve the most specific descriptor entered for that problem using its id in the <Detailed_Problem_Descriptors> table.
2. The retrieved id, can then be used to obtain its parents one by one from the <Detailed_Descriptors> table*.*
3. The retrieved tree, should then be presented to the user.
4. For the user to be able to modify the descriptors, all siblings in any given level should be retrieved from the <Detailed_Descriptors> table in a manner similar to that described in steps 3 to 5 in section 3.3.1
5. When the user submits the problem description, the id of the last selected descriptor, should be stored in the table <Detailed_Problem_Descriptors> along with the problem's ID.

### 3.3.3 End users search interface

The end user search interface should be modified so as to allow end users from specifying what they are looking for on a finer level of detail.  To do that, the following steps should be carried out:

1. Whenever a user selects a subcategory, the database table <Detailed *Descriptors> should be searched for all entries that have the selected sub*category as its S_ID and a level of 1.
2. In case entries are found, then these should be displayed in a dropdown list the location of which is best determined by the developer of this system.
3. If the user makes a selection from this list, then the table <Detailed *Descriptors*> should be searched for all entries that have the selected descriptor as their Parent_Did.
4. Steps 3, and 4 should be repeated until:

    a. No further descriptors can be found in the database
    b. The user submits the query
5. When the user submits a query, the id of the most detailed descriptor entered should be checked against ids in the <Detailed_Problem_Descriptors> table. Problem Ids associated with that descriptor should  be used to problems that match that descriptor. These should then be  filtered using other entered criteria, and what’s left should be displayed to the user in the customary manner.