Crop Rotation Economic and Environmental Impact Decision Aid (CREEDA): A COM-Based Application

A PROJECT REPORT
Submitted to
Oregon State University

Song Chang

In partial fulfillment of the requirement of
The requirements for
Degree of
Master of Science in Computer Science

Graduate Committee:

Dr. Walter Rudd, Major Advisor
Dr. Toshimi Minoura, Minor Advisor
Dr. Curtis Cook, Committee Member
Dr. Jeffrey Steiner, Committee Member

ACKNOWLEDGEMENT

My earnest thanks are given to Dr. Walter Rudd of Computer Science Department, Oregon State University, my major advisor, who introduced me to this project, and Dr. Toshimi Minoura of Computer Science Department, Oregon State University who helped me generously on my paper and spent so much time with me to discuss the interesting problems on COM. I would like to thank Dr. Jeffrey J. Steiner of USDA-Agricultural Research Service, who initiated this project, skillfully managed the whole project and supported me financially to work on this project. I would like to thank Kevin Boyle of USDA-Natural Resources Conservation Service, who directed me to solve the hardest problems, and inspired me with his enthusiasms on this project. I would like to thank Hal Gordon, Thoms Gohlke and Steve Campbell of USDA-Natural Resources Conservation Service for the rich information they provided generously. Finally, my deepest love and all gratefulness go to my dear wife Lingzhi Xu for her help and supports during the hardest times.

CONTENTS

ABSTRACT

CREEDA (Crop Rotation Economic and Environmental Impact Decision Aid) is a Windows application for assessing economic and environmental impacts of agricultural activities. In implementing it, we extended the `ProCosts` database and incorporated the RUSLE (Revised Universal Soil-Loss Equation) application and the SCI (Soil Conditioning Index) application as COM components. `ProCosts`, developed by USDA-NRCS (Natural Resources Conservation Service), is used to estimate profits and costs for a multi-year crop rotation. The core component of CREEDA was implemented with MFC (Microsoft Foundation Classes) and ATL (Active Template Library), and consists of the following layers: the Presentation Layer (PL), the Business Logic Layer (BLL), and the Data Access Layer (DAL). We built the Presentation Layer with MFC by using Visual C++ and other layers as COM components by using ATL. The back-end database built with MS SQL Server is accessed with Microsoft ActiveX Data Objects (ADOs).

1. Introduction

Environmental impacts of human endeavors can no longer be ignored. Some researchers in the Integrated Cropping Systems Research Team at the USDA-Agricultural Research Service (USDA-ARS) in Corvallis, OR and those in the USDA-Natural Resources Conservation (USDA-NRCS) Service Technology Team in Portland, OR realized the necessity for a software tool that analyzes simultaneously both economic and conservation impacts of multi-year crop rotations. They believe that such a software tool will help economists, conservation planners, and farmers make management decisions based on scientific data.

The `ProCosts` program, which was developed by Kevin Boyle at USDA-NRCS, calculates the profits and costs for multi-year crop rotations. A multi-year crop rotation is a sequence of farming activities in which different crops are grown over a period of multiple years. The `ProCosts` program performs profit-and-cost analysis based on the mathematical formulas described in the reference manual "Commodity Costs and Returns Estimation Handbook". The operational data such as the amounts of seeds, fertilizers, and labor hours consumed and the economic background data such as the prices of the crops, fertilizers, machines, and labor hours are stored in a relational database. The `ProCosts` program, written in Visual Basic, provides a user interface for entering operational and background data and generates a budget report for a crop rotation selected. This program utilizes the Component Object Model (COM) technology.

The Soil Conditioning Index (SCI), which was developed with Microsoft Excel by USDA-NRCS, can be used to evaluate the effect of a conservation practice on soil organic-matter. It computes the amount of residue left, soil disturbance rate and soil erosion. The SCI is a weighted-average of the three factors: Field Operation (FO), Organic Matter (OM) and Erosion (ER), which are calculated from the amount of residue, soil disturbance rate and soil erosion

RUSLE, which was developed by USDA-ARS, was used to predict soil-loss caused by water erosion. It calculates six factors: R, C, K, P, L, and S. The result of RUSLE is the product of these six factors. The factor R reflects the effect of local weather

4. We defined three user levels: economist and researchers on the top, conservation planners in field offices in the middle, and basic users at the bottom, who performed different tasks with different user interfaces.
5. We used ADO (Active Data Object) objects – a set of COM components and interfaces provided by Microsoft to access the database which was implemented with MS SQL Server and carry data among components.
6. We provided 600 pages of source codes and a comprehensive package which include `ProCosts`, `CREEDAMain`, `CREEDAInputEditor` and `CREEDALinker`.

This system was tested by experienced agronomists. The test showed that users could benefit from CREEDA in these aspects:

1. Users do not have to re-enter crop rotation data.
2. The users of three levels can perform their tasks cooperatively during the decision-making process, which are important to find the most economically feasible and environmental protective combinations of production practices and technologies for a farm.
3. Friendly GUIs help users to learn the system quickly.
4. In addition to the economic report generated with `ProCosts` and those generated with SCI and RUSLE, additional reports about crop management can be generated (e.g., the report about fertilizer applications in a rotation is useful for water quality analysis).

Currently the integration for CREEDA implementation is not perfect, but the value of such an integration prototype was valuable and we are going to develop a more robust framework for integration in the future.

2. System Overview

The CREEDA application consists of the following four programs: ProCosts, CREEDAMain, CREEDAInputEditor and CREEDALinker. ProCosts was developed by USDA-NRCS independently; CREEDAInputEditor and CREEDALinker were auxiliary tools written in Visual BASIC; CREEDAMain was the focus of our effort. All programs use the same database. The architecture of CREEDA is given in Figure 1.

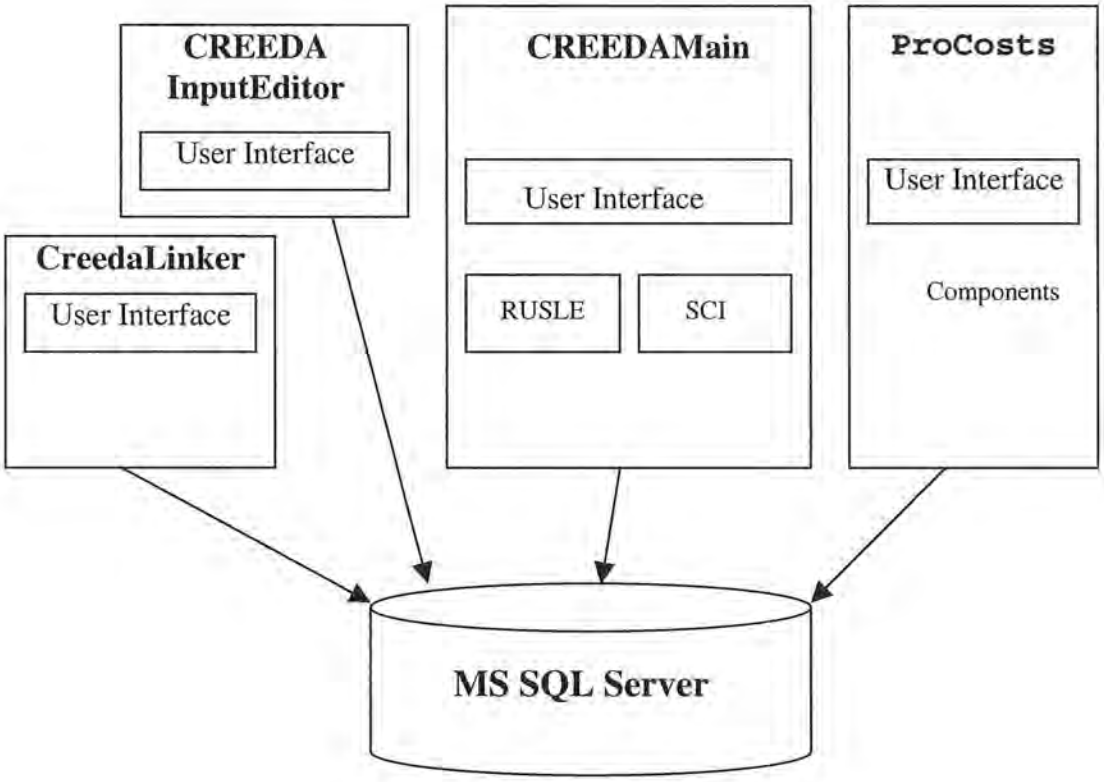

Figure 1 Architecture of CREEDA

All functions were divided to three levels vertically: researcher and economist level, conservation planner level and basic user level. In this chapter, we will introduce these levels.

2.1 Functions of CREEDA

2.1.1 Functions of ProCosts, RUSLE and SCI

ProCosts provides a user interface for users to enter data of farming activities, the prices of materials and products, and generate budget reports in terms of costs and returns.

RUSLE calculates and reports soil erosion result based on site-specific data, and the information of crop rotating and other farming activities.

SCI (Soil Condition Index) calculates a composite index based on site-specific data, the result of RUSLE application, and the information of crop rotating and other farming activities. The index reflects the effect of the farming activities on soil organic matter.

2.1.2 Additional Features of CREEDA

In addition to the functions above, CREEDA has:

(1) A unified data model

In this system, the database of ProCosts was extended to accommodate data for RUSLE and SCI. The architecture of the new data model can be extended further easily. Detail of the data model is given in next chapter.

(2) Owner information management

New functions were provided for owner information management. An "owner" is defined as a farmer or his representative, such as a conservation planner running the application for the sake of the farmer. An owner may have multiple fields, for each of them he may perform a rotation defined in ProCosts. A rotation can be applied to multiple fields owned by various farmers. This system allows multiple owners to share the rotation information, the programs and the database but still be able to enter and manipulate their own data independently.

(3) More reports generated

In addition to reports generated by ProCosts, SCI and RUSLE, more reports can be generated: Irrigation Management, Crop Residue Management, Crop Rotation and Crop Management, Tillage Equipment and Tillage Sequence, Pest Management Inputs

and Crop Nutrient Inputs. All reports were generated in a format compatible with MS Excel.

(4) User Groups on Different Levels

Three user levels were defined and users of these levels were assigned different privileges. The users are: researchers and economists, conservation planners and basic users. They perform different tasks on different levels when run the system. The user's tasks and the data flow in CREEDA are shown in Figure 2.

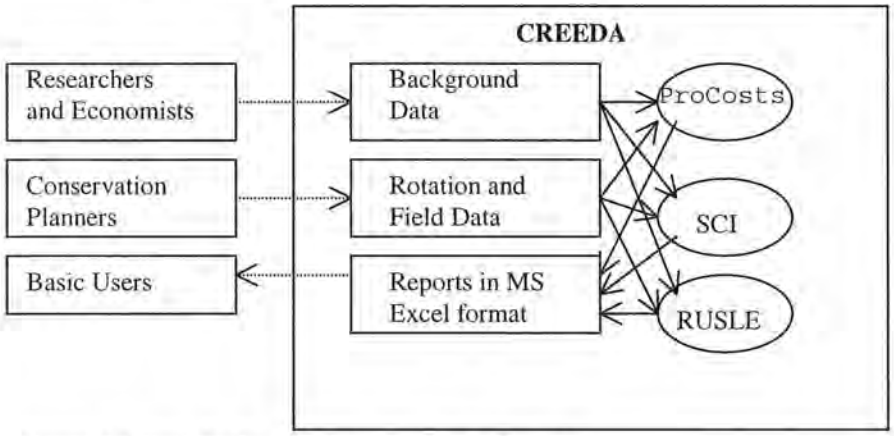

Figure 2 Users' Tasks and Data-Flow in CREEDA

Researchers and economists can modify the background data of budgets and environmental impacts. The background data are read-only to other users.

Conservation planners build budgets, enter field data, link conservation impacts to inputs and outputs of a rotation defined in a budget, and generate reports.

Basic users read or print the reports generated for them. In CREEDA, reports are provided as MS Excel files.

All user interfaces are implemented as native Windows interfaces. Users use Drag-and-Drop operations to manipulate the data. Four programs provided different functions:

`ProCosts` is used to enter the background data for economic analysis, generate budgets and economic reports.

CREEDAMain is used to enter field data, and generate conservation impacts reports.

CREEDAInputEditor is used to enter background data for conservation impacts.

CREEDALinker is used to associate the inputs in ProCosts with conservation impacts.

2.2 Hardware and software requirements

This application requires Windows operating systems (NT, 95, 98). For Windows 95, DCOM must be installed. MS SQL Server or its client version Microsoft Data Engine (MSDE) must be installed.

3. The Architecture of CREEDA

In this chapter, we introduce the data model, then focus on the architecture of CREEDAMain.

3.1 Data Model of CREEDA

In CREEDA application, ProCosts, CREEDAMain, CREEDAInputEditor and CRREDALinker share the database that is an extension of the ProCosts database. The most important classes in `ProCosts` database are `ROTATION`, `BUDYEAR`, `INPUT`, `OUTPUT`, `BUDOPERATION`, `BUDINPUT`, and `BUDOUTPUT`. `ROTATION` is the plan for rotating crops. `BUDYEAR` is the plan of farming activities in a rotation year. `BUDOUTPUT`s are products of a rotation year. `BUDINPUT`s are agricultural operations or materials applied in a rotation year. `OUTPUT`s are pre-defined products. `INPUT`s are pre-defined agricultural operations or materials.

Their relationships are given in Figure 3.

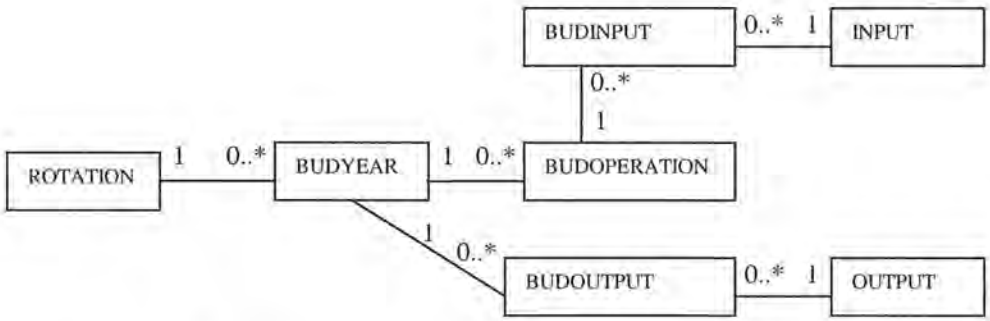

Figure 3 The Relationships among the Classes in ProCosts

`INPUT`s defined in ProCosts have only economic attributes, such as salvage values, prices, etc. The following five classes were defined in CREEDA for various conservation impacts (impact attribute groups) incurred by `INPUT`s: `MACHINERY`, `IRRIGATION`, `RESIDUE-MANAGEMENT`, `PESTICIDE`, `FERTILIZER`. Class `CROP` was defined for the conservation impacts incurred by `OUTPUT`. `MACHINERY` is used to store the soil disturbance rate of an operation. `IRRIGATION` is used to store the amount

of water applied and the irrigation times. RESIDUE-MANAGEMENT is used to store the ratio of residue removed with an operation. FERTILIZER is used to store the depth to which a fertilizer is applied. CROP is used to store the properties of a crop. Currently no information is stored in PESTICIDE. These classes were simply denoted as CREEDAINPUT or CREEDAOUTPUT in the following discussions. The relationships between INPUT or OUTPUT classes and CREEDAINPUT or CREEDAOUTPUT classes are given in Figure 4.

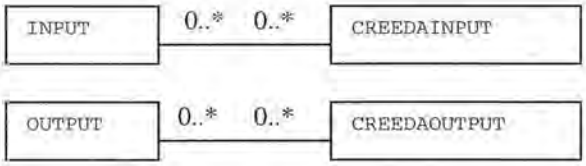

Figure 4 Relationships Between INPUT or OUTPUT and CREEDAINPUT or CREEDAOUTPUT

The following two classes were defined in CREEDA for owner information management and site information:

FIELD: it refers to a site with hydrologic and geologic properties.

OWNER: it refers to the person who owns fields.

The relationships between ROTATION, FIELD and OWNER are given in Figure 5.

Figure 5 Relationship Between ROTATION, FIEL and ROTATION

When a ROTATION is applied to a FIELD, all of its BUDINPUTs and BUDOUTPUTs are appended with impact attribute groups according to the links between BUDINPUT (or BUDOUTPUT) and CREEDAINPUT (or CREEDAOUTPUT) through INPUT (or OUTPUT). The following classes were defined for these appended attribute groups: BUDFD-MACHINERY, BUDFD-IRRIGATION, BUDFD-RESIDUE-MANAGEMENT, BUDFD-PESTICIDE, BUDFD-FERTILIZER, and BUDFD-CROP. These classes are denoted as CREEDABUDINPUT and CREEDABUDOUTPUT in

following discussions. Relationships between CREEDABUDINPUT, CREEDABUDOUTPUT and other classes are given in Figure 6:

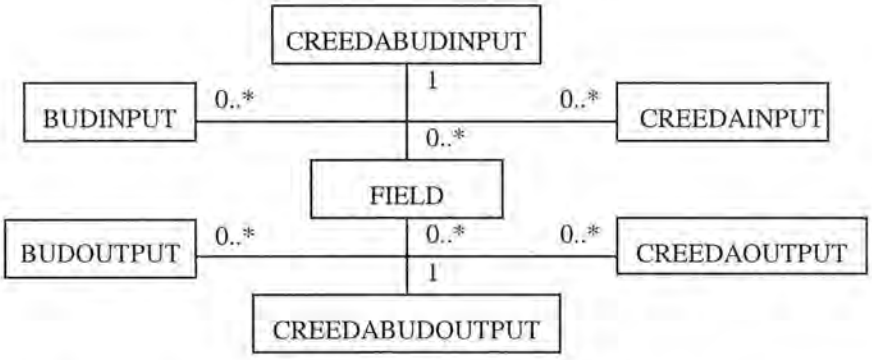

Figure 6 Relationships of Class CREEDABUDINPUT and CREEDABUDOUTPUT with other classes.

The program CREEDAInputEditor is used to add, delete or modify INPUT and OUTPUT objects. The program CREEDALinker is used to add or remove relationships between INPUT (or OUTPUT) objects and BUDINPUT (or BUDOUTPUT) objects and. In the rest part of this chapter, we focus on CREEDAMain.

3.2 User Interface of CREEDAMain

Layout of the user interface is given in Figure 7. Here we described the UI elements and user's operations on them:

3.2.1 UI Elements

FrameWnd is the container of all the following windows:

InputGroupWnd has six buttons each of which is linked to a window for an impact attribute group. It has a SearchBoxWnd including SearchText and SearchButton. InputListWnd is contained in InputGroupWnd and displays CREEDAINPUT or CREEDAOUTPUT.

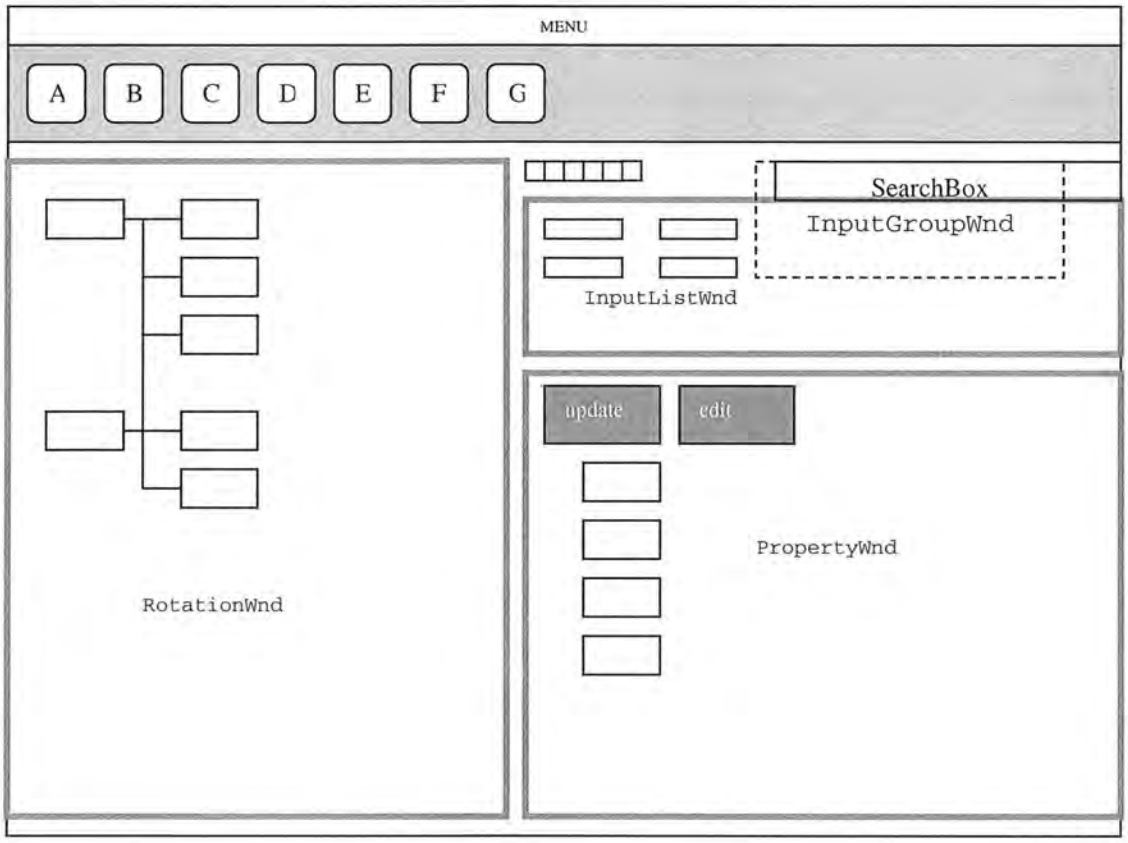

Figure 7 Layout of the main window

`RotationWnd` displays fields and associated rotations as a tree. The structure of the tree is:

```
FIELD
    -ROTATION
        -OPERATION
            -BUDINPUT
                -CREEDABUDINPUT
        or
        -OUTPUTGROUP
            -BUDOUTPUT
                -CREEDABUDOUTPUT.
```

PropertyWnd displays attributes of a CREEDABUDINPUT or CREEDABUDOUTPUT selected in RotationWnd and allows users to update the attributes. It displays a button for editing a FIELD entity if a FIELD entity is selected in RotationWnd. It displays nothing if other objects are chosen.

Two SpiltterBar split the windows vertically and horizontally.

ToolBar has seven buttons for different functions, as shown as A through G in Figure 7. They are:

A: Exit	B: Open a budget	C: Current Report
D: New Report	E: Trash Can	F: List of Rotations
G: List of Fields		

User's operations on each window:

FrameWnd

Change size.

InputGroupWnd

Select an impact attribute group by clicking the buttons.

SearchBoxWnd

(1) Enter search text.

(2) Click the SearchButton, locate the CREEDAINPUT searched for in InputListWnd.

InputListWnd

Select a CREEDAINPUT or CREEDAOUTPUT by clicking the mouse.

RotationWnd

(1) Expand an object by clicking it.

(2) Select an object by clicking it which attributes will be displayed on PropertyWnd if available.

PropertyWnd

(1) Change and update the attributes of a CREEDABUDINPUT or CREEDABUD objects.

(2) Go to a dialog box for displaying and editing FIELD attributes.

SplitterBar

Change its position by pressing down the left button of the mouse and dragging it.

User's operations between windows:

1. When the size of FrameWnd changes, the size of other windows in FrameWnd will change too.
2. In InputListWnd, drag a CREEDAINPUT or CREEDAOUTPUT entity by pressing the left-button of mouse, and drop the entity to the RotationWnd by loosing the pressed mouse button. A CREEDABUDINPUT or CREEDABUDOUTPUT will be created and added to the RotationWnd.
3. In RotationWnd window, drag a CREEDABUDINPUT, CREEDABUDOUTPUT, or ROTATION by pressing the left-button of mouse, and drop the item to TrashCan (E on Figure 7) by loosing the pressed mouse button. The objects will be removed from the RotationWnd.

3.3 The Architecture of CREEDAMain

CREEDAMain consists of the following three layers: the Data Access Layer (DAL), the Business Logic Layer (BLL) and the Presentation Layer (PL).

Now we explain these layers.

3.3.1 Data Access Layer (DAL)

The functions of DAL are:

(1) Handle transactions for other layers. The boundary of a component in this layer is also the boundary of any transaction involved in CREEDAMain and other layers do not see any transaction.

(2) Prevents programmers of other layers if there is any from writing codes "on the fly" by providing a unified interface for database access. For example, it hides the process of making and releasing ODBC connections from components in other layers, and thus prevents them from abusing the usage of ODBC connections, which are often a limited resource.

3.3.2 Business Logic Layer (BLL)

Retrieving and updating data operations were implemented in this layer. So were the RUSLE and the SCI. The main tasks of this layer were:

(1) Retrieve `BUDGET`, `ROTATION`, `YEAR`, `BUDINPUT`, `BUDOUTPUT`, `INPUT` and `OUTPUT` objects,

(2) Retrieve `CREEDAINPUT` and `CREEDAOUTPUT` objects,

(3) Retrieve `CREEDABUDINPUT` and `CREEDABUDINPUT` relationships,

(4) Remove `CREEDABUDINPUT` and `CREEDABUDINPUT` relationships,

(5) Calculate SCI and RUSLE,

(6) Generate reports.

3.3.3 Presentation Layer

The user interfaces define in 3.2 was implemented in this layer. Detailed of the implementation of these layers is given in next chapter.

4. Implementation of CREEDAMain

In this chapter, we introduce how the three layers of CREEDAMain were implemented.

4.1 Implementation of the Data Access Layer

There was only one component in this layer: `vbdal` which CREEDAMain shares with ProCosts. It had an interface `_CDataAccess` with two methods: `GetRecordset` and `ExecQuery`. `GetRecordset` returns query result as an object with `_Recordset` interface. `ExecQuery` does not return query result.

Active Data Object (ADO) was used to access the database and commute query results. There are two ways of using ADO. One is to keep its connection to database all the time. The other one is to disconnect the connection right after the query result is returned. For system scalability, we adopted the disconnected one because we did not want numerous users to connect to the database simultaneously which otherwise have to maintain the same number of connections. Therefore a copy of the data is always stored in the Presentation Layer, which can be updated on time, because each user need only update the data which he owns exclusively. The background data and budget information only need to be updated in occasions and we can avoid conflicting with other users by choosing a proper time of updating. Strict concurrency control was implemented in MS SQL server which we used for the back-end database.

4.2 Implementation of the Business Logic Layer

This layer was implemented as COM components. Each COM component has two parts: interfaces and implementations.

4.2.1 Interfaces in BLL

`ICREEDAInputs:`

It is used to retrieve `CREEDAINPUT` objects from the database. It may retrieve an object by its ID, or a list of objects in an attribute group.

`ICREEDAOutputs:`

It is used to retrieve CREEDAOUTPUTs from the database. It may retrieve an object by its ID, or a list of objects in an attribute group.

IProcosts:

It is used to retrieve BUDGETs, ROTATIONs, BUDYEARs, BUDINPUTs, BUDOUTPUTs, INPUTs and OUTPUTs.

IInputsForAProCosts:

It is used to retrieve BUDFDINPUTs. It takes two parameters: ID of FIELD and ID of BUDINPUT.

ICropForAProCost:

It is used to retrieve CREEDABUDOUTPUTs.

IEditFdBudInfo:

It is used to edit, delete or modify BUDFDINPUT, BUDOUTPUT and FIELD.

ISCIModel:

It is used to calculate SCI.

IComCalcLS:

It is used to calculate LS factor of RUSLE.

ICREEDAFields:

It is used to retrieve FIELD.

ICREEDARepoter:

It is used to generate reports.

IUserManager:

It is used to retrieve owner information.

These interfaces were defined in IDL (Interface Define Language). If a parameter of any methods was an interface, a COM object was passed. It must be noted that MS IDL usually does not allow a COM object to be passed by its value, therefore the COM objects are usually passed by reference. In the case of DCOM, the marshaled interface is passed. In this application, COM object `Recordset` defined in ADO was often passed as a parameter.

4.2.2 COM components in BLL

Components in BLL implemented some of the interfaces given above.

`CropForAProCost`

It implemented these interfaces: `IcropForAProCosts` and `IeditFdBudInfo`.

FertForAProCost, IrriForAProCost, `MachForAProCost`, `ResForAProCost`, `PestForAProCost`

All of them implemented two interfaces: `IinputsForAProCosts` and `IeditFdBudInfo`.

`FertInputs`, `IrriInputs`, `MachInputs`, `ResInputs`, `PestInputs`

All of them implemented an interface `ICREEDAInputs`.

`Outputs`

It implemented an interface: `ICREEDAOutputs`.

`CREEDAFields`

All of them implemented two interfaces: `ICREEDAInputs`, `IEditFdBudInfo` and `ICREEDAFields`.

`CREEDACities`

It implemented an interface `ICREEDAInputs`.

`ProCosts`

It implemented an interface `IProCosts`.

`SCIModel`

It implemented an interface `ISCIModel`.

`ComCalcLS`

It implemented an interface `IComCalcLS`.

`CREEDAReporter`

It implemented an interface `ICREEDARepoter`.

`UserManager`

It implemented an interface `IUserManager`.

They were implemented as common in-process COM components and loaded to the memory space of CREEDAMain. Error-handling codes contributed to a big part of their codes. They exchanged `Recordset` with the DAL and the PL. Most of them encapsulated calls to stored procedures defined in the back-end database.

4.2.3 DCOM implementation

The advantage with COM is that it hides the location information from common users. Microsoft provided a facility DCOMCNFG that allowed you to configure the system so that we could access a remote COM object without changing any code. Another way without using DCOMCNFG was to modify the system registry. Since we did not have privilege in the lab to run DCOMCNFG, we used the latter way. The detail was given in the book written by Richard Grimes[2].

Microsoft provided a DLL library `ole32aut.dll`. With this library, any COM object that uses only so-called "OLE-compatible" parameters does not need to provide stub\proxy codes if it implements `IDispatch` interface, or its type library is registered. `IDispatch` interface can help system get to information about the parameters without

type library. We used only OLE-compatible parameters, therefore, there was no explicit stub\proxy code in CREEDAMain.

4.3 Implementation of the Presentation Layer

4.3.1 Classes

There are two kinds of classes: those representing user interface components and those representing a data item. The latter are exchanged among UI components.

Two classes for data were defined as:

(1) `CInputItem`: it stored IDs and type information of `CREEDAINPUT` or `CREEDAOUTPUT`. It was also used to store `FIELD` dragged and dropped to the `RotationWnd`.

(2) `RotationItem`: it stored IDs and type information of the items listed in `RotationWnd`.

Five main UI components were defined:

`CMainFrame`

It was a subclass of `CFrameWnd` in MFC and required for a MFC application. It served as the main container of other components and coordinated the communication between the tool bar, the view windows, and the menus.

`CChildView`

It was a subclass of `CWnd` and contained the rotation window, property window, input group window and served as the main display area. It was responsible for coordinating the communications between the child windows and the main frame window. It also contained two splitter bars and was responsible for layout management. This window is required by MFC architecture and has no direct correspondence in requirement specification.

`CRotationWnd` and `CRotationTree`

CRotationWnd was a subclass of CRotationTree, which was a subclass of CTreeCtrl of MFC. CRotationWnd was responsible for constructing the hierarchic structures of rotations linked to fields. CRotationTree was responsible for not only displaying the hierarchic structure, but also temporarily stored some information of the items.

CPropertyWnd

It was a subclass of CWnd and contained sub-windows (CPropView) for displaying properties of various kinds of rotation items and fields. It also responded to user's changing window size operation.

CPropView

It was a subclass of CWnd and super class of several classes for displaying the properties of various types of rotation items. It was an abstract class, therefore its responsibilities were of its subclasses too. It managed the states of two buttons – "update" and "reset".

CInputGroupWnd

It was a subclass of CWnd. It contained property-set-list windows (CInputListWnd) for displaying various groups of property sets. It allowed user to choose displayed groups and communicated with the view window to handle the drag-drop operation between the window for any property set group and the rotation window. It has a search box that handled searching operation on current displayed property-set-list windows.

CInputListWnd and CInputListCtrl

CInputListWnd was a subclass of CInputListCtrl. They list all items of a kind of CREEDAINPUT or CREEDAOUTPUT. Users can drag an item from it and drop the item to the Rotation window. It notified its parent window when a user started to drag any of its items.

4.3.2 Mapping Classes to Elements in Design

Main classes in PL can be mapped to elements defined in 3.2.

Classes defined in PL	Elements defined in the design
CmainFrame and CchildView	FrameWnd
CrotationWnd and CrotationTree	RotationWnd
CPropertyWnd and CpropView	PropertyWnd
CinputGroupWnd	InputGroupWnd
CInputListWnd and CinputListCtrl	InputListWnd

4.4 Implementation of Database

The database was implemented on MS SQL server and could be distributed with either MS SQL Server or Microsoft Data Engine (MSDE).

As we said, most of operations of BLL were actually implemented in the back-end database as stored procedures. Stored procedures in MS SQL Server were written in T-SQL (Transactional SQL). An example of stored procedures is given here:

```
    CREATE PROCEDURE [CREEDA_calc_mach_per_year] (@fieldID as int)
AS
SELECT CREEDAFIELD.FieldID,
    SUM(CREEDAMACHINERY.A_INVERT + CREEDAMACHINERY.A_MIX +
    CREEDAMACHINERY.A_LIFT + CREEDAMACHINERY.A_SHATTER +
    CREEDAMACHINERY.A_AERATE + CREEDAMACHINERY.A_COMPACT) AS SUMMACH,
    BudSysToTime.TimeName as TimeNum,
    BudSysToTime.TimeName
    FROM BudSysToOperation INNER JOIN CREEDAFIELD
    INNER JOIN CREEDACITY
    ON CREEDAFIELD.CITYCODE = CREEDACITY.CITYCODEID
    INNER JOIN CREEDABudSysEnttoField
    ON CREEDAFIELD.FieldID = CREEDABudSysEnttoField.FieldID
    INNER JOIN BudSysToEnt
    ON CREEDABudSysEnttoField.BudSysToEntID = BudSysToEnt.PKId
```

```
INNER JOIN BudSysToTime
ON BudSysToEnt.PKId = BudSysToTime.CAREToEntPractId
ON BudSysToOperation.CAREToTimeId = BudSysToTime.PKId
INNER JOIN Time
ON BudSysToTime.TimeId = Time.PKId
INNER JOIN CREEDAMACHINERY
INNER JOIN CREEDAFdBudMachInfo
ON CREEDAMACHINERY.OPERATIONID = CREEDAFdBudMachInfo.MACHID ON
CREEDAFIELD.FieldID = CREEDAFdBudMachInfo.CREEDAFieldID
INNER JOIN BudSysToInput
ON CREEDAFdBudMachInfo.BudSystoInputID = BudSysToInput.PKId AND
BudSysToOperation.PKId = BudSysToInput.CAREToOperationId
GROUP BY CREEDAFIELD.FieldID, BudSysToTime.TimeId,
BudSysToTime.TimeName
HAVING (CREEDAFIELD.FieldID = @fieldID)
ORDER BY BudSysToTime.TimeName
```

This stored procedure was used to calculate the sum of soil disturbance rates of all operations in a rotation year.

4.5 Testing

Because of limitation on resources and time, we did only necessary tests. For unit test, we tested components during the coding stage by debugging the codes.

For system test, we tested the typical scenarios. The typical scenarios were given as our test report in Appendix B.

4.6 Implementation as a Distributed Application

COM components are location-transparent, and therefore we can deploy them on different locations in a LAN or Internet (though it is hard to authenticate remote users through Internet) without being noticed by clients. A typical deployment plan we had tested with CREEDAMain is shown in Figure 8.

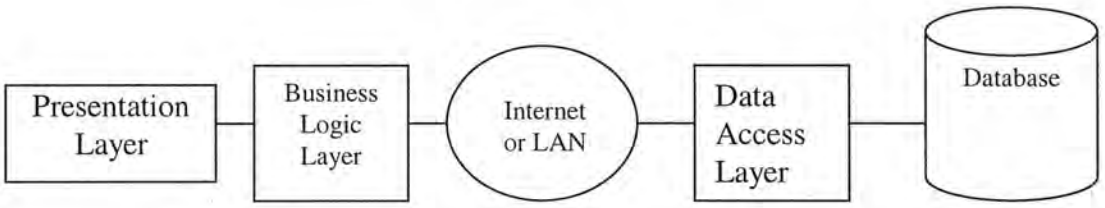

Figure 8 A typical distribution schema for CREEDA.

5. Summary and Future Work

5.1 Summary

The CREEDA application integrates the ProCosts application for profits and costs analysis of farming activities, the application RUSLE for soil loss prediction and the SCI application for evaluating the effect of framing activities on soil organic matter. The CREEDAMain program, which has incorporated RUSLE and SCI as COM components, was designed and implemented. We also extended the ProCosts database to accommodate the data for SCI and RUSLE. CREEDAMain program and ProCosts program, which was developed by USDA-NRCS independently, share the DAL layer. A new version of ProCosts could be merged to the CREEDA application in two days. Visual BSAIC program CREEDAInputEditor was implemented to enter conservation impacts data. Visual BSAIC program CREEDALinker was implemented to link inputs and outputs to their conservation impacts. Finally we explored the feasibility of deploying CREEDAMain in a distributed computing environment, and of reusing COM components in a web application.

In this project, however, we found that COM programming increased the complexity of system testing. Since the COM components in CREEDAMain have to interact with system services frequently, the number of "unexpected" system events was increase, and thus more testing was needed than that for a program without COM components.

5.2 Future Works

1. We currently provided only the batch files for installing the application. A commercial installation software, such as InstallShield, can make an installation easier.
2. In this application we did not use MTS. In fact, we dropped MTS objects to make an installation easier for desktop users. MTS facilities the installation and management of COM objects in a network environment. In Microsoft 's latest operating system

Windows 2000, MTS is a provided as a standard service. In the future, we can install the components in BLL and DAL on MTS.

3. We can move on to develop a web application based on this application by replacing the Presentation Layer with HTML and Active Server Page (ASP) files. Because COM components can be created and called in ASP, the web application should be able to reuse the components on the BLL and the DAL, both of which were designed for multiple users and implemented as COM components.
4. This application provides a framework that allows us to extend it and incorporate more conservation impact evaluation applications.

Reference

1. Grimes, Richard, Professional ATL COM Programming, 1998, Wrox Press Ltd.
2. Grimes, Richard, Professional Visual C++6 MTS programming, 1999, Wrox Press Ltd.
3. Gunter, David, Client/Server Programming with RPC and DCE, 1995, Que Corporation.
4. Pfleeger, Shari Lawrance, Software Engineering –Theory and Practice, 1998, Prentice Hall.
5. Rogerson, Dale, Inside COM, 1997, Microsoft Press.
6. Renard, K.G., Foster, G.R., Weesies, G.A., etc, Predicting Soil Erosion by Water: A Guide to Conservation Planning With The Revised Universal Soil Loss Equation (RUSLE), USDA-ARS, Agriculture Handbook Number 703, 1997.
7. Sinha, Pradeep K, Distributed Operating Systems –Concepts and Design, 1996, IEEE press.
8. USDA Report, "Commodity Costs and Returns Estimation Handbook", 1998, Ames, Iowa.

Appendix A Introduction to COM technology

COM is a binary standard about interfaces, invented and promoted by Microsoft. It is used mainly on Windows platform and neutral to programming languages. It can be implemented or accessed with various programming tools, such as Visual C++, Dephi, Visual BASIC, Java and ASP (Active Scripting Programming). A COM component has two parts: its implementation and interfaces.

1. Interface

An interface is a pointer to an array which elements point to function entries. Interfaces can be shared by components. Components that share an interface are binary-substitutable.

Interfaces are defined in IDL (Interface Defining Language) and processed by IDL compilers which generate stub \ proxy codes in C, C++, or Java. The codes generated for the server side are called stub, and those for the client side are called proxy. Proxy codes are responsible for encoding arguments to the input stream and decoding returned values from the output stream for the client side. Similarly, stub codes are responsible decoding arguments from the input stream and encoding returned values to the output stream for the server side. Marshaled interfaces are used to access remote objects. The concept of "remote object" changed with the development of COM technology. A "remote object" meant an object on another memory space or a remote host initially. Later it meant an object in a different apartment. The concept of "Apartment" will be introduced in the implementation of COM.

The basic COM interface `IUnknown` was defined in C and C++:

- **C++**

```
class IUnknown {
        virtual HRESULT __stdcall QueryInterface(REFIID
iid,
                                        void ** ppvObject)
= 0;
        virtual ULONG __stdcall AddRef() = 0;
        virtual ULONG __stdcall Release() = 0;
            };
```

- C

```
typedef struct IUnknown {
     IUnknownVtbl *pVtbl;
     } IUnknown;
typedef struct IUnknownVtbl IUnknownVtbl;
struct  IUnknownVtbl {
     HRESULT _stdcall (*QueryInterface)(IUnknown* this,
     REFIID iid,
      ULONG _stdcall (*AddRef)(IUnknown* this);
      ULONG _stdcall (*Release)(IUnknown* this);
     };
```

The memory layout of class `IUnknown` was given in Figure 1.

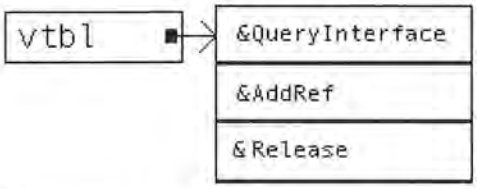

Figure 1 Memory Layout of class `IUnknown` Interface

Other interfaces must be derived from `IUnknown`. Therefore, all interfaces must have the methods defined in `IUnknown`. `AddRef()` is usually called by a client to signal the object that a client is using it. `Release()` is called by a client to signal the object that the client is done with it. When all clients are done, the object may release itself from the memory. It should be noted here that COM standard was actually made for C++. The definition in C was mapped from that in C++ according to the standard conventions of C and C++.

2. Implementation of COM objects

Naming service of COM is provided by SCM (Service Control Manager). COM run-time library is used to access the system service by client applications. A COM object may be implemented as a DLL (Dynamic Linked Library). The object in DLL can be used as an in-process server when it is loaded to the client's memory space by COM run-time library through SCM. A COM object can also be implemented in an executable file in which the COM object can be created only in the memory space of a running instance of that executable file. A COM object created in a running instance of an application is

called out-of-process server. In-process server can be accessed directly by calling from a function entry in its interfaces. Out-of-process server can only be accessed through stub\proxy codes, generated by an IDL compiler or the operating system.

Microsoft provided many COM components to facilitate accesses to a wide scope of system services, such as transaction management and security management. ADO is one of them. MTS (Microsoft Transaction Server) is another example, which provides COM objects and interfaces that expose the distributed transaction management service MS DTC (Microsoft Distributed Transaction Coordinator) to programmers.

The concept of "Apartment" was introduced to COM technology initially for concurrency control. "Apartment" is a boundary for COM object with a certain synchronization property. For example, all COM objects which invocations must be synchronized should be put to the same apartment. All COM objects which do not have to be synchronized should be put to another apartment. An apartment is created when the COM run-time library is initialized and can not be changed. This concept did not show up explicitly in CREEDA project, since all COM objects developed in this project were in-process servers. Concurrency control was performed by MS SQL server in current version of CREEDA.

4. DCOM

DCOM object is an out-of-process server on another machine. A client locates or creates DCOM objects through SCM (Service Control Management). At first, when the client submits a request to an object on another machine, the client's COM run-time routines contacts local SCM (SCM on the local machine), which then contacts targeted remote SCM (SCM on the another machine). The latter creates or locates a COM object and returns marshaled interface of the COM object to local SCM. They use ORPC protocol to communicate.

5. Advantages with COM

Like CORBA, COM technology allows software modules (COM objects) have an independent life-cycle management. Around COM objects are numerous system services provided by the operating system independent of any particular application. For example,

COM objects can be cached in a pool by system services and recycled among various applications. They can have their own security properties and concurrency control mechanisms.

A unique feature of COM is that it provides version service. Different version of COM objects can co-exist in a system and clients can choose which version to load.

Appendix B The typical scenarios of CREEDA

This document is given as a test report.

Open CREEDAInputEditor.exe

ODBC Logon/

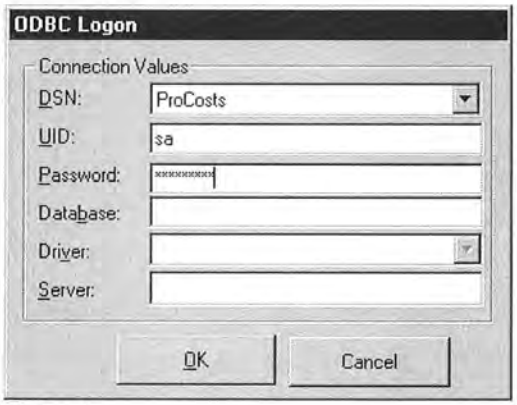

Figure 1
UID: sa
Password: ********
OK

CREDDA Input Editor
- Irrigation Methods
- Fertilizers
- Machinery Operations
- Pesticides
- Residue Management
- Crops
- City Database

Click Irrigation Methods

Figure 2

Name irrigation practice

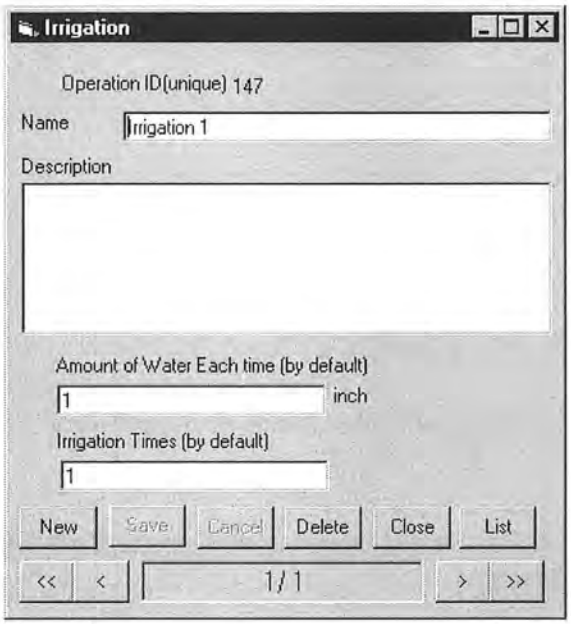

Figure 3

Description
Amount of water per application [x inches]
Number of applications [x applications]
Click [new]
Save or Cancel
Click List to see the different irrigation methods are available (browse)

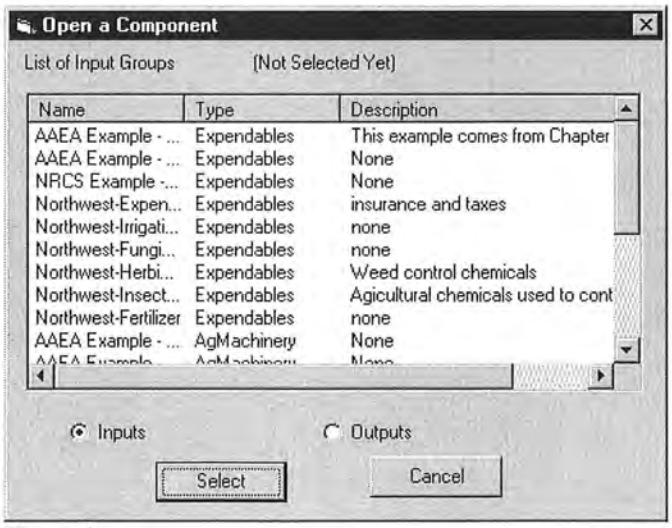

Figure 4

Close

Click Fertilizer

Repeat as above

Machinery

Name of machinery

Description

SCI disturbance parameters (I, M, L, S, A, and C)

Fraction of biomass Left on the field after this operation: [range is 0 to 1]

Residue Management

City Database

CREEDALinker.exe

Logon

Same as above

Select either [inputs or Output]

Then select from list of Input/Output Group from Open a Component window [Northwest Agmachinery Tractor-T]

ProCosts-CREEDA Linker

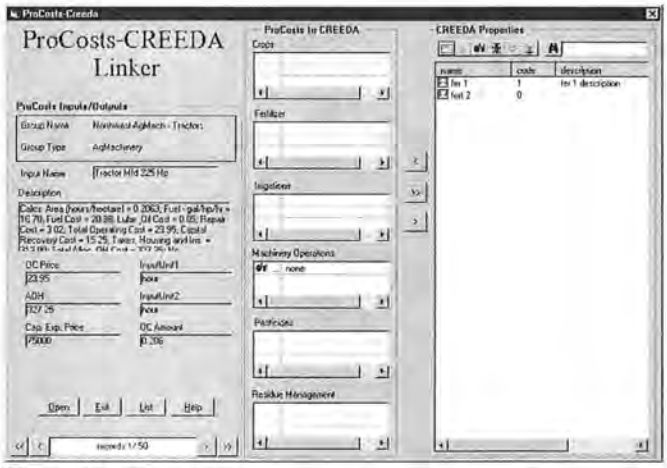

Figure 5

Left side shows `ProCosts` Inputs/Outputs

Right is list of CREEDA Properties

Center of window, list of the links between CREEDA properties `ProCosts` Inputs and Outputs. Use the one or as many of the six CREEDA properties (Fertilizers, Irrigation, Machinery, Pesticides, Residue Management, & Crops) to link CREEDA and ProCosts.

CREEDAMain.exe

Figure 6

Login/

User Name: demo
Password: demo
Click Login

CREEDA

Click Open Budget
Dialogue Box: List of Budgets

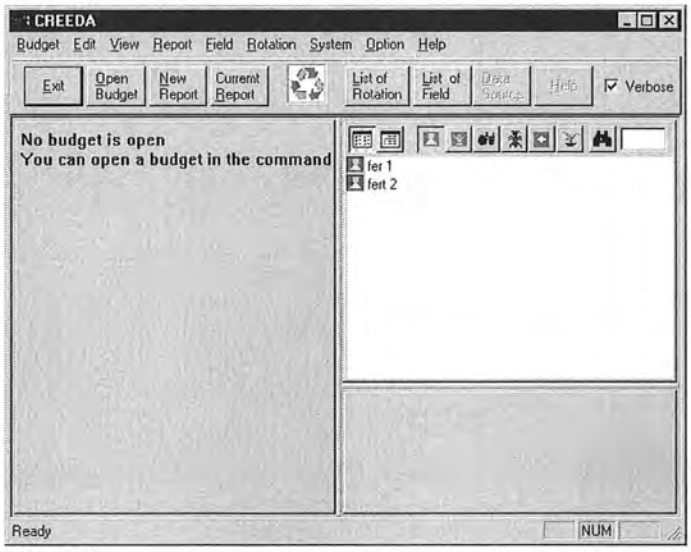

Figure 7

Select a Budget

Figure 8

Click List of Field

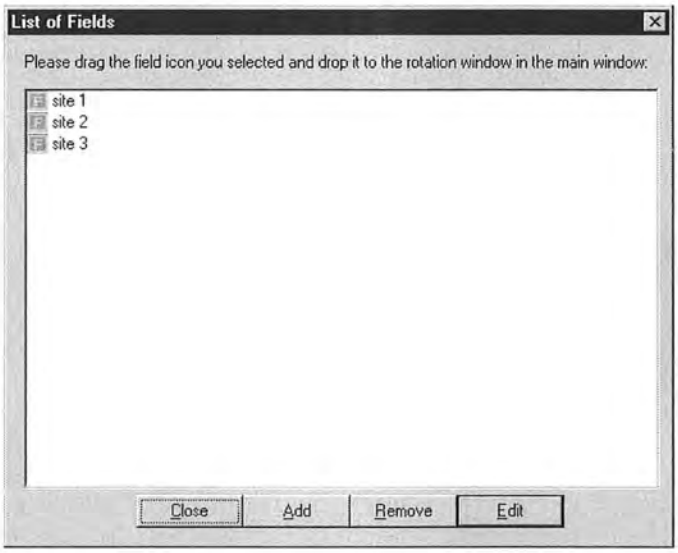

Figure 9
Click Add to create a new field

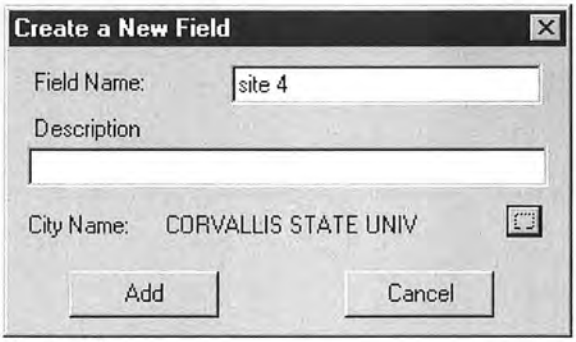

Figure 10

Field name: [field name]

Description:
City Name: click the button
Dialogue Box: List of Cities
Click on a city
Click Select
Click Add
Modify the soil data for the newly defined field/
Click Edit
Look at map for RUSLE R
.....
Click Save
Click Exit

Place cursor on desired Field, press down on left click button of mouse and drag and drop the Field on the CREEDA palette.

Close List of Fields window

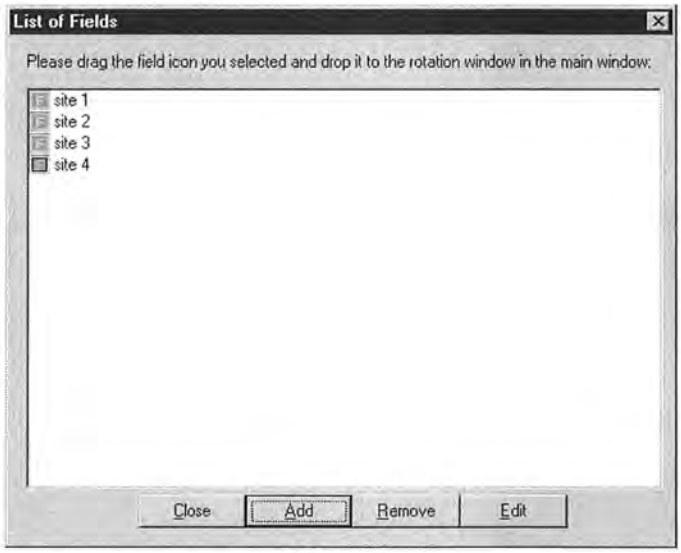

Figure 11

Click List of Rotations/select a rotation

Figure 12

Drag and drop rotation on the Field in the CREEDA palette.
Click the rotation

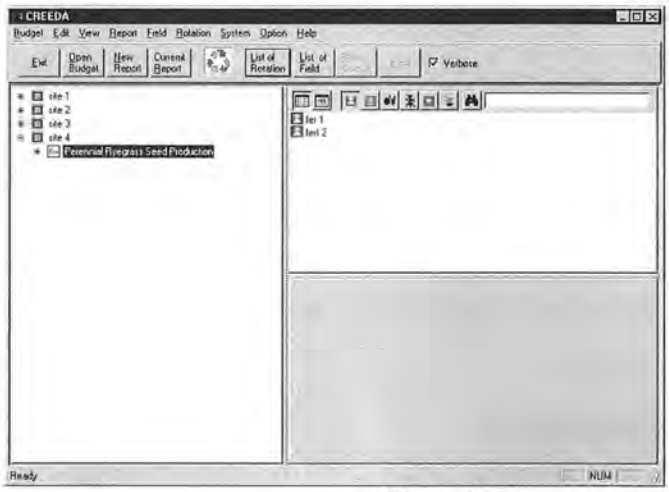

Figure 13
Click the Year in the rotation
Click on the operation

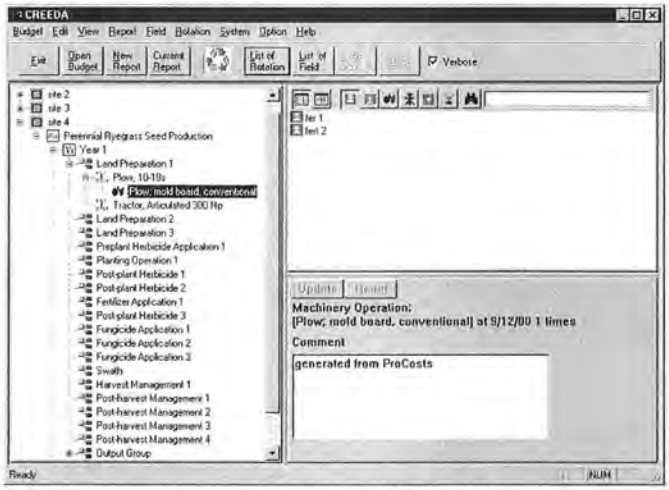

Figure 14

If the link is defined for the operation, there will be a CREEDA property attached to the operation. If not, you need to manually define the properties.

Choose one of the six CREEDA properties, and drag and drop to the ProCost input in the ProCost tree that is shown in CREEDA (harvest and residue operations should be in shown in a time sequence order because of the calculation of remaining residue dependent upon operation order).

Attach a Crop property to the ProCost output

Click the [Crop Button]

Find the crop in the list view window

Press on left mouse button and drag and drop Crop to ProCost output.

Click the crop in the tree,
Select the Planting Data
Click Update button
Click Residue Management property attached to the ProCost operation input.
Click [Calc]

Click Residue Management in the tree

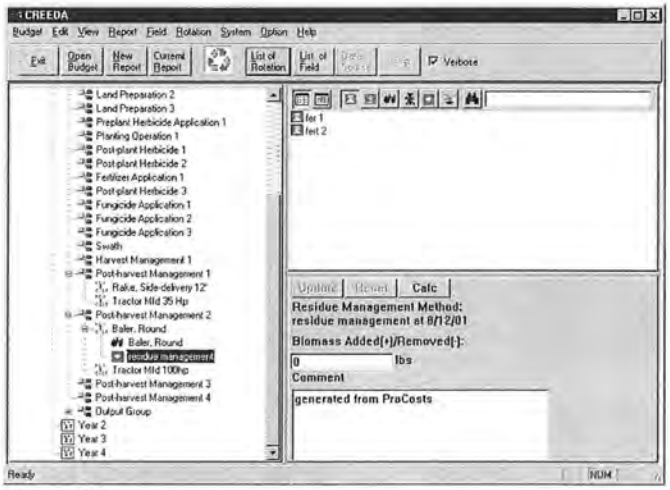

Figure 15

Then click [Calc]
Biomass Removed dialogue box shows the amount of residue remaining in the field

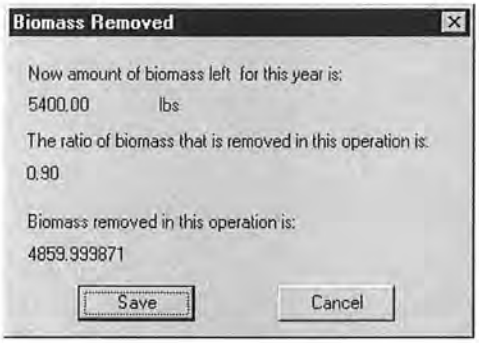

Figure 16

Click [save]

Click [New Report]
Gray fields indicate that there has been no linkage between CREEDA and ProCosts;
Green colored fields can be used to generate reports.
Choose from output selections to report CREEDA effects.

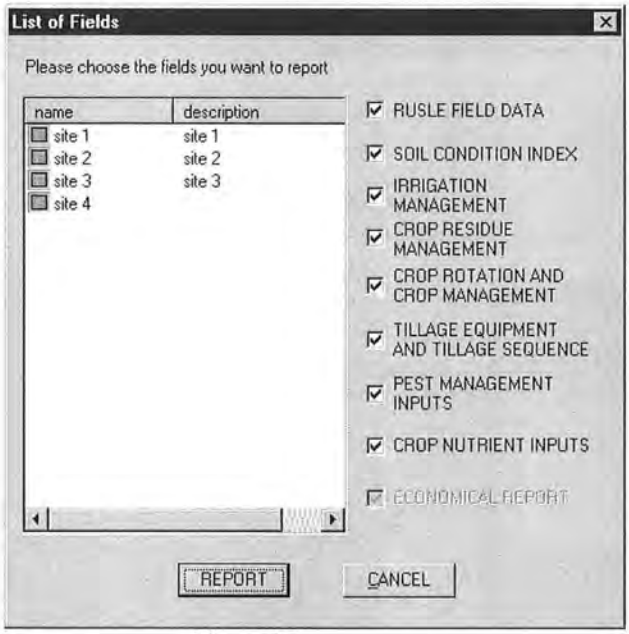

Figure 17

Click [Report]

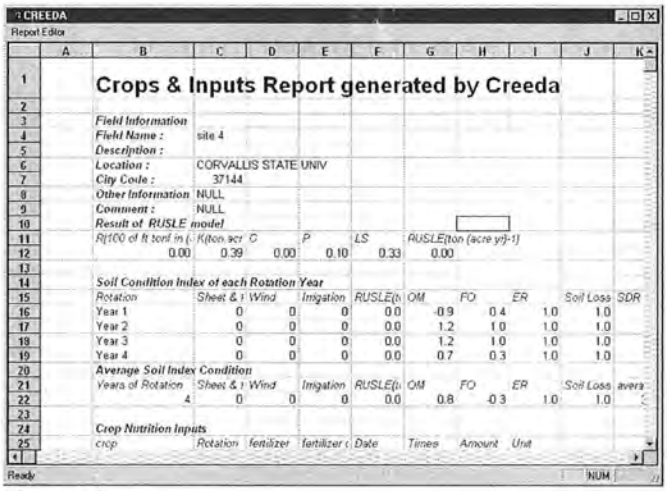

Figure 18

Menu/Report Editor/Save As

Dialogue box allows you to name the CREEDA output file in EXCEL format.

After you are finished:

Report Editor/Back to Main Window

Conservation Effects windows:

RUSLE dialogue windows

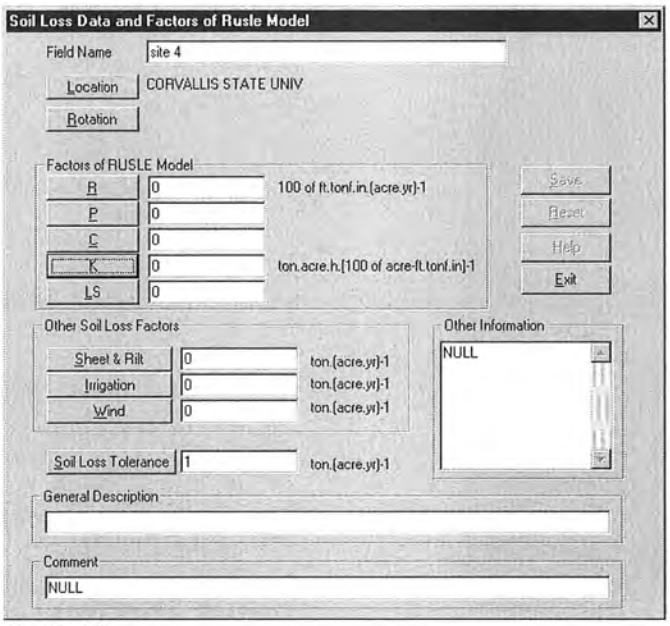

Figure 19

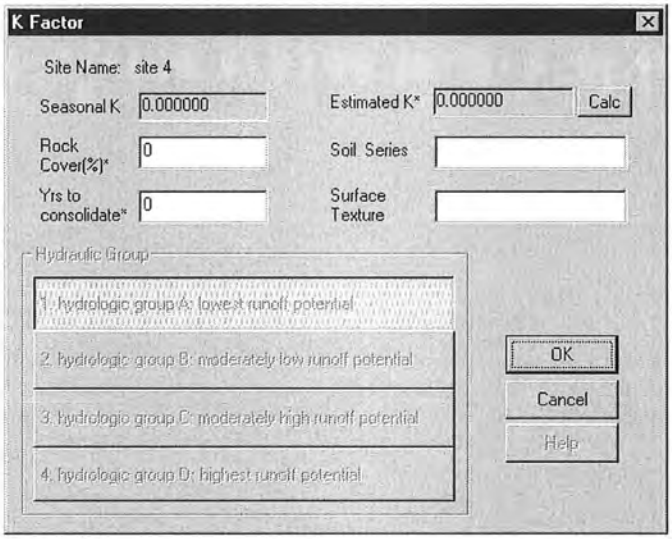

Figure 20

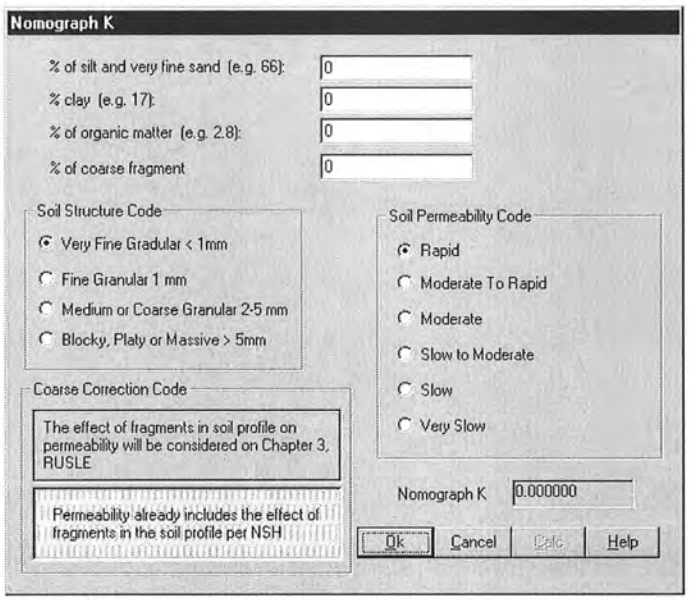

Figure 21

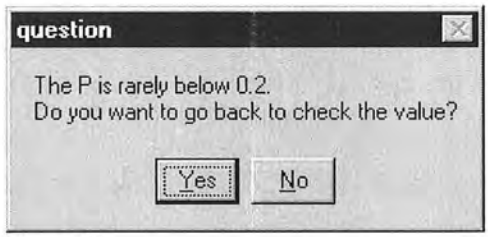

Figure 22