Design Document

Software Design Specification

Document Version: 2.0
Team: All Your Base
Date: Feb 22, 2005

Table of Contents

1.0 Introduction

1.1 Purpose of this document

The Software Design Specification (SDS) document will have two major releases:

1. Version 1 focuses on specifying a high-level view of the architecture of our system, and on the interaction between the user and the system.
2. Version 2 focuses on detailing a low-level view of each component of the software and how the components interact with each other.

This is Version 1, and so you will find only brief descriptions or "N/A" in areas that will be the main focus of Version 2.

This document's purpose is to provide a high-level design framework around which to build our project tracking system (a.k.a., ProjectTracker). It also provides a list of requirements against which to test the final project and determine whether we were able to successfully implement the system according to design.

1.2 Scope of the development project

Our ProjectTracker application is a server/client combination that will allow a user to track multiple projects from start to finish, keeping track of all the resources (human or material) necessary for each project's completion. This will include keeping track of time spent on the project, documentation steps, costs associated with the project, and reporting capabilities. Via the Java Runtime Environment, the ProjectTracker will be able to run on various platforms, including Unix, Linux, Windows and PocketPC—any computer that supports Java and JFC/Swing. When a network connection to the server is available, the user will be able to synchronize his PDA or PC with the server--updating new jobs and job information from the server to his device, and vice versa.

PDA Issues: Because of memory limitations, a PDA will only store project information that is very likely to be of interest to the PDA user. Also, PDAs have reduced screen size and limited input capability compared to PCs, so we will limit PDA standalone functionality to features that can be easily presented on a typical 240x320 PocketPC screen. These features will include the ability to add and modify projects, time logs, and project costs.

Synchronization: We will design and implement server software to serve as an interface between the PC or PocketPC and the Database. It will listen on open sockets for PCs or PDAs to connect and synchronize.

1.3 Definitions, acronyms, and abbreviations

- GUI - Graphical User Interface
- PC - Personal Computer (desktop or laptop)
- PDA - Personal Digital Assistant; a portable computing device.
- PocketPC - A Windows-based operating system
- ProjectTracker - The name of our Project Tracking Software
- Project Server - The computer which contains the central project database, and which serves project data to various clients

1.4 References

N/A

1.5 Overview of Document

The System architecture description section is the main focus of version 1 of this document. It provides an overview of the system's major components and architecture, as well as specifications on the interaction between the system and the user.

The Detailed description of components section will be the main focus of version 2 of this document. It will describe lower-level classes, components, and functions, as well as the interaction between these internal components.

In section 4, we detail the steps that we are taking to focus on code reuse in this software, and we explain our motives for doing so.

Section 5 lists the major decisions we had to make when designing our system, and why we made the choices we did.

The Pseudocode section will provide pseudocode in order to clarify the intended operation of certain components. This is beyond the scope of this version of the SDS.

Back to Table of Contents

2.0 System architecture description

2.1 Overview of modules / components

Our system is designed with extensibility and scalability in mind. We are taking great care in designing a framework which can be updated easily. Many of the anticipated changes to our system in future phases will only require adding new types of data and changing the user presentation code to make use of these new data. The framework we have designed will only require "plugging in" these new types of data without refactoring the logic that passes the data over the network, retrieves and updates the database, etc. There are five basic, logical components of the system: the Database Engine, the Class Library, the Server Procedure Proxy, the Server Application, and the Client Applications.

2.2 Structure and relationships

Refer to the appendix at section 7.1 for a graphical representation of the relationships between these objects.

- Database Engine
  - Existing open source software: Ingres
  - Hosts the backend database which is used for central data storage.
- Class Library
  - Java package which holds all Java classes (and "Java Beans") that both the server and client applications use to represent the data in memory.
  - Provides a uniform representation on both sides of the network connection.
  - Most updates and additions will require only updating this library.
    - Updates to the package *must* be backwards compatible.
  - Client applications can use these objects to represent data, separating it completely from the server-side representation.
- Server Procedure Proxy
  - Java class which provides local methods for invoking remote methods on the server application.
  - Will be contained within the client-side class library.
  - Takes care of serializing the objects passed to and from the client application so they can be sent over the network.
  - Hides the details of how communication with the server is achieved.
  - If a change in communication strategy is desired, this would only require updating this class.
  - Simplifies client/server communication for the client.
- Server Application
  - Implemented in Java
  - Provides methods and procedures that can be invoked remotely by a client application via the server procedure proxy.
    - Retrieve project data.
    - Update project data.
    - Generate reports.
  - Central process which can make all decisions that arise due to the distributed nature of this application.
    - For instance, when a client wishes to update a project, there may be conflicts that need to be resolved if another client has updated the same project.
    - The server can coordinate conflict resolution with the client application (which can ask the user

questions, if necessary).
    - Uses the class library package defined above (must have the same or newer library as the client application).
- Client Applications
    - Implemented in Java.
    - Contains all presentation logic.
    - Interacts exclusively with the user.
    - Uses the class library package defined above.
    - Communicates with the server application through the server procedure proxy class contained in the class library.
- Reporting Application
    - Uses Jasper Reports to present html-based, web-accessible reports based on the data found in the database.

2.3 User interface issues

The user interface of the Project Tracker will be divided into three main sections: The "Employees and Reports" application, the "PC Client" application, and the "PDA Client" Application. These tools will allow the various users to accomplish the management of the software, and the management of the rest of their projects. The personas described in the SRS represent the types of people who will make use of the Project Tracker - and the purpose of this section is to describe how the user interfaces allow those people to do their tasks.

2.3.1 The Employees and Reports Application

This application will reside on the Project Server, and will be used mostly by the Administrator (a.k.a. "Steve Techie"). It will have the following basic layout:

Login Screen - This first screen will allow *Steve Techie* to protect access to the rest of the application.

Employee Management Screen - This screen allows *Steve Techie* to enter new employees, who can then be assigned to projects.

Reports Screen - This screen allows *Steve Techie* to generate reports on project and employee activity, which will be of great interest to the owner ("Mr. Manager"), and perhaps the customer ("Bob PaysUsWell"). The reports types are:

- Weekly Project Overview: gives a brief summary of each employee's actions with regard to a project over the week.
- Weekly Employee Detail: gives a summary of how each employee's time was split amongst various projects

2.3.2 The PC Client Application

This application will reside on the computers of all the users, and will be the main interface to the database for all but *Steve Techie*. It will have the following layout:

Login Screen - This lets any user log in to the Project Server.

Projects Overview Screen - After login, the application will present this screen and will list all of the projects that the client program knows are assigned to the current user. This may be incomplete or out of date if the user has not synchronized with the Project Server recently. The client will automatically attempt to synchronize after login. During this synchronization a potential conflict can occur if a project has been altered both server side and client side during the time the client was offline. The user will be asked to resolve such conflicts before being allowed to go onto other tasks. This screen includes:

- A status display showing whether the application is online with the Project Server or not, and the last date and time when a successful synchronization occurred.
- A tabbed list of the projects: The tabs are "Unfinished Projects", "Finished Projects", and "All Projects". Any of the lists may be sorted by priority. A user may, by double clicking on a project in the list, go to the Project Maintenance Screen for that project.
- An "Add Project" button, which will allow any user to create a new project. This will go to a blank Project Maintenance Screen for that project.
- A "Time Entry" button, which will allow a user to report his/her work hours.

Project Maintenance Screen - This screen contains information about a particular project, and buttons to investigate further details. It has three main tabs.

The first tab, "Summary" contains general data in the top two thirds of the screen--for a full list of those fields see the mock up of the screen in section 7.2 of the appendix. The bottom portion of the screen lets the user look at lists of Action Items, Costs, and Notes.

Action Items are steps to be taken in the completion of the project. These can be marked as finished when the user completes them. When viewing the action items lists, the user may click on an action item to view further details, or to convert the action item to a project. In this latter case, the newly created project will remain in the action items list but will have a modified look, and the details screen for that action item will be another Project Maintenance Screen instead of a simpler action item detail. This feature may be used by the SuperOwner ("Veep McBleep"), in order to delegate further tasks to his subordinates.

The second tab of the Project Maintenance Screen, "Contact Information", will show contact information for key people related to the project as further proof against communication breakdown.

The third tab, "Extended Documentation" will contain a detailed description of the project, and links to related resources.

Time Entry Screen - This will be a very simple screen which allows the user to specify a project, a start time, and an end time. When the values are entered, the user clicks the "Check in" button, and the time report is added to the project. This will allow the average user ("Joe Schmoe"), to account for his time. These time reports will be compiled into reports for *Mr. Manager* using the Employees and Reports Application. Also, a tab will allow a user to account for costs accrued during the course of their duties.

2.3.3 The PDA Client Application

The PDA client will be designed as an alternate interface from the PC Client, for easier tracking on the go. It will contain most of the basic functionality of the PC Client.

The issues related to the PDA client are ones of scale. Some of the screens will need to be modified for easy viewing on the PDA screen. Here are the screens and how they might be affected:

Login Screen - Nearly identical to the PC Client Screen

Projects Overview Screen - Nearly identical to the PC Client Screen

Project Maintenance Screen - In particular the summary view will need a major redesign. Data groups will be replaced by buttons which will bring up more detailed screens for those data groups. The groups will be "Project Identification", "Schedule and Priority", "Client and Billing", and "Project Items". This last button will replace the tabbed view of Action Items, Costs, and Notes.

Time Entry Screen - Nearly identical to the PC Client Screen

Back to Table of Contents

3.0 Detailed description of components

3.1 Component Overview

The following bulleted outline provides a basic overview of the purpose and architecture of our system's major components. The tables in the remainder of section 3 will provide more details on each component:

- 3.2 **Database**: We will be using Ingres as our database software. It will contain tables to represent various project data.
- 3.3 **Class Library**: The Class Library will be a package of classes common to the Server Software and the Client Software. The server software will use these class definitions to know how to store the objects in the database. The client software will use the class definitions in order to ask for and display the correct information about each class type.
    - Project class implements SubTask
    - ProjectNote class
    - SubTask interface

    - ActionItem class implements SubTask
    - TimeDetail class
    - ProjectCost class
    - Entity class
    - Employee class extends Entity
    - Supervisor class extends Employee
- 3.4 **Reporting Application**: A JasperReports-based web application which will use the information in the database to produce useful reports for users.
- 3.5 **Server Software**: Our server software will directly manipulate the contents of the database, based on commands from the Server Procedure Proxy.
- 3.6 **Server Procedure Proxy**: The Server Procedure Proxy will provide an interface for the clients to request information or submit updates to the server.
- **Client Software**: The Client Software will reside on either a desktop computer or a PocketPC. Its purpose is to present data to the user as requested, and provide an interface so that the user can easily update project information. It will handle conflict resolution automatically when possible, and otherwise allow the user to choose which version of an item to keep.
    - 3.7 **PocketPC Client**: The PocketPC client will allow the user to view basic information about and make simple changes to data regarding projects assigned to them.
    - 3.8 **Desktop Client**: With significantly more computing power, screen size, and storage space than the PocketPC, the desktop client will provide more significant functionality in allowing the user to view and edit project information.

Back to Table of Contents

3.2 Database

Identification	Database Software
Type	Module
Purpose	Provides means of data management and storage for the server software.
Function	Takes SQL QUERY and UPDATE commands from the server software, and stores the data according to those commands.
	The database will store data for each type of item in the class library in tables. • project ○ project_id* ○ project_no ○ employee_id ○ status_cd ○ user_id ○ system_id ○ application ○ description ○ project_purpose ○ project_tpe ○ consultant_priority ○ system_priority ○ release_version ○ secs_total ○ project_use ○ proj_secs ○ proj_finish_dt ○ completed_dt ○ bill_yn ○ master_project_id ○ follows_project_id ○ paying_entity_id ○ est_completion_dt ○ actual_hrs ○ start_dt ○ actual_cost ○ url ○ project_phase ○ requested_by ○ contact_name

Subordinates

  - s_project_id**
  - s_db_id**
  - suser**
  - m_db_id**
  - muser**
  - m_dt**
  - t_db_id**
  - tuser**
  - t_dt**
- project_note
  - project_note_id *
  - project_id
  - note_type
  - note
- proj_item
  - proj_item_id *
  - project_id
  - line_item
  - priority
  - description
  - status_cd
- project_cost
  - project_cost_id*
  - project_id
  - line_item
  - cost_amt
  - bill_amt
  - stock_no
  - employee_id
  - cost_type
  - description
- timedetail
  - timedetail_id *
  - timedetail_no
  - employee_id
  - project_id
  - user_id
  - paying_entity_id
  - check_in_dt
  - check_out_dt
  - seconds
  - comments
  - bill_yn
  - status_cd
  - entry_dt
  - consultant_priority
  - client_priority
  - system_priority
  - post_no
  - post_st
  - post_dt
- entity
  - entity_id *
  - c_id
  - ccode
  - name
  - location_1
  - location_2
  - street_box
  - city
  - state
  - zipcode
  - province
  - country
  - geocode
  - birth_dt
  - status_cd

    - allowance_hrs
- address
    - address_id *
    - entity_id
    - address_type
    - name
    - address_name
    - location_1
    - location_2
    - street_box
    - city
    - state
    - zip
    - province
    - country
    - geocode
- employee is an entity
    - employee_id *
    - employee_no
    - employee_name
    - employee_entity_id
    - def_client_entity_id
    - employee_uid
    - supervisor_yn
    - hour_wage
    - salary
    - last_in_dt
    - timecard_mod_yn
    - status_cd
    - scheduled_week_secs
- supervisor is an employee
    - supervisor_id *
    - supervisor_employee_id
    - employee_id
    - department
    - supervisor_entity_id
    - employee_entity_id
    - status_cd
- system
    - system_id *
    - system_cd
    - manager_entity_id
    - description
    - visible_yn
- contact is an entity
    - contact_id
    - entity_id
    - contact_name
    - contact_type
    - contact_info

* table primary key

** every table has similar fields for distributed database.

Dependencies	An administrative user must perform database setup functions, adding and modifying the structure of tables so that the server software can store the data as appropriate.
Interfaces	Apart from administrative setup, all modification of the database items will be performed via JDBC by the server application. The Reporting application will read the database through JasperReports to generate reports.
	The database we are using is Ingres 3.0, which will run on a Windows based computer. All tables will be created by an administrator (us). We will use the Ingres interface to manually enter initial data in the following tables: Entity, Address, Employee, Supervisor, System and Contact. Once these are entered, we will use the data held in these fields as a baseline for our user interface. If time permits, we may create a GUI which will allow entry of these parts through our application. Our project tracker will connect to the database using JDBC connectivity. Once the server

Resources	Our project tracker will connect to the database using JDBC connectivity. Once the server gets a request for a certain data type from a client it will look up the table that is associated with the requested object, then parse the restriction clause and create a valid SQL statement. Once it has generated a valid SQL statement, the server will connect to the database, execute the query and get a java ResultSet back. At this point, it will convert the ResultSet to the Object that was requested by the client and send it back to the client via Sockets and ObjectStreams over the network. Calls to the database should not take longer than (but should be much less than) 30 seconds and transfers of the each Object over the network should not take longer than 1 minute. Conversion of a ResultSet to a Java Object should take less than 1 second. Data retrievals in general should be less than 2MB. A client may also send an updated object to the server, in this case it will convert to a Set and then do an UPDATE to the database. This is a basic database call so it should not take longer than 1 second from start to finish.
Processing	**SQL QUERY** will come from the client in the form of *Object*.getAll(String("condition1 = condition2 and condition3 = condition 4")) For example: Project.getAll("employee_id = 120 and status_cd = "CURR") We will determine the Project class is associated to the project table in the database and it will generate the following SQL statement: SELECT * FROM project WHERE employee_id = 120 AND status_cd = "CURR" It will send this to the database and will get a java ResultSet. A ResultSet is easily converted to our object because the column names in the database will exactly match the fields in our class. For example, in the database we have project_no and employee_id columns, in our class Project we have fields project_no and employee_id. **SQL UPDATE** will come from the client in the form Object.update(); For example: Project.update() We know for Project that the primary key is project_id so we will create an SQL statement saying: UPDATE project set employee_id = 2, user_id = 2....(all fields) WHERE project_id = 25; This statement will be sent to the database which will either update or trigger an exception.
Data	The data in the database will be filled by the server except for the cases stated above. We will use valid SQL statements called within Java through JDBC to execute these commands. In every case we will either get a ResultSet (for queries), a working return code or a SQLException if the command does not work for any reason. We will handle these exceptions according to their type.

Back to Table of Contents

3.3 Class Library

Identification	ProjectTracker.DataObjects
Type	Package
Purpose	Provides data structures for the server and clients.
Function	This is simply a package of classes. Each class has a set of accessor and mutator methods as well as logic for retrieving, inserting, updating, and deleting data from the database.
Subordinates	The Class Library includes the following classes, all of which implement the general DataObject interface presented in the *Interfaces* sub-section: • SubTask interface • Project class implements SubTask • ActionItem class implements SubTask • ProjectNote class • TimeDetail class

	• TimeDetail class • ProjectCost class • Entity class • Employee class extends Entity • Superviser class extends Employee
Dependencies	The Server software and the Client software use the class library to store and retrieve data. The Server software relies on the database logic being present in every DataObject in order to load and store data from the database.

Interfaces	All classes in the class library will implement the abstract class (or interface) DataObject, which has the following interface:
	
	DataObject implements Serializable
	{
	
	private boolean dirty;
	private int id;
	private DateTime downloadDate;
	private DateTime uploadDate;
	
	public boolean isDirty ();
	public DateTime getDownloadDate();
	public DateTime getUploadDate();
	
	private static HashMap fieldToColumn;
	
	public static List getAll (Connection conn, String filter);
	public static void put (Connection conn, DataObject obj);
	public static void delete (Connection conn, int id);
	protected static String makeSqlWhereClause (String filter);
	
	}
	
	All DataObjects are Serializable so that they can easily be marshalled over the network by the server as well as saved to disk on the client. In addition, each DataObject will have its set of JavaBean-like accessor and mutator methods. The clients will use these methods for presenting and manipulating DataObjects; they will also use the isDirty, getDownloadDate, and getUploadDate methods to perform synchronization with the server. The server will use the static methods getAll, put, and delete to load and store data for this DataObject to and from the database. The makeSqlWhereClause method and the fieldToColumn field will be used in the getAll, put, and delete methods to parse filter clauses passed from the server application and dynamically create the corresponding SQL WHERE clause.

Resources	The specific resources required for the class library are JDBC and Java's serialization mechanism.
Processing	A detailed description of the DataObject's layout is presented in the *Interfaces* sub-section.
Data	A detailed description of the DataObject's layout is presented in the *Interfaces* sub-section.

Back to Table of Contents

3.3.1 SubTask

Identification	ProjectTracker.DataObjects.SubTask
Type	Interface
Purpose	Supertype for action items and sub-projects. Both are considered "sub-tasks" of a project.
Function	Simply a Java interface that provides a contract for action items and sub-projects. Allows the GUI to categorize both as sub-tasks of a project.
Subordinates	ActionItem, Project.
Dependencies	The GUI will make use of this interface while interacting with action items and sub-projects.

Interfaces	Classes that implement SubTask also implement DataObject. In addition to the usual DataObject interface, classes that implement SubTask will have methods and fields for getting and setting the name, responsible person, and status of the sub-task.
Resources	None.
Processing	None.
Data	N/A

Back to Table of Contents

3.3.2 Project

Identification	ProjectTracker.DataObjects.Project
Type	Class
Purpose	Represents a project and encapsulates all necessary data for a project.
Function	A project has the following data items. These are the column names as defined in the database schema, but the object may have different field names. Also, each field will be accessed and mutated via JavaBean-like get and set methods. • project_id • project_no • employee_id • status_cd • user_id • system_id • application • description • project_purpose • project_tpe • consultant_priority • system_priority • release_version • secs_total • project_use • proj_secs • proj_finish_dt • completed_dt • bill_yn • master_project_id • follows_project_id • paying_entity_id • est_completion_dt • actual_hrs • start_dt • actual_cost • url • project_phase • requested_by • contact_name • s_project_id • s_db_id • suser • m_db_id • muser • m_dt • t_db_id • tuser • t_dt
Subordinates	Defined in the *Function* sub-section.
Dependencies	Being a DataObject and SubTask, this class relies on those interface definitions.
Interfaces	Defined in the *Function* sub-section.

Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

[Back to Table of Contents]

3.3.3 ActionItem

Identification	ProjectTracker.DataObjects.ActionItem
Type	Class
Purpose	Represents a project and encapsulates all necessary data for an action item.
Function	An action item has the following data items. These are the column names as defined in the database schema, but the object may have different field names. Also, each field will be accessed and mutated via JavaBean-like get and set methods. • proj_item_id • project_id • line_item • priority • description • status_cd
Subordinates	Defined in the *Function* sub-section.
Dependencies	Being a DataObject and SubTask, this class relies on those interface definitions.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

[Back to Table of Contents]

3.3.4 ProjectNote

Identification	ProjectTracker.DataObjects.ProjectNote
Type	Class
Purpose	Represents a project and encapsulates all necessary data for a project note.
Function	A project note has the following data items. These are the column names as defined in the database schema, but the object may have different field names. Also, each field will be accessed and mutated via JavaBean-like get and set methods. • project_note_id • project_id • note_type • note
Subordinates	Defined in the *Function* sub-section.
Dependencies	Being a DataObject, this class relies on that interface definition.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

[Back to Table of Contents]

3.3.5 TimeDetail

Identification	ProjectTracker.DataObjects.TimeDetail

Identification	ProjectTracker.DataObjects.TimeDetail
Type	Class
Purpose	Represents a project and encapsulates all necessary data for a time detail.
Function	A time detail has the following data items. These are the column names as defined in the database schema, but the object may have different field names. Also, each field will be accessed and mutated via JavaBean-like get and set methods. • timedetail_id • timedetail_no • employee_id • project_id • user_id • paying_entity_id • check_in_dt • check_out_dt • seconds • comments • bill_yn • status_cd • entry_dt • consultant_priority • client_priority • system_priority • post_no • post_st • post_dt
Subordinates	Defined in the *Function* sub-section.
Dependencies	Being a DataObject, this class relies on that interface definition.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Back to Table of Contents

3.3.6 ProjectCost

Identification	ProjectTracker.DataObjects.ProjectCost
Type	Class
Purpose	Represents a project and encapsulates all necessary data for a project cost.
Function	A project cost has the following data items. These are the column names as defined in the database schema, but the object may have different field names. Also, each field will be accessed and mutated via JavaBean-like get and set methods. • project_cost_id • project_id • line_item • cost_amt • bill_amt • stock_no • employee_id • cost_type • description
Subordinates	Defined in the *Function* sub-section.
Dependencies	Being a DataObject, this class relies on that interface definition.
Interfaces	Defined in the *Function* sub-section.
Resources	None.

Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Back to Table of Contents

3.3.7 Entity

Identification	ProjectTracker.DataObjects.Entity
Type	Class
Purpose	Represents a project and encapsulates all necessary data for an entity.
Function	An entity has the following data items. These are the column names as defined in the database schema, but the object may have different field names. Also, each field will be accessed and mutated via JavaBean-like get and set methods. • entity_id • c_id • ccode • name • location_1 • location_2 • street_box • city • state • zipcode • province • country • geocode • birth_dt • status_cd • allowance_hrs
Subordinates	Defined in the *Function* sub-section.
Dependencies	As a DataObject, this class relies on that interface definition.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Back to Table of Contents

3.3.8 Employee

Identification	ProjectTracker.DataObjects.Employee
Type	Class
Purpose	Represents a project and encapsulates all necessary data for an employee.
Function	An employee has the following data items. These are the column names as defined in the database schema, but the object may have different field names. Also, each field will be accessed and mutated via JavaBean-like get and set methods. • employee_id • employee_no • employee_name • employee_entity_id • def_client_entity_id • employee_uid • supervisor_yn • hour_wage • salary • last_in_dt

	• timecard_mod_yn • status_cd • scheduled_week_secs
Subordinates	Defined in the *Function* sub-section.
Dependencies	As a DataObject, this class relies on that interface definition. As an instance of Entity, this class relies on that class definition.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Back to Table of Contents

3.3.9 Supervisor

Identification	ProjectTracker.DataObjects.Supervisor
Type	Class
Purpose	Represents a project and encapsulates all necessary data for a supervisor.
Function	A supervisor has the following data items. These are the column names as defined in the database schema, but the object may have different field names. Also, each field will be accessed and mutated via JavaBean-like get and set methods. • supervisor_id • supervisor_employee_id • employee_id • department • supervisor_entity_id • employee_entity_id • status_cd
Subordinates	Defined in the *Function* sub-section.
Dependencies	As a DataObject, this class relies on that interface definition. As an instance of Employee, this class relies on that class definition.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Back to Table of Contents

3.4 Reporting Application

Identification	Report Package
Type	Package
Purpose	The Report Package will reside on the server and generate reports requested by the client software.
Function	The Server will get a request from the client in the form of Reports.run*ReportName*(Arguments). The server will keep the incoming socket open so the generated report can be sent back. After a request is made, the ReportPackage will call our JasperInterface, which will be created so a programmer doesn't have to be familiar with the JasperReports package in order to write reports. The run*ReportName* call will know already know the SQL query and have the definition of that specific report contained within. This will then call JasperInterface.generateReportPDF*(ReportName*, *outputfile)* Once report is generated, the server will send the resulting output file back to the client over the

	same connection that the request was made. The report output will be stored on the server in a folder called reports. If a report is needed twice by the same client it can use the getFile(filepath) command instead of having to generate the same report twice.
Subordinates	This will take a JasperReports report specification, and will access the data to fill that report.
Dependencies	This reporting application is just a package of pre-defined reports that the server can call once a request is made by a client computer. The report specifications must be hand written with 3rd part software (i.e. JasperAssistant) Once the report specification is created, that report is generated by using JasperReports interface classes. A report should be generated and sent over the network in less than 5 minutes time but should be around 30-60 seconds from request to transfer.
Interfaces	• Report requested from client: Reports.run*ReportName*(Arguments), • Server will respond to client a keepalive signal as long as the connection is kept and the report is still being generated. If the report has a problem or cannot run, the Exception that is thrown will be passed to the server and it will send an error code back to the client. • Server creating report with our defined interface. JasperInterface.generateReportPDF*(ReportName*, *outputfile)*
Resources	Our application will run on a TCP/IP network and use Java predefined sockets. The Reports Package requires that our server is running. It will also communicate with a Ingres 3.0 Database which will be running and accepting JDBC connections. Ingres 3.0 must have the users who will login to our client application to have their name and password setup the same as in their client application. Ingres 3.0 may not need to be running on the same machine the server is running on. But it must be accessible over the network. To run a report and transfer it to the client, it should take less than 5 minutes but should be more like 30-60 seconds if good network conditions permit. The file sent will be of the type requested by the client.
Processing	Reports.run*ReportName*(Arguments) 1. Parse argument list that may include a date-range, users (or clients) to run the report on, depending on what type of report is being requested. Make Report Call to interface class: JasperInterface.generateReportPDF*(ReportName*, Args, *outputfile)* 1. Take arguments stated above and pack them into a HashMap. Make the above call to Jasper Assistant sending it also a list of the reports arguments. 2. If report is generated: Send File back over the same socket, if report didn't work. Return a proper error code back to the client. 3. Close socket. 4. If client knows about a certain report it can use the getFile(filepath) which is a part of server's implementation.
Data	Arguments will be contained in a HashMap, which can be sent to JasperReports and retrieved from report specification. The report specification file will be held in a folder on a /ReportSource file on the server. All other data is held in the Database or contained in other classes or methods.

Back to Table of Contents

3.5 Server Software

Identification	ProjectTracker.Server
Type	Subprogram, package
Purpose	The server program will run on the same machine as (or at least have access to) the database engine. It is the program that the client applications connect to and communicate with to retrieve and synchronize project data and to run reports.

Function	The server continuously listens on a known port for client connections. It creates a new thread to handle each connection.
Subordinates	The server will rely on the class library outlined above.
Dependencies	The server depends on the database engine. It relies on the implementation of the DataObject classes in the class library. The client applications rely on the server to retrieve and synchronize project data and to run reports.

Interfaces	The server will listen on a predefined TCP/IP port. It will coordinate with the Server Procedure Proxy (which sits on the client side) and use Java's ObjectStream along with the serializable DataObjects in the class library to pass information across the network. There are two basic methods that the server will expose:
	
	List getAll (Class type, String filter);
	
	void put (DataObject obj, DateTime ifOlderThan);
	
	The first method will retrieve from the database all DataObjects of the given type and with the given filter string applied. A filter string has the exact same syntax and semantics as a SQL WHERE clause, except the object's field names are used instead of column names. The second method inserts the given DataObject into the database on the condition that the object which is currently stored in the database is older than the ifOlderThan parameter.

Resources	As previously mentioned, the server will use the TCP/IP network protocol, relying on Java's Socket and ServerSocket classes to do this. It also relies on Java's ObjectStream class and Java's serialization mechanism.
Processing	The exposed methods are described in the *Interfaces* sub-section.
Data	The server uses the Ingres database engine to store data, and is ignorant to the specific logic utilized to load, store, and manipulate the data. The server acts only as a communication device between the clients and the database.

Back to Table of Contents

3.6 Server Procedure Proxy

Identification	ProjectTracker.ServerProcedureProxy
Type	Package, class
Purpose	Manages communication with the server on the client side. Hides all the network specifics from the client code. Exposes a simple API to call the methods that the server exposes.
Function	The server procedure proxy class implements the communication protocol with the server. It knows how to connect to and communicate with the server. It marshals data over the network using Java's serialization mechanism.
Subordinates	The server procedure proxy will rely on the class library.
Dependencies	Client applications will use this class to retrieve and synchronize project data with the server. The server will rely on the clients utilizing this class for this purpose, since the server and the server procedure proxy agree in terms of the communication protocol.

Interfaces	The server procedure proxy simply exposes the methods offered by the server to the client. These are:
	
	List getAll (Class type, String filter);
	
	void put (DataObject obj, DateTime ifOlderThan);
	
	The first method will retrieve from the database all DataObjects of the given type and with the given filter string applied. A filter string has the exact same syntax and semantics as a SQL WHERE clause, except the object's field names are used instead of column names. The second method inserts the given DataObject into the database on the condition that the object which is currently stored in the database is older than the ifOlderThan parameter.

Resources	As mentioned, the server procedure proxy will use the TCP/IP network protocol, relying on Java's Socket and ServerSocket classes to do this. It also relies on Java's ObjectStream

	class and Java's serialization mechanism.
Processing	The exposed methods are described in the *Interfaces* sub-section.
Data	The server procedure proxy will use the class library to encapsulate data. These will be passed over the network to and from the server. The server is free to use whatever mechanism it chooses to store the data on its side.

Back to Table of Contents

3.7 PocketPC Client

Identification	PocketPC Client
Type	User Interface
Purpose	Provide a user access to project data stored locally or on a database.
Function	• Logs a user into the system, and attempts to connect to a specified project server using the Server Procedure Proxy. • When possible, downloads any new or modified projects from the specified server, and uploads any changes that have been made locally since the last successful server connection. In some cases, there may be conflicts between local and remote project data, and the PocketPC client will provide a means for conflict resolution through user interaction. • Allows the user to see a list of their projects, which consists of those downloaded and those created by the user locally. Allows the user to create new projects. • Allows the user to inspect and change a specified project's data. • Allows a user to report times and costs associated with a project
Subordinates	To see the general structure of the the PocketPC Client, see the flow diagram in 7.1.2. 1) The Login Screen - provides a level of security for the project data • Inputs ◦ User Name ◦ Password ◦ Server Settings • Outputs ◦ Status of the login attempt ◦ Presents the next form based on connection status and server authentication 2) The Projects Overview Screen - Shows the user projects which are assigned to him or her and provides access to other screens. • Inputs ◦ The new task that a user wants to do. The choices are Time/Cost Entry, New Project, and Modify Project. ◦ In the case of Modify Project, a highlighted entry in the Projects List is an input. • Outputs ◦ Displays the Project Maintenance Screen if the user chooses either New Project or Modify Project ◦ Displays the Time/Cost Entry Screen if the user chooses Time/Cost Entry 3) The Project Maintenance Screen - Shows details related to the project and provides means for modifying them. • Inputs ◦ Any data related to the selected project ◦ Ability to request the creation of new Action Items, Costs, or notes ◦ Ability to request the saving of modified items • Outputs ◦ All data related to the selected project ◦ The Action Items, Costs, and Notes which have been created for this project 4) The Time/Cost Entry Screen - Allows the user to report on their activities and costs related to a project • Inputs

	- Start Time - End Time - Project - Cost - Comment • Outputs - None
Dependencies	This component will interact closely with the Server Procedure Proxy for the retrieval of project data. The gui will create and pass around an instance of the Server Procedure Proxy class and use it to get and put data on the server. Such server calls will be synchronous, so there will be no timing issues for the PocketPC client to consider. The PocketPC Client will use various data classes found in the Class Library. Data from the Server Procedure Proxy will be packaged into one of these classes, and the PocketPC Client can then access that data through the public methods of the appropriate class. These classes also provide the PocketPC Client with a handy way to save the data locally (as do all java classes).
Interfaces	The PocketPC Client uses Swing classes for interaction with the user. The screen formats for these windows are shown in section 7.2. Uses the Server Procedure Proxy for interaction with a server. For more information on the Server Procedure Proxy see section 3.6.
Resources	The PocketPC Client will not have any significantly CPU- or memory-intensive tasks--however, normal overhead associated with Swing components is to be expected in both cases. Much of the project data to be queried will be accessed on a remote database.
Processing	Some pseudocode related to the application can be found here.
Data	A project list kept on the disk as a serialized List of Project objects, loaded into memory in the same format. Such a list will, of course, initially be empty. Associated project data kept on the disk as a serialized List of Objects, of any of several formats, loaded into memory in the same format. Initially empty. A list of user names and passwords encrypted on disk using javax.crypto.

Back to Table of Contents

3.8 Desktop Client

Identification	Desktop Client
Type	User Interface
Purpose	Provide the user access to project data stored locally or on a database.
Function	• Logs the user into the system, and attempts to connect to a specified project server using the Server Procedure Proxy. • When possible, downloads any new or modified projects from the specified server, and uploads any changes that have been made locally since the last successful server connection. In some cases, there may be conflicts between local and remote project data, and the Desktop client will provide a means for conflict resolution through user interaction. • Allows the user to see a list of their projects, which consists of those downloaded and those created by the user locally. Allows the user to create new projects. • Allows the user to inspect and change a specified project's data. • Allows a user to report times and costs associated with a project
	To see the general structure of the Desktop Client, see the flow diagram in 7.1.2. 1) The Login Screen - provides a level of security for the project data • Inputs - User Name

Subordinates	<ul><li><ul><li>Password</li><li>Server Settings</li></ul></li><li>Outputs<ul><li>Status of the login attempt</li><li>Presents the next form based on connection status and server authentication</li></ul></li></ul>2) The Projects Overview Screen - Shows the user projects which are assigned to him or her and provides access to other screens.<ul><li>Inputs<ul><li>The new task that a user wants to do. The choices are Time/Cost Entry, New Project, and Modify Project.</li><li>In the case of Modify Project, a highlighted entry in the Projects List is an input.</li></ul></li><li>Outputs<ul><li>Displays the Project Maintenance Screen if the user chooses either New Project or Modify Project</li><li>Displays the Time/Cost Entry Screen if the user chooses Time/Cost Entry</li></ul></li></ul>3) The Project Maintenance Screen - Shows details related to the project and provides means for modifying them.<ul><li>Inputs<ul><li>Any data related to the selected project</li><li>Ability to request the creation of new Action Items, Costs, or notes</li><li>Ability to request the saving of modified items</li></ul></li><li>Outputs<ul><li>All data related to the selected project</li><li>The Action Items, Costs, and Notes which have been created for this project</li></ul></li></ul>4) The Time/Cost Entry Screen - Allows the user to report on their activities and costs related to a project<ul><li>Inputs<ul><li>Start Time</li><li>End Time</li><li>Project</li><li>Cost</li><li>Comment</li></ul></li><li>Outputs - None</li></ul>
Dependencies	This component will interact closely with the Server Procedure Proxy for the retrieval of project data. The gui will create and pass around an instance of the Server Procedure Proxy class and use it to get and put data on the server. Such server calls will be synchronous, so there will be no timing issues for the PocketPC client to consider. The Desktop Client will use various data classes found in the Class Library. Data from the Server Procedure Proxy will be packaged into one of these classes, and the Desktop Client can then access that data through the public methods of the appropriate class. These classes also provide the Desktop Client with a handy way to save the data locally (as do all java classes).
Interfaces	The Desktop Client uses Swing classes for interaction with the user. The screen formats for these windows are shown in section 7.2. Uses the Server Procedure Proxy for interaction with a server. For more information on the Server Procedure Proxy see section 3.6.
Resources	The Desktop Client will not have any significantly CPU- or memory-intensive tasks--however, normal overhead associated with Swing components is to be expected in both cases. Much of the project data to be queried will be accessed on a remote database.
Processing	Some psuedocode related to the application can be found here.
Data	A project list kept on the disk as a serialized List of Project objects, loaded into memory in the same format. Such a list will of course initially be empty. Associated project data kept on the disk as a serialized List of Objects, of any of several

	formats, loaded into memory in the same format. Initially empty.

A list of user names and passwords encrypted on disk using javax.crypto.

Back to Table of Contents

4.0 Reuse and relationship to other products

4.1 Java and Java tools

From the beginning we set a goal to make use of any existing code to avoid wasting time duplicating other's work. We also decided to use open source or freeware solutions wherever possible. Because we are creating software for a defined application, it is possible to use open source technologies in each area. First, we decided to implement our code in Java because it is free and allows us to run on a wide range of operating systems. Next, we chose a development environment that would allow us to edit our Java code. Eclipse was chosen because it is free and has many plug-ins which allow us to use already existing applications. Finally, we chose the open source Eclipse plug-in Jigloo to create the GUI because it allows us to drag and drop frames and edit the generated code all within the same environment.

4.2 Database and Reporting

With these decisions in place, we needed a database on which to store our records. We chose Ingres 3.0, because it is open source, has many management tools, and includes a visual interface that is very appealing. We chose JasperReports as the means to generate reports because it is a freeware/open source package written in Java that generates reports in XML, PDF, or HTML files.

Back to Table of Contents

5.0 Design decisions and tradeoffs

5.1 Java/.NET Debate

The first issue we dealt with as a group was whether to develop in Java or .NET. Two of our members wanted to develop in .NET because they are familiar with it and enjoy working with it more. Also, PDA development is easier with .NET. However, while some members of the group do not have .NET experience, all have some Java coding experience. Another important issue we considered in selecting a code was the fact that .NET does not work in all standard operating environments. The range of operating systems that support Java is much larger. Operating systems of interest were: Windows, Unix, Linux, and PocketPC, all of which are supported by Java and not by .NET.

5.2 Three Tier Design

Deciding how to judiciously divide the project between all team members was another design issue. We finally decided on a 3 tier design, which is an application program organized into three major parts, each of which is distributed to different places in a network.

The three parts are:

1. The client application
2. The server application
3. The database and programming related to managing it

A 3-tier application uses the client/server computing model. With three tiers or parts, each part can be developed concurrently by a different team of programmers. Because the programming for a tier can be changed or relocated without affecting the other tiers, the 3-tier model makes it easier for an enterprise or software packager to continually evolve an application as new needs and opportunities arise. Existing applications or critical parts can be permanently or temporarily retained and encapsulated within the new tier of which it becomes a component. This design idea was very appealing to our team, especially for portability purposes. Reference http://searchdatabase.techtarget.com on 3 Tier design.

5.3 Schedule

As a team we also chose not to include a scheduling feature in our software. There are several existing software products that help you plan for the future but few that let you track things done in the past. We also recognized that we would not have enough time to properly implement a scheduling feature and decided to exclude it from our plan.

5.4 Projects/ProjectItems

We also considered using one class for both projects and sub-tasks. This would allow the user to easily upgrade a sub-task to project status, or to assign the sub-task to another user. The only difference between the two would be the amount of information that the user provided. However, after examining the options, we decided that it will be easier to create two different tasks and provide a means by which the user can "upgrade" a sub-task to project status, at which point they can reassign it if they so desire.

Back to Table of Contents

6.0 Pseudocode for components

Our team members have enough experience with Java that it is often easier for us to read and write our pseudocode ideas with Java syntax, so don't be surprised to see plain English mixed in with Java-style language in our pseudocode.

6.1 Client application pseudocode

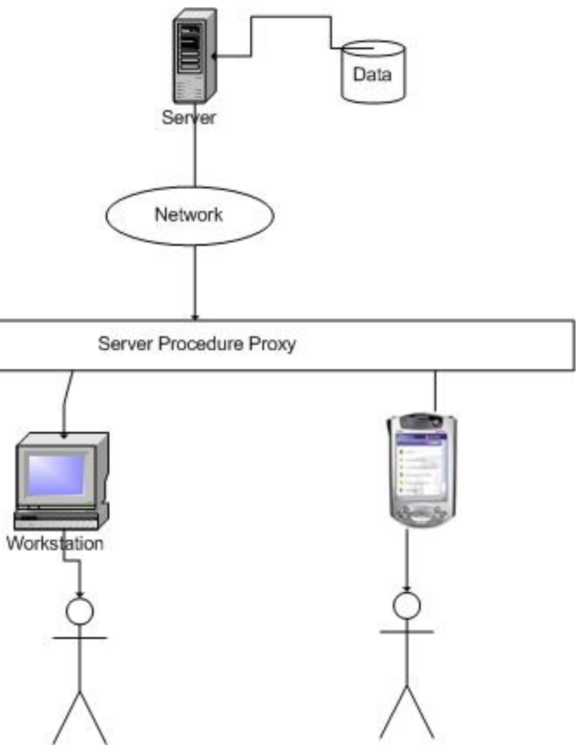
```
Server serv=new Server(usr, pwd, server, port);

try{
            serv.login();
}catch (LoginFailedException e){
            localmode=true;
}

//go to disk and check for local projects
List localProjects = getLocalProjects();
List projects;

//check for server projects

if (!localmode) {
            projects = serv.getAll(Project.class(), “user='user' AND date>='date'”);

            // Synchronize projects
            Synchronize(projects, localprojects);
            // refresh list now that everything's synced up.
            localProjects = getLocalProjects();

            // display current project info
            Iterator i = localprojects.iterator();
            while( i.hasNext() ) {
                        Project proj = (Project) i.next();
                        add project to listbox.
            }
}





listboxdoubleclickhandler() {
            projInfoForm form = new projInfoForm(listbox.selectedProject);
            form.show();
}


projInfoForm {
            Project proj;
            Server serv;
            projInfoForm(Project proj, Server serv);

            showHandler() {
                        List list = serv.getAll(Subtask.class, “project='projnum'”);
                        downloadDate = serv.getDownloadDate();
                        Iterator i = list.iterator();
                        while( i.hasNext() ) {
                                    Subtask task = (Subtask) i.next();
                                    // add subtask info to listbox
                        }
            }
}



public void Synchronize (List serverProjList, List localProjList, Server serv) {
            for each project in server project list,
                        try to find an equivalent project in local project list.
                        If found,
                                    If isDirty(localProj) then display conflict window
                                                If user chooses overwrite local, overwrite local
                                                Else if user chooses overwrite server version then
                                                            serv.put(localproject, downloadDate)
                                                            project.clean();
                                                end if
                                    else overwrite local project.
                        Else
                                    Write as new local project.
                        End if
            next project
            for each project in localProjList,
                        if isDirty(project) and !serverProjList.contains(project) then
                                    serv.put(localProject);
                                    project.clean();
                        end if
            next project

            set lastSynchdate = serv.servDate.
}
```

Back to Table of Contents

7.0 Appendices

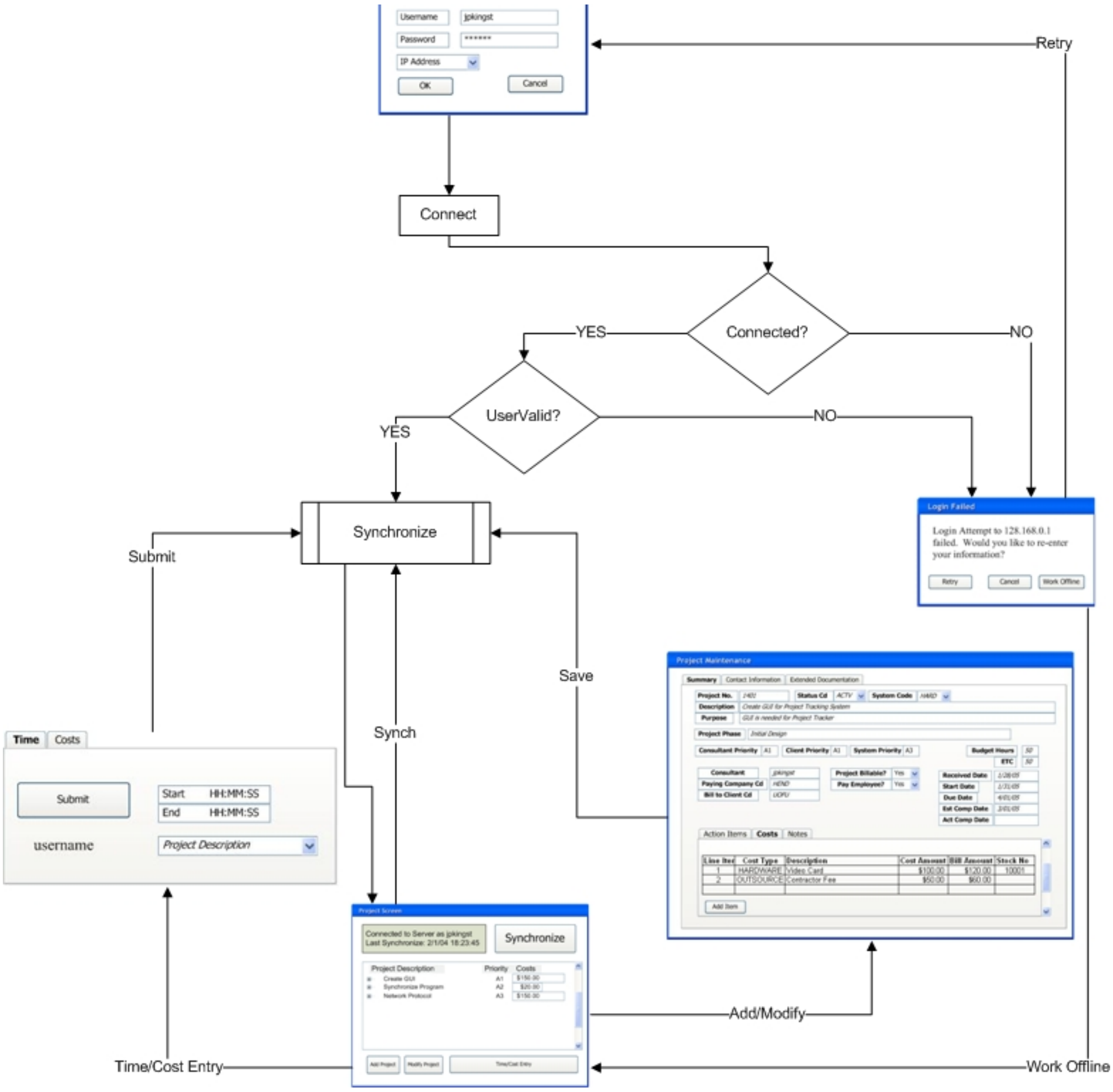
7.1 Diagrams

7.1.1 System Structure Diagram



Back to Table of Contents

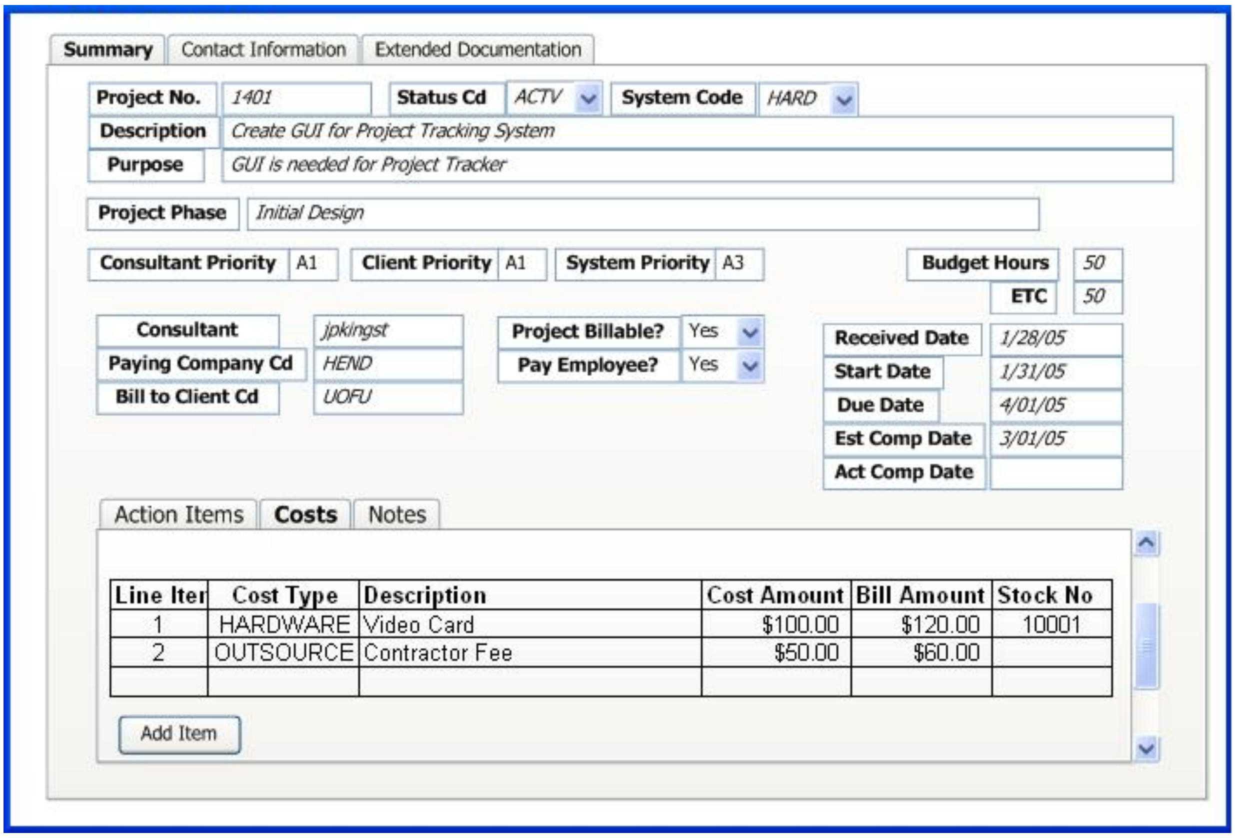
7.1.2 User Experience Flow Diagram



[Back to Table of Contents](#)

7.2 User Interface Screen designs

7.2.1 Project Entry Screen



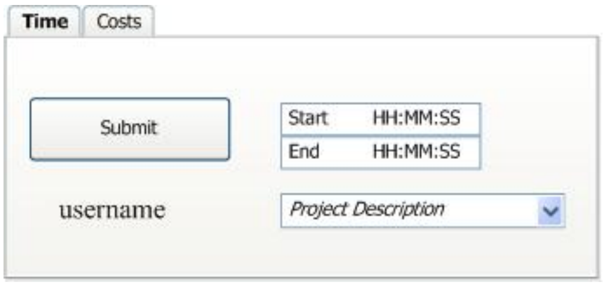
Project No.	1401	Status Cd	ACTV	System Code	HARD
Description	Create GUI for Project Tracking System				
Purpose	GUI is needed for Project Tracker				
Project Phase	Initial Design				
Consultant Priority	A1	Client Priority	A1	System Priority	A3

Budget Hours	50
ETC	50
Consultant	jpkingst
Paying Company Cd	HEND
Bill to Client Cd	UOFU
Project Billable?	Yes
Pay Employee?	Yes
Received Date	1/28/05
Start Date	1/31/05
Due Date	4/01/05
Est Comp Date	3/01/05
Act Comp Date	

Line Iter	Cost Type	Description	Cost Amount	Bill Amount	Stock No
1	HARDWARE	Video Card	$100.00	$120.00	10001
2	OUTSOURCE	Contractor Fee	$50.00	$60.00	

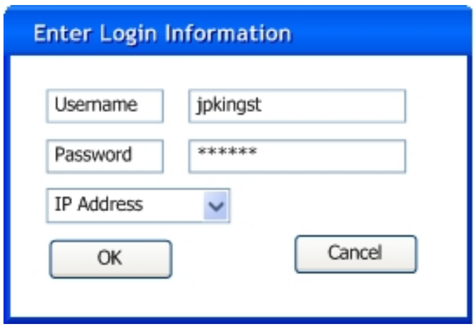
[Back to Table of Contents]

7.2.2 Time Entry Screen



[Back to Table of Contents]

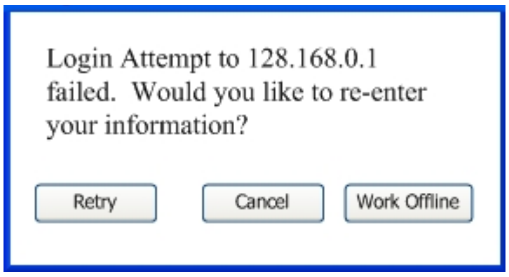
7.2.3 Login Screen



Username	jpkingst
Password	******
IP Address	

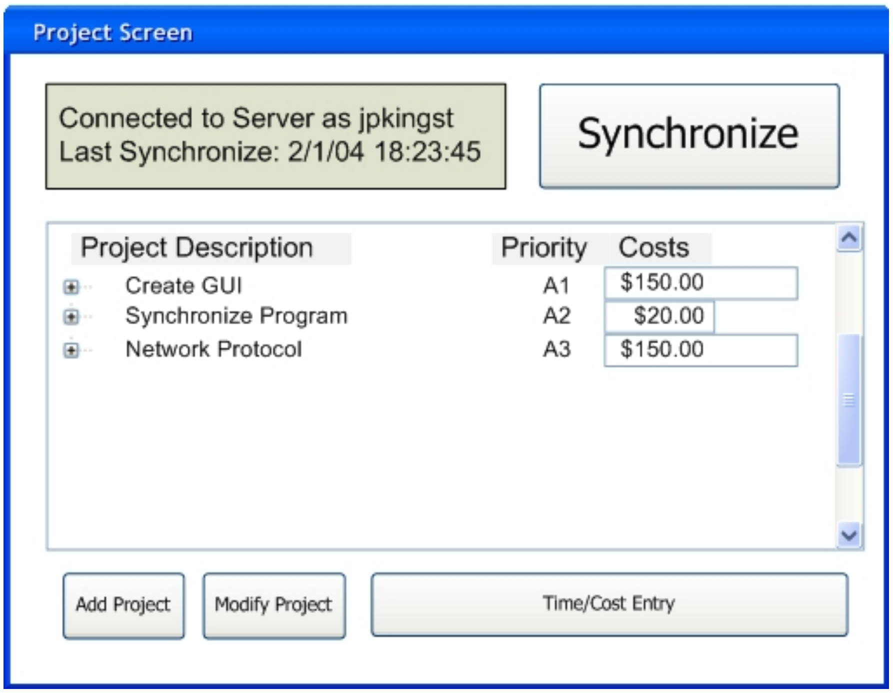
[Back to Table of Contents]

7.2.4 Login Failed Dialog


Login Attempt to 128.168.0.1 failed. Would you like to re-enter your information?

[Back to Table of Contents]

7.2.5 Main Window


Project Screen

Connected to Server as jpkingst
Last Synchronize: 2/1/04 18:23:45

Project Description	Priority	Costs
Create GUI	A1	$150.00
Synchronize Program	A2	$20.00
Network Protocol	A3	$150.00

[Back to Table of Contents]

Change Log

version	date	description
2.0	02/22/2005	- Added component info in section 3, pseudocode, and appendices to make SDS v2.
1.0	02/04/2005	- Software Design Specification document is created

[Back to Table of Contents]