Object-Oriented Software Design Description

Instructions

Version 1.2 ● 14 JAN 2008

TEXAS PROJECT DELIVERY FRAMEWORK

DIR

Version History

This and other Framework Extension tools are available at www.dir.state.tx.us/pubs/framework/extensions/.

Release Date	Description
14-Jan-2008	Version 1.2 released. Modified “Using this Template” section of the Template and italicized all section instructions to align with the Framework and Change Request (CR) #34. CR #34 was recommended by the Framework Change Advisory Board (CAB) and approved by DIR.
13-Mar-2007	Version 1.1 released. Made minor modifications to indicate Framework Extension.
24-May-2006	Version 1.0 – Instructions and Template Released.

Contents

Introduction

The Object-Oriented Software Design Description Template and Template Instructions are included within the within the System Development Life Cycle (SDLC) Extension of the Texas Project Delivery Framework (Framework) to establish a consistent method for documenting the hierarchy of the domains, components, and classes that comprise the software design. A software design is the design fulfillment of the requirements stated in the Software Requirements Specification (SRS) and the basis for the implementation of the software to be built. The Software Design Description (SDD) documents the system architecture and design of the application. The purpose of the SDD is to communicate in sufficient detail how the software is to be constructed and integrated into the system design.

Providing documentation of the software design can reduce project risk by reducing uncertainty in implementation. Documentation of the software design contributes to the success of information technology systems by establishing and communicating how the properties of the software requirements will be transitioned into a design. Expectations for all aspects of the software's features and performance can be contrasted with the design in order to identify and resolve potential design flaws. Identification and resolution of design flaws and problems positively impact the quality and customer satisfaction of the implemented application. In addition, flaws and problems corrected early in the development effort minimize impact to a project's schedule and budget.

Use of the Object-Oriented Software Design Description

Within the Framework, System Development Life Cycle (SDLC) tools are included as an extensible Framework toolset. Use of this toolset is intended to be tailored or customized to meet project requirements and minimize project risk. Project requirements may be met and risk minimized by producing a System Design Description (SyDD) or a Software Design Description (SDD) as the sole design description, or by producing a SDD in conjunction with a SyDD. If the SDD is produced in conjunction with a SyDD and the information within a SDD section has not changed since it was documented in the SyDD, it is appropriate to reference the information in the SyDD from sections of the SDD.

The Object-Oriented Software Design Description is completed to document the hierarchy of the domains, components, and classes that comprise the software design. It is completed, reviewed, and approved in the Project Planning Review Gate. The SDD documents and communicates sufficient details regarding the architecture and design of the system to enable the technical community to produce specifications and construct the software application. Technical resources

that may not be familiar with the project will use the SDD to build the software, or components of the software.

The format of the Object-Oriented Software Design Description Template serves as a basis for creating an actual project document. Customize the Object-Oriented Software Design Description Template, as directed within the Object-Oriented Software Design Description Template Instructions, to contain the sections necessary to comprehensively document the software design.

The SDD should be developed in coordination with and be accessible by appropriate project team and stakeholder entities. In addition, all information in the SDD should be consistent with the Project Plan and the related project documents. All documented system and software requirements, including interfaces, should be addressed by the design.

Approval of the SDD constitutes agreement that the software design documented within satisfies the approved and baselined system and software requirements. Once approved, changes can be made to the design in the SDD only through the change management process.

NOTE: Examples included in the Object-Oriented Software Design Description Template Instructions have no design relationship to each other and are intended for illustration purposes only.

The Object-Oriented Software Design Description must contain descriptive labels for and references to every figure, table, and diagram included within the document.

Section 1. Overview

Provide high-level introductory information about the Software Design Description (SDD) in the following subsections. Include an overview of the entire software design. This section should stand alone as an executive summary.

1.1 Purpose

Describe the purpose of the SDD and its intended audience. Describe the object-oriented software design approach.

1.2 Scope

Describe the scope of the software to be produced. Within the description:

- Identify the software product(s) to be produced and include a short description of the function of each
- Explain what each software product(s) will and will not do. Describe the application of the software being specified and describe the relevant benefits, objectives, and goals

Section 2. System Architecture

Provide and describe a figure that depicts the overall system architecture, including the system component(s) in software, hardware, networks, and any other pertinent major system components (e.g., databases, operating systems) that support the complete system. This depiction will typically require a diagram showing the major hardware components (drawn as titled boxes) and the software that resides on them (as text within the boxes), the major databases (drawn as named cylinders), and any interfaces between these components (drawn as named lines with arrowheads to depict the direction of the interface). An example of an architecture diagram is provided below.

Information in this section must be consistent with existing system architecture documentation for the project.

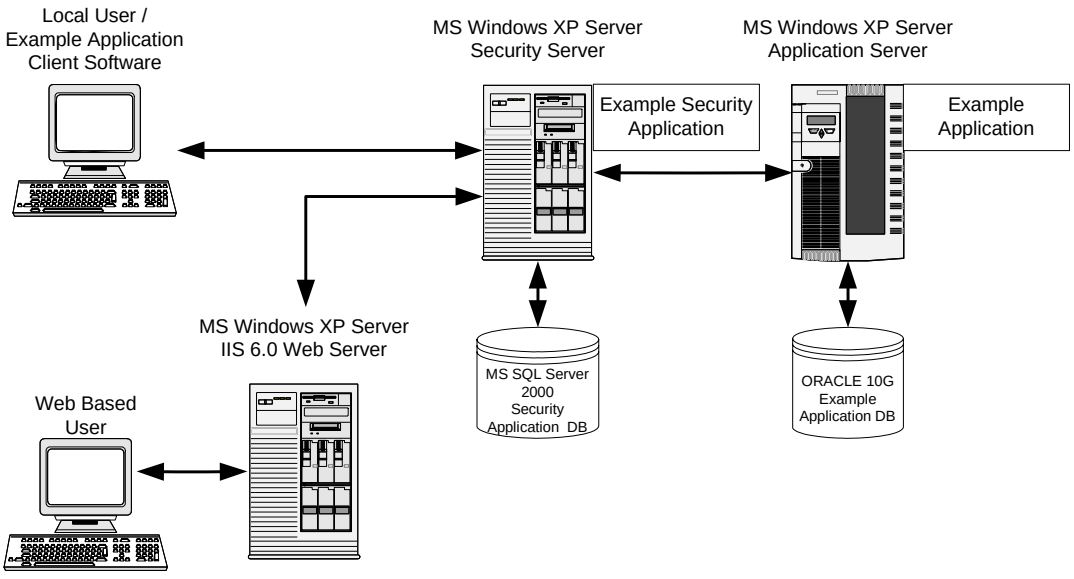

Figure 1. Example of an Architecture Diagram

Section 3. Data Dictionary

Provide a reference to the location of or provide the actual Data Dictionary Table that contains a description of each element in the system. The table should include the entity name and the following details about the data element:

- name
- definition
- data type (e.g., text, character, integer)

- storage format
- scale
- bounds
- display format
- mandatory entry or fill information (e.g., element is required, every character of the element is required)
- default value
- list of functions or other architectural features that can create and modify its values
- list of functions or other architectural features that read its values
- constraints on the data (e.g., some data is protected by Family Educational and Rights and Privacy Act (FERPA))

A sample Data Dictionary Table is provided as an additional tool in the appendix of the System Design Description Template Instructions.

An example of a Data Dictionary Table entry for a customer address table is provided below.

Note: Maintaining the Data Dictionary Table as a separate document and performing appropriate updates in a controlled fashion is more efficient than including it within the SDD and requiring that the SDD be revised each time the Data Dictionary Table is modified.

Entity Name	Element Name	Definition	Type	Storage Format	Scale	Bounds	Display Format	Mandatory Entry/Fill	Default Value	Modified by	Read by	Constraints
Customer Address	Address	Customer Address	Text	Any	n/a	n/a		required	n/a	Purchase, Lease	Purchase, Lease	n/a
Customer Address	City	Customer City	Char	Any	n/a	n/a	Any	required	n/a	Purchase, Lease	Purchase, Lease	n/a
Customer Address	State	Customer State	Char2	Caps Alpha	AZ-WY	n/a	Caps Alpha	required	n/a	Purchase, Lease	Purchase, Lease	Must be valid US State abbreviation
Customer Address	Zip	Customer Zip	Int	5 digit		00000-99999	nnnnn	5 digits required	n/a	Purchase, Lease, Report	Purchase, Lease, Report	n/a

Table 1. Example of a Data Dictionary Table Entry

Section 4. Software Domain Design

Document the software domain design in the following subsections. The software design is represented as a set of domains or views. A domain or view is a representation or description of the entire application for a single technical category, grouping, or perspective. The domains or views form the building blocks of the technology blueprint of the software application. These technical categories provide perspective and structure in the process of representing the design. Although the domains, when combined, form a representation of the whole application, they are largely independent of one another. Domains can include, but are not limited to, business function

operations, reporting, operating system, network, hardware, user, service, application, execution, deployment, and system interfaces.

4.1 Software Application Domain Chart

Provide a figure depicting the set of software application domains showing major components and their relationships. A domain may contain more than one component. An example of a domain chart is provided below.

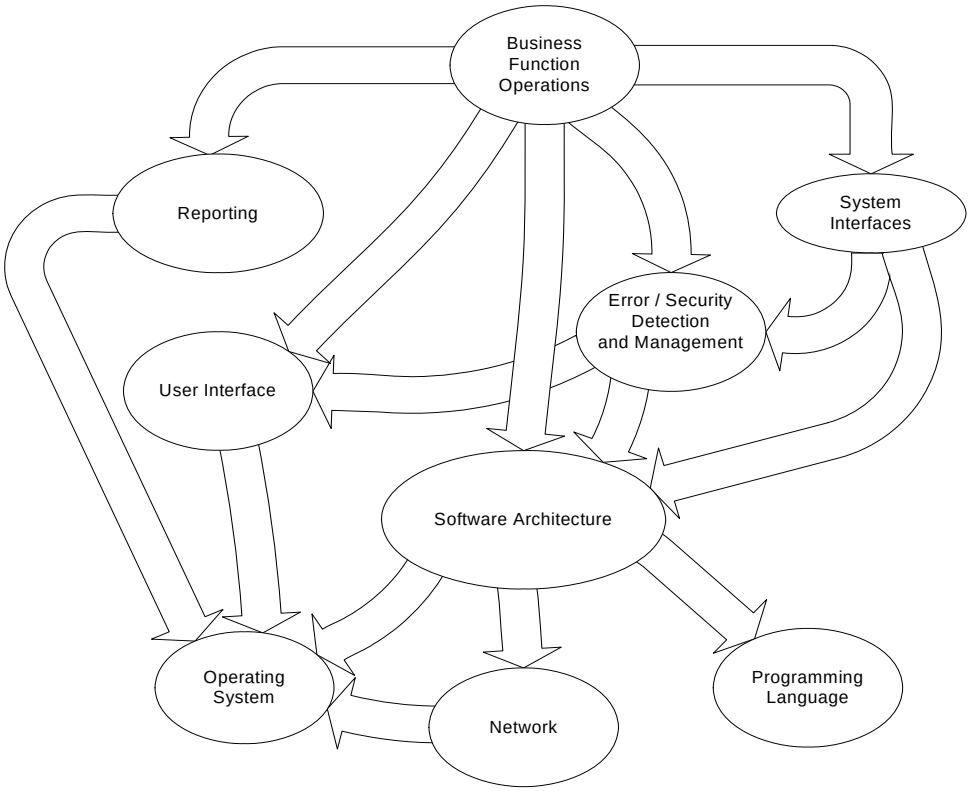

Figure 2. Example of a Domain Chart

4.2 Software Application Domains

Customize this section to contain the subsections necessary to comprehensively document the domains, components, classes, and behavior models (state or activity models) of the software design. Each subsection should be labeled appropriately and titled for a specific domain, component, class, or behavior model. The logical structure of the hierarchy is:

Domain *X*, where *X* is a specific domain name

Component *Y*, where *Y* is a specific component name

Class *Z*, where *Z* is a specific class name

Describe each domain within the design. Depict and describe the hierarchy of domains, components, and classes. These domains may include hardware, application, user, service, and other domains. Any number of domains, components, and classes may exist.

Subsection templates for documenting Domain *X* are provided below.

4.2.*x* Domain *X*

Provide a high-level description of the family of components that make up this domain and provide a class hierarchy chart of the component relationships. Include any database domains and their stored procedures and triggers. If appropriate, provide a hierarchical depiction of the components within the domain. An example of a hierarchy chart for a domain is provided below.

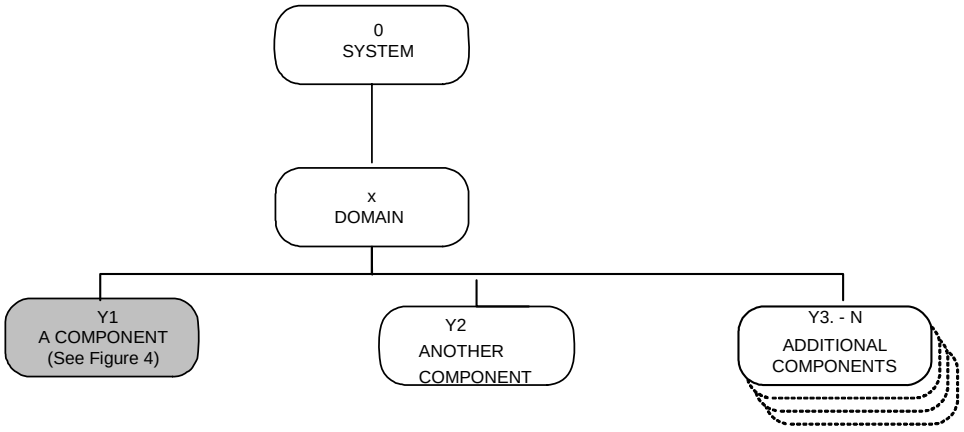

Figure 3. Example of a Hierarchy Chart of the Components within the Domain *X*

4.2.*x.y* Component *Y1* of Domain *X*

Provide and describe a class hierarchy diagram that depicts the set of classes for Component *Y1* of Domain *X*. An example of a class hierarchy diagram for a component is provided below.

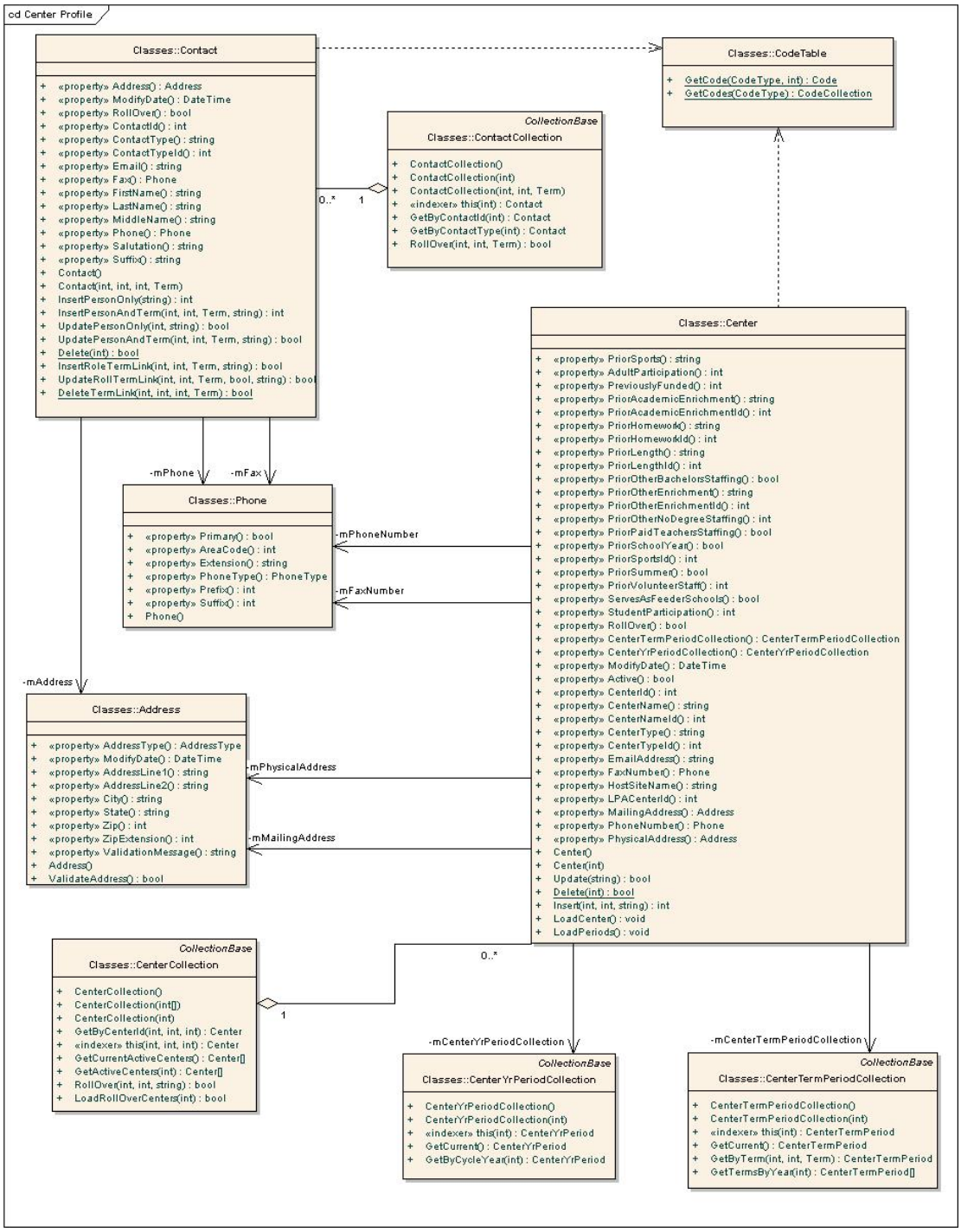

Figure 4. Example of a Class Hierarchy Diagram for a Set of Classes for a Component

4.2.*x.y.z* Class *Z* of Component *Y1* of Domain *X*

Provide a class diagram for Class *Z* of Component *Y1* of Domain *X*. If appropriate, provide a description of any relevant characteristics of Class *Z*. An example of a class diagram is provided below.

Figure 5. Example of a Class Diagram

4.2.*x.y.z.1* Behavior (or Activity) Diagram/Description for Class *Z* of Domain *X* Component *Y1*

For each class that exhibits behavior, provide one or more state or activity diagrams for that class or refer to applicable stored procedures, as appropriate. If the class behavior is trivial or the class has no behavior, provide a description of the class, instead of a diagram. Depict meaningful behavior within each class using the same diagramming technique (e.g., all class behavior depicted through state diagrams) rather than mixing state and activity diagrams to represent behavior in different classes. Rarely, nested state or activity diagrams may be needed to represent complex behavior. An example of a state diagram is provided below.

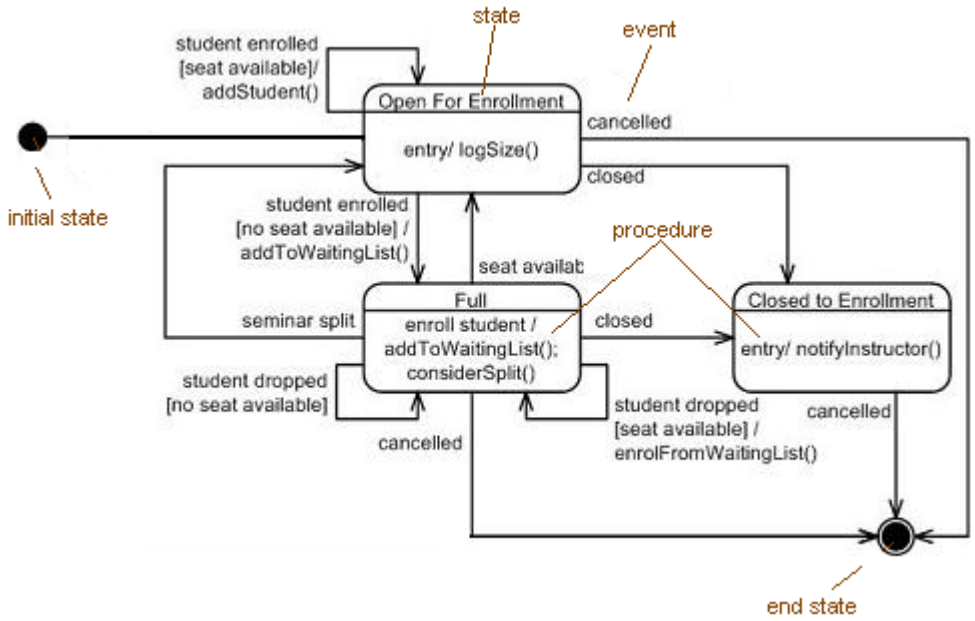

Figure 6. Example of a State Diagram

Section 5. Sequence Diagrams and Descriptions

Customize this section to contain the subsections necessary to comprehensively document the sequence diagrams that depict the interaction behavior between classes. Each subsection should be labeled appropriately and titled for the interaction behavior between classes.

5.*X* Interaction Behavior between Class *X* and *Y*

Provide and describe sequence diagrams that depict the interaction behavior between *Class X* and Class *Y*. The typical sequence diagram will depict some or all of the behavior described in a use case, though this behavior may sometimes be abstracted and generalized to represent a number of similar use cases. This sequence diagram typically shows the classes that interact at the top of a set of swim lanes. A swim lane is a vertical column representing actions on a class as depicted in the following sequence diagram example. Lines depict the behavior between classes with arrowheads showing the class that initiates a call or sends a message to another class. The sequence of these lines down the page represents a logical behavior flow between classes. An example of a sequence diagram is provided below.

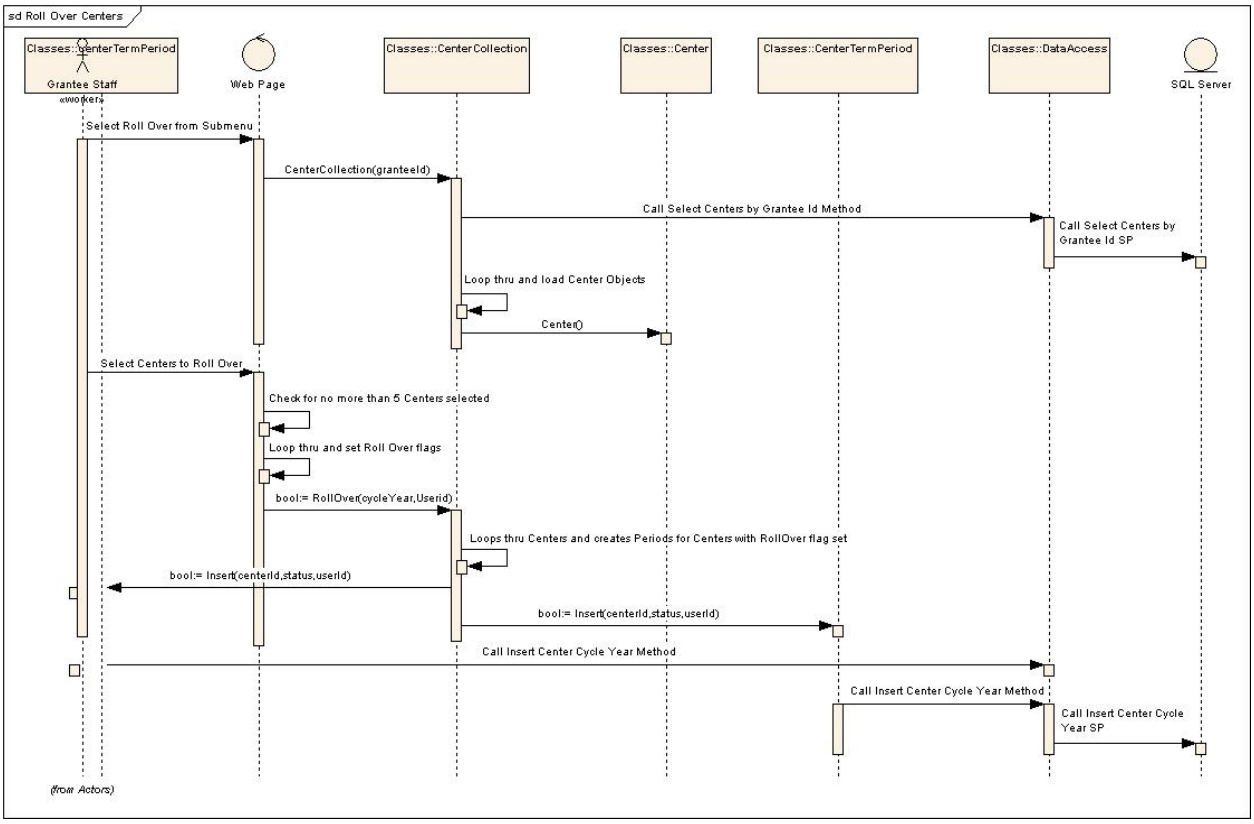

Figure 7. Example of a Sequence Diagram

Section 6. Data Design

Customize the following subsections to describe the data contained in databases and other data structures shared between classes, components, and other major design elements of the software design. Include persistent/static data, transient/dynamic data, external interface data, and transformation of data. Label and title each subsection appropriately.

6.1 Persistent/Static Data

Persistent/static data is data that is stored by a system at the end of execution, and retrieved later for additional processing.

6.1.*X* Persistent/Static Data Store *X*

Describe and provide an illustration of the logical data model or entity relationship diagram(s) for the Persistent/Static Data Store *X*. Include the purpose and general configuration of the data store. An example of a database logical model is included below.

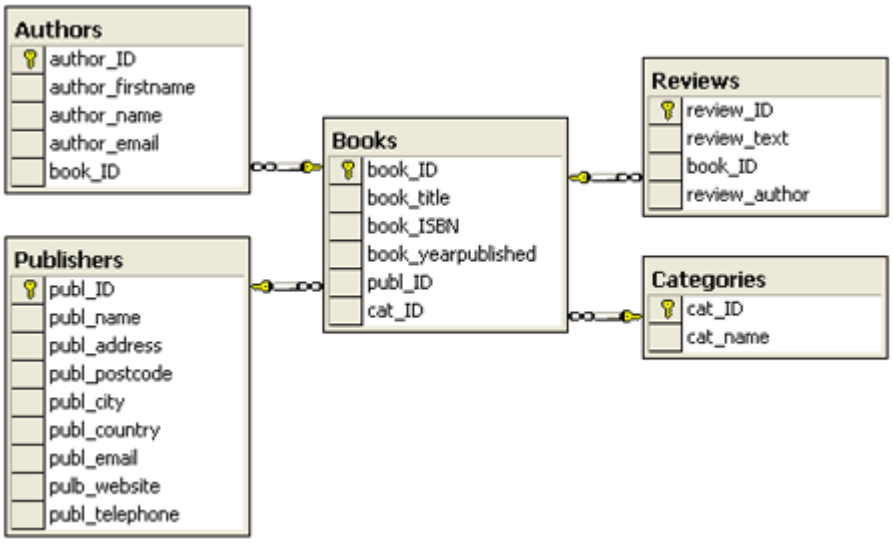

Figure 8. Example of a Persistent/Static Data Store Logical Model

6.2 Transient/Dynamic Data

Transient/dynamic data is data used by the system that does not persist after execution is completed.

Provide a description of the application's transient/dynamic data design and its general configuration. Include the purpose for each of the transient/dynamic data design elements.

6.3 External Interface Data

Describe and, if appropriate, provide diagrams of the external interfaces' data design and its general configuration. Include the purpose for each of the interfaces' data design elements.

6.4 Transformation of Data

Systems often require the transformation of data formats or structures. This is especially true when systems exchange information with other systems. If the application performs explicit data transforms or contains distinct data transform components, enumerate and describe the application's data transformation design. Include the general configuration and purpose for each of the data transform design elements and the transformation mapping rules.

Section 7. User Interface Design

7.1 User Interface Design Overview

Provide a high-level description of the user interface for this software application. Describe any systems requirements (e.g., performance or usability) associated with all of the user interfaces.

7.2 User Interface Navigation Hierarchy

Provide and describe a diagram of the navigation hierarchy that illustrates how a user moves through the user interface. An example of a navigation hierarchy diagram that illustrates how a user moves through the pages of a user interface is provided below.

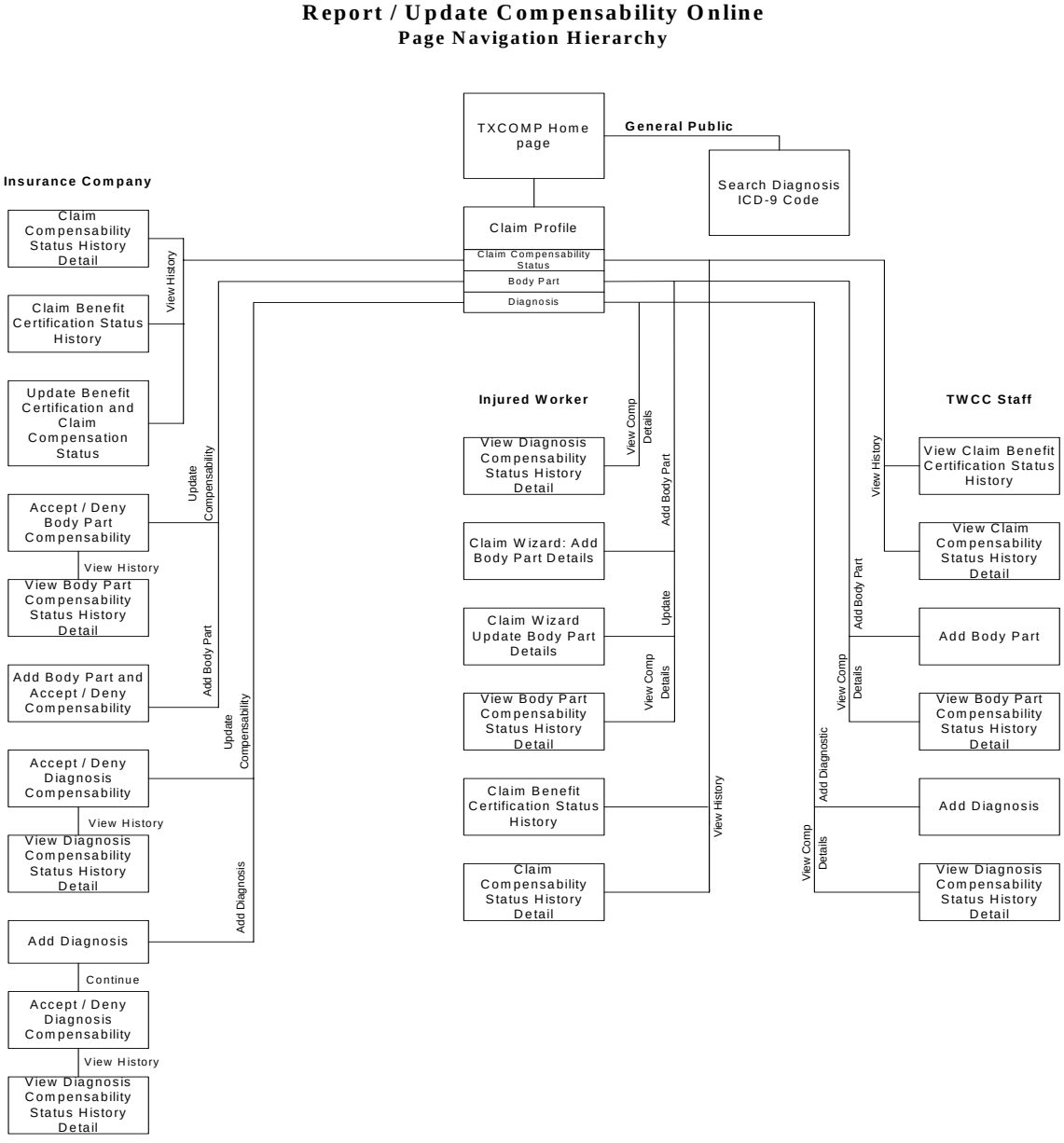

Figure 9. Example of a User Interface Navigation Hierarchy Diagram

7.3 User Function Categories (or Use Cases)

Customize the following subsections to accurately and comprehensively document each category of user function (e.g., transactions, reports, administration) or use case that requires an interface. Typically, a major category of user function correlates to a particular use case in the Software Requirements Specification (SRS) and each use case described in the SRS will require one or more screens for those use cases depicting a user. Document each category of user function or use case individually in a corresponding subsection. Label each subsection appropriately and title each subsection descriptively to indicate the function or use case being documented.

7.3.*X* Function (or Use Case) *X*

Provide a description of the function supporting this category of user interfaces. Where applicable, this description may be derived from its source use case.

7.3.*x.y* Function (or Use Case) *X* Screen/Report Format/Other User Interface *XX*

Provide a description, and if appropriate, an image or mockup of each screen, report, or other user interface within this function or use case. For reports that use standard reporting tools (e.g., Crystal Reports) or standard data exchange languages (e.g., XML), describe the form and formatting for Report *XX* that uses these technologies.

Examples of an image or mockup of a report and screen are provided below.

High School Graduates' Longitudinal Analysis — Statewide

Metrics	Year	2000-2001	1999-2000	1998-1999
Graduates Minimum High School Program		97,827	121,232	112,698
Minimum High School Program (%)		45.43%	56.94%	55.41%
Graduates Recommended and Advanced High School		99,454	76,358	56,398
Recommended and Advanced High School Program (%)		46.19%	35.86%	27.73%
Graduates Distinguished Achievement and Advanced Honors Program		10,661	8,463	27,522
Distinguished Achievement and Advanced Honors (%)		4.95%	3.97%	13.53%
Graduates Individual Education Plan		7,374	6,872	6,775
Individual Education Plan (%)		3.42%	3.23%	3.33%
Total Number of Graduates		215,316	212,925	203,393

Table 2. Example of a Report Format

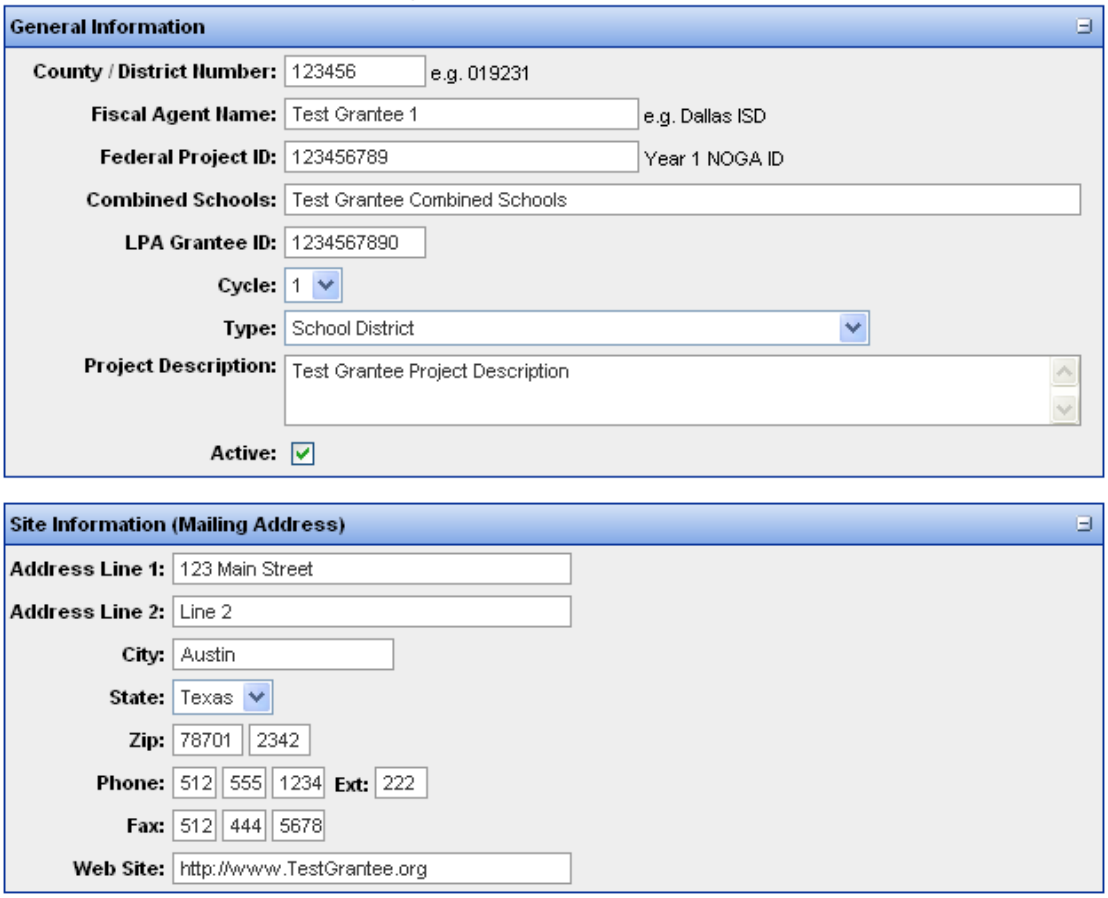

Figure 10. Example of a Screen Mockup

7.3.*x.y.*1 Function (or Use Case) *X Screen/Other User Interface XX* Fields

Provide a table that includes the following information for each field on each screen or other user interface within this function or use case:

- field name
- field label
- data source (e.g., data dictionary source or user entry)
- data type (e.g., text, character, integer)
- storage format
- scale
- bounds
- display format
- mandatory entry or fill information (e.g., element is required, every character of the element is required)
- default value
- constraints or special restrictions on the data (e.g., non-display asterisks for passwords)

In addition, if the data is selected from a pick list, then include the list of possible values or their description. If the content of a field is derived from client-side calculations using other fields or values, then specify the algorithm for the calculation in a descriptive footnote to the table. If the content of a field is derived from server-side calculations or lookups, then specify the source of that calculation (e.g., the class or stored procedure where the calculation occurs).

Also, specify the error messages to be displayed when the input does not meet requirements (e.g., type, format, scale, bounds, or constraints) for the field.

A sample Screen/Other User Interface Fields Table is provided as an additional tool in the appendix of the System Design Description Template Instructions.

An example of a Screen Fields Table is included below.

Field Name	Label	Source	Type	Storage Format	Scale	Bounds	Display Format	Mandatory Entry/Fill	Default Value	Constraints
Dist	County/ District Number	PID database	Num	6 digit	100000	1000001 - 999999	As stored	Yes	n/a	Must match PID lookup
FIS	Fiscal Agent Name	PID database	Text	80 char	n/a	n/a	As stored	Yes	n/a	Must match PID lookup

Table 3. Example of a Screen Fields Table

Section 8. Other Interfaces

Customize the following subsections to accurately and comprehensively document the design of any additional interfaces not described in the previous sections, including specific application-to-application interfaces, database-to-database interfaces, or other interfaces. In addition, identify the technology that will be used to enable the interaction. Label each subsection appropriately and title each subsection descriptively to indicate the interface being documented.

8.*X* Interface *X*

Describe the interface design including technology (e.g., XML), the protocol (e.g., TCP), any specific message formats, error conditions, handshakes, initiation and closure, and other features that define the design of the interface.

Section 9. Other Design Features

Describe any design features that are not captured in the previous sections.

Section 10. Requirements Traceability Matrix

In this section, provide reference to the location of the Requirements Traceability Matrix (RTM) that indicates traceabilty from the:

- System requirements documented in the System Requirements Specification (SyRS) to the design elements documented in the System Design Description (SyDD)
- Design elements documented in the SyDD to the software requirements documented in the Software Requirements Specification (SRS)
- Software requirements documented in the SRS to the design elements documented in the Software Design Description (SDD).

The RTM is initiated in the SyRS and is updated appropriately during the life of the project to indicate traceabilty to the design elements documented in the SyDD, the software requirements documented in the SRS, and the design elements documented in the SDD. The completed RTM assures that every requirement has been addressed in the design and that every design element addresses a requirement. The RTM also provides the necessary traceability for integration, acceptance, regression, and performance testing.

The Requirements Traceability Matrix in the SDD should:

- Contain the columns used to illustrate traceability of the system requirements to the software design elements, software requirements, and software design elements
- Contain the columns necessary to illustrate traceability for integration, acceptance, regression, and performance testing
- Be populated with all requirements documented in the SyRS
- Be populated with all design elements documented in the SyDD
- Be populated with all requirements documented in the SRS
- Indicate traceabilty from the system requirements documented in the SyRS to the design elements documented in the SyDD
- Indicate traceabilty from the design elements documented in the SyDD to the software requirements documented in the SRS
- Indicate traceabilty from the software requirements documented in the SRS to the design elements documented in the SDD
- Indicate the source or origin of each requirement

A sample RTM template is provided as an additional tool in the appendix of the System Requirements Specification Template Instructions.

Section 11. References

Provide a list of all documents and other sources of information referenced in the Software Design Description (SDD) and utilized in developing the SDD. Include for each the document number, title, date, and author.

Section 12. Glossary

Define all terms and acronyms required to interpret the Software Design Description properly.

Section 13. Revision History

Identify changes to the Software Design Description.

Section 14. Appendices

Include any relevant appendices.