5-2011

Utah Preschool Outcomes Data System

Sandeep Venigalla
Utah State University

UTAH PRESCHOOL OUTCOMES DATA SYSTEM

by

Sandeep Venigalla

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

_____	_____
Dr. Stephen W. Clyde Major Professor	Dr. Curtis Dyreson Committee Member

_____
Dr. Stephen J. Allan
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2011

ABSTRACT

UTAH PRESCHOOL OUTCOMES DATA SYSTEM

by

Sandeep Venigalla

Utah State University, 2011

Major Professor: Dr. Stephen W. Clyde
Department: Computer Science

In the State of Utah, both state and federal government agencies work together to provide special education services to eligible children. An important part of this effort is to document the effectiveness of the program, which can be measured in terms of the outcomes, namely, the progress of the individual children. This information can help identify both strengths and weaknesses of special education programs within the state, which in turn can lead to program improvements and better allocation of resources.

This report describes a software system that supports the tracking of child and program outcomes for special education within Utah. Specifically, it provides an overview of the project, the motivation behind the software system, the new technologies used in development, and suggestions for future work.

(37 pages)

ACKNOWLEDGMENTS

I thank Dr. Stephen Clyde for helping me throughout my graduate career and providing his valuable support to me. He not only gave me the technical knowledge but also the inspiration to carry out my work. Dr. Clyde has always taught me good software practice and design, and elegant programming principles.

I am grateful to my committee members, Dr. Curtis Dyreson and Dr. Steve Allan, for their interest in this project and their valuable guidance.

I thank Brian Smith (Vitruvian framework developer) for his co-operation and help. The use of the Vitruvian framework significantly reduced the development time for this project.

I also thank my mother and sister who have always supported me in many ways during my time in school.

Sandeep Venigalla

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Introduction

A child develops many basic life skills during his first five years, such as walking, eating, playing, and interacting with others. Some children have disabilities or developmental delays with respect to these basic skills. Children with a risk of developing such disabilities are referred by their physicians to participate in an *Early Intervention* (EI) program [10], which is provided by most states in the USA and aim to meet the special education needs of these children. In Utah, intervention for children ages 0-2 years, known as EI Part C, is under the purview of the Utah Department of Health (UDOH).

Some children only need EI Part C services to catch them up to their peers. Other children, specifically those of ages 3-4 years, need additional services to help them get ready for kindergarten. These children enroll in the special education program provided the Utah State Office of Education (USOE)[1], also referred to as EI Part B. Children can enroll in EI Part B, even if they did not use EI Part C services.

Each child who receives special education and related services has an individualized education program (IEP). As indicated by the name, an IEP is individualized for a child's specific needs. Tracking the child's progress is then customized to match the plan and measure his or her skill-level relative to average

[1] This special education program is funded by both state and federal governments.

children of the same age. This information can help educators ensure that the IEP is effective and adjust it, if necessary.

This report describes a web application, called Utah Preschool Outcomes Data (UPOD), for tracking the progress of a child during the implementation of the child's IEP. This web application is accessible to all special education teachers, supervisors, center staff, and state staff throughout the state. However, the features and level of child information that they can access varies.

1.2. Overview of Utah Preschool Outcomes Data System

The Baby and Toddler Online Tracking System (BTOTS) keeps track of the children receiving EI Part-C services. The data about children who are likely to need preschool special education services are sent from BTOTS to a system within the USOE. This system, called Transition from Early-Intervention to Preschool Data Input System (TEDI), is responsible for tracking children already in the EI Part-C system who are entering EI Part-B up to the time the implementation of their IEP is started, e.g., they start receiving special education from EI Part-B. Once a child starts receiving special education from EI Part-B, the information is loaded into UPOD, and teachers start to track said child's progress. Figure 1 depicts the relationships among these three systems.

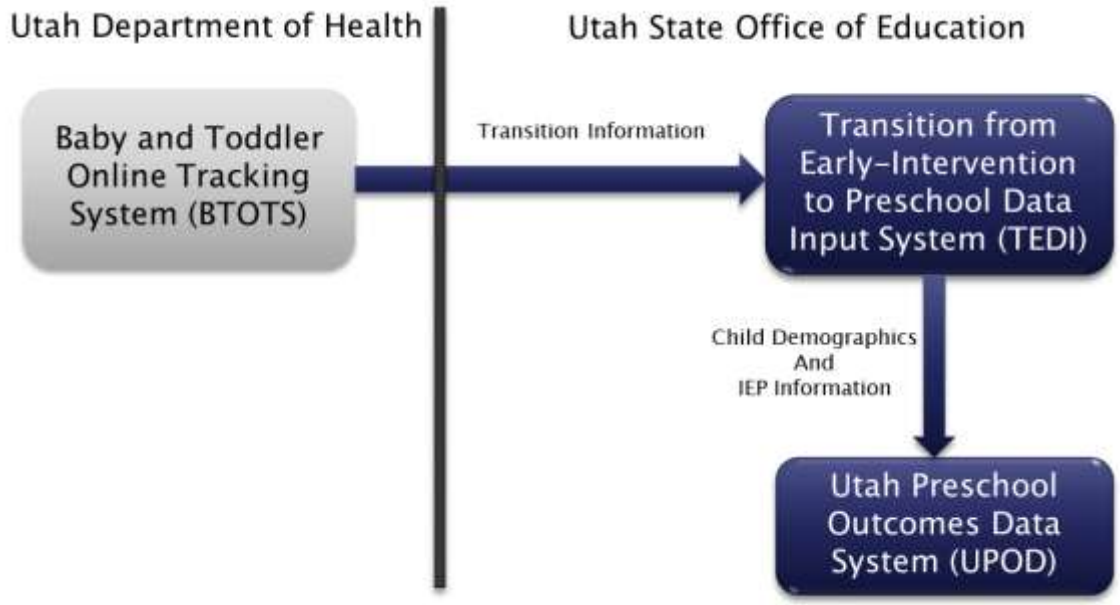

Figure 1: Relationship among BTOTS, TEDI and Utah Preschool Outcomes Data System.

The children in TEDI are not immediately admitted into the special education program, but evaluated for a suspected disability. The evaluation is undertaken by a group of qualified professionals and parents, who in turn follow the guidelines defined by the Individuals with Disabilities Education Act 1997 (IDEA)[1] to determine if a child has a developmental disability. They decide if the child has a disability as defined by IDEA.

If the child is found to have a disability as defined by IDEA, the teachers, special education teachers, and parents meet to write an IEP. Once this is done, the implementation of the IEP is started.

---

1 Individuals with Disabilities Education Act 1997 (IDEA) is a law ensuring services to children with disabilities throughout the nation. IDEA governs how states and public agencies provide early intervention, special education and related services to more than 6.5 million eligible infants, toddlers, children and youth with disabilities [1].

The progress of the child during the implementation of IEP is tracked by the Utah Preschool Outcomes Data (UPOD) system. Special-education teachers evaluate the skills of the child at various stages of the IEP implementation by performing assessments. To support these actives, UPOD provides three general types of assessment:

- **Entry Assessment:** This assessment is done prior to the start of IEP. Performing this assessment is mandatory.
- **Intermediate Assessment:** This is an optional assessment that can be performed by the teacher at any time during the implementation of the IEP.
- **Exit Assessment:** This is done after the implementation of IEP. If the IEP is fully implemented, the child should have an exit assessment.

Each assessment has three or more outcomes that correspond to a set of skills. For each outcome, a teacher rates the child on a scale of 1-7 according to a standard rubric that relates the child's current skill level to those of average children of the same age. The teacher must also record mechanisms used to determine the ratings. These are called the rating sources. Currently, the national standard for assessment requires tracking of the following three outcomes [2]:

- **Positive social relationship:** This outcome rates the child's ability to relate with other children as well as with adults, follow group rules, and generally interact with others.
- **Knowledge and skills:** This outcome rates the child's ability to reason, remember, solve problems, understand symbols, and understand both the physical and social world.

- **Action needs:** This outcome rates the child's ability to take care of basic needs, use tools, move from place to place, and contribute to his own health and safety.

More outcomes may be included in an assessment as needed. So, UPOD must support the addition of new types of outcome and rating sources.

Also, UPOD needs to track children through multiple, non-contiguous, or relocated episodes of service. For example, if the child moves to a new center within the state, the IEP execution is continued at the new center, and assessments are transferred to the new center. However, if the child moves out of state and then later returns, s/he exits the program and re-enrolls when he/she moves back.

CHAPTER 2

SYSTEM ANALYSIS

2.1. Introduction

This chapter documents the requirements for UPOD's use of the Unified Modeling Language[1] (UML) diagrams and the functional requirements. The UML diagrams include use-case diagrams and class diagrams. The use-case diagrams in Section 2.2 provide software developers with a high-level overview of who uses UPOD, as well as said users' goals. The functional requirements listed in Section 2.3 expand on these goals with a more detailed description including constraints of the system's features and behavior. Finally, the class diagrams in Section 2.4 describe the key objects in the system and their relationships to each other from an analysis perspective. This information is provided to help the developer solidify his/her understanding of system components and thus set the stage for database, business logic, and user interface design.

[1] The *Unified Modeling Language*, or UML, provides industry standard mechanisms for visualizing, specifying, constructing and documenting software systems [11]. It includes use-case diagrams, class diagrams, interaction diagrams, state charts, activity charts, and more. Readers who are unfamiliar with UML can refer to any of the many textbooks on the subject, or the official specification published the *Object Modeling Group* (OMG).

2.2. User Goals

A use-case defines the interactions between external actors and the system under consideration to accomplish a goal. An actor specifies a role played by a person or system while interacting with the system [3]. There are four types of actors in UPOD: state user, LEA user, program center user, and teacher. See Figure 2. A state user can manage all the children, teachers, and users in the system, but cannot see certain, private assessments. An LEA user has access to child, teacher, and user accounts in his/her LEA. Similarly, a program center user can manage the children, teachers and users in his/her program center. A teacher can only manage the information of the children in his/her program center.

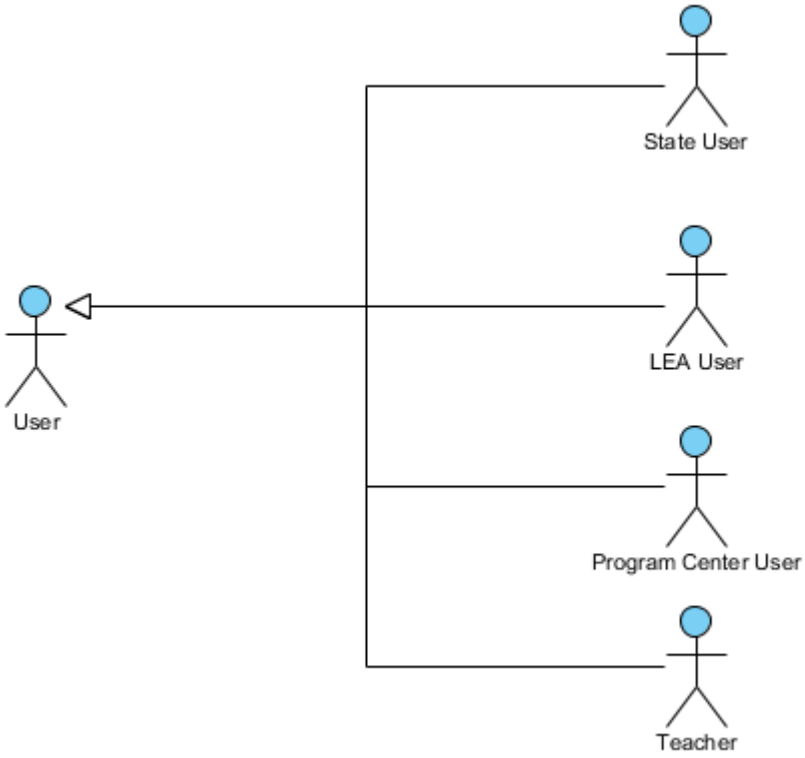

Figure 2: Actors in the system.

The use-case diagram in Figure 3 documents the management of teachers and users in the system. State users, LEA users, and program center users can manage teachers and users. Teachers may or may not be associated with a user account. If the teacher has a user account, the management of the teacher extends the management of the associated user account and vice versa.

Teachers without a user account exist in the system so that the assessments can be attributed to them. Consequently, teachers can never be deleted. They can only be set to inactive. However, users can be deleted.

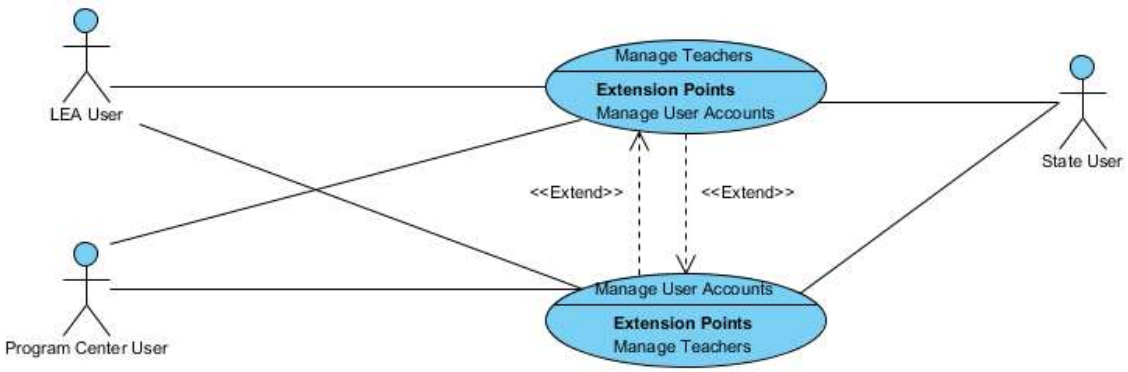

Figure 3: Use-Case - Manage users and teachers.

The use-case diagram in Figure 4 documents the user goals relative to assessments and reports. Teachers, program center users, and LEA users can view, add, and edit assessments for the children. State users can only view assessments. This is because state users are not involved in performing assessments.

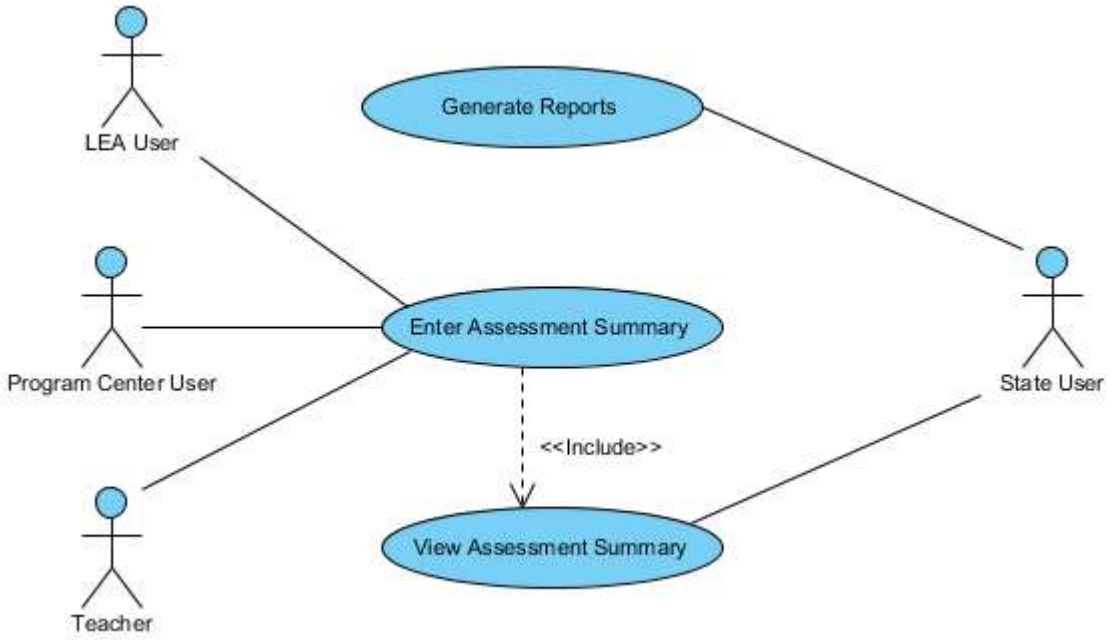

Figure 4: Use-Case - Manage assessments and generate reports.

The use-case diagram in Figure 5 documents the user goals associated with transferring children. A state user can transfer any child in the system. However, an LEA user, program center user, or a teacher can only transfer a child they have access to. This means if an LEA user wants to transfer a child into his/her LEA from a different LEA, he/she must place a transfer-in request. This request is then approved or rejected by a state user. Similarly, if a program center user or a teacher wants to transfer-in a child from outside the program center, a transfer-in request is required. If the child is in the same LEA, the LEA user can process the transfer. Otherwise, the state user must process the transfer.

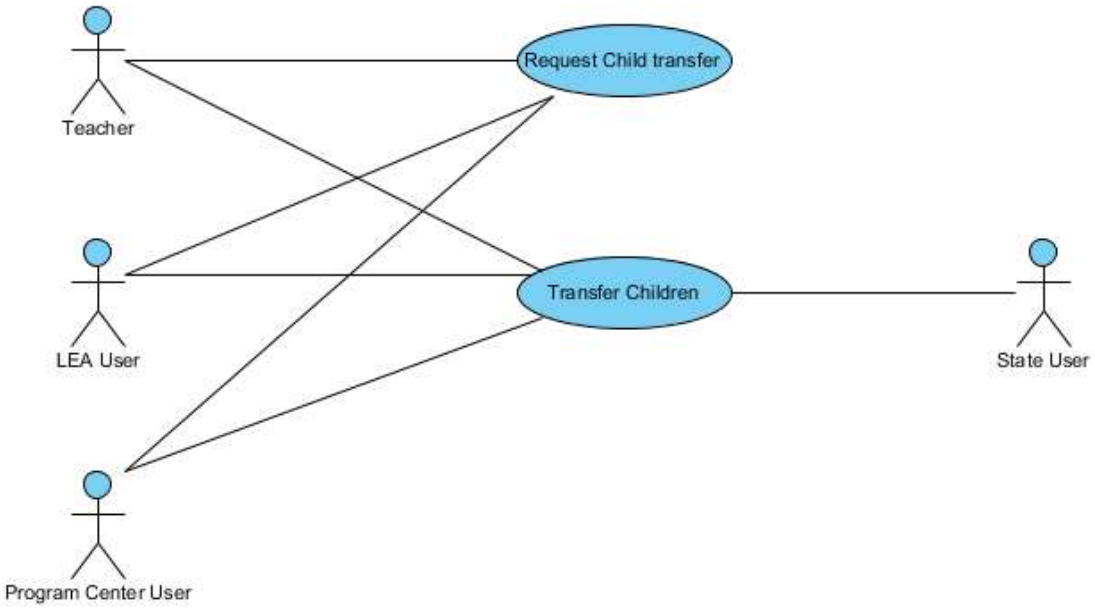

Figure 5: Use-Case - Child transfers.

2.3. Functional Requirements

Functional requirements describe the services the system should provide. Sometimes the functional requirements state what the system should not do. Functional requirements can be high-level and general or detailed expressing inputs, outputs, exceptions, and so on [4]. Since the analysis and design are dependent on the functional requirement specifications, having complete and accurate functional requirements is very important. Having an incorrect or incomplete set of requirements can significantly increase the time and effort required to develop a system, in this case thee Utah Preschool Outcomes Data System.

The functional requirements for this project are as follows:

1. **Usability:** The system should provide an easy to use interface for managing child data, assessments, users, program centers, and teachers. It should also provide for the creation of reports.
2. **Data integrity:** The system should validate all user entered data. It should prompt for any missing information and show easy to understand error messages when necessary.
3. **Security:** Access to the system should be restricted to authorized users with a valid username and password. The system should be designed to deal with hackers and accidental loss of information.
4. **Users:** The following types of users should be supported:
   - 4.1. **State users:** A state user has access to any child, user, or teacher in the system.

4.2. **LEA users:** An LEA user has access to any child, user, or teacher in his/her LEA.

4.3. **Program center users:** A Program center user has access to any child, user, or teacher in his/her program center.

4.4. **Teacher:** A teacher can only access a child in his/her program center.

5. Teachers:

5.1. The system must be able to handle multiple teachers in a program center.

5.2. Maintaining a direct relationship between a child and a teacher is not required. An assessment must be related to a specific teacher. This is important for training teachers and attributing an abnormal sssessment to a specific teacher.

6. User and teacher management:

6.1. The system should allow users to add, modify, and delete users and teachers.

6.2. A teacher may or may not have a user account.

7. **Entry:** A child can enter the system in two ways:

7.1. The child information is loaded into the Utah Preschool Outcomes Data System from the TEDI database; or

7.2. A new child can be added by LEA users, program center users, and teachers.

8. Child information:

8.1. The system should store first name, last name, middle initial, birth date, gender, SSID, and LEA student number.

8.2. SSID should be visible to state users only.

9. Program centers :

9.1. The system should provide for the management of program centers.

9.2. The program centers in an LEA are managed by LEA users.

9.3. State users can manage all program centers.

9.4. A program center can never be deleted. It is made inactive when no longer needed.

10. Transfer:

10.1. When a child is transferred, another program center takes over the enrollment.

10.2. All existing assessments go to the new program center with the child. So, only enrollment per child is required.

10.3. LEA users, program center users, and teachers should be able to transfer a child to another LEA or to a program center within the same LEA. However, they should not be able to transfer a child to a program center outside their LEA.

10.4. A state user should be able to transfer a child to an LEA but not to a specific program center.

10.5. LEA users, program center users, and teachers should be able to place a transfer-in request for a child they need to transfer but cannot view.

10.6. If a child is out of state for more than six months, the child is exited from the system, and the data goes into a report.

10.7. If the child returns, the original enrollment is considered to be continued. The period for which the child is to be gone will be specified in the notes.

11. Child exit:

11.1. A user who can access a child can exit the child from the program.

11.2. A child can exit without an exit assessment. If the child exits without an exit assessment, this information is captured in the exit reason.

11.3. The exit reason and LEA student number should always be captured before a child exits.

12. Child conflict:

12.1. A child conflict alert is generated when a new child's first name, middle initial, last name, gender, and birth date match those of an existing child.

12.2. When a user adds a new child with a conflict, the system should display a warning to the user about a possible conflict. If the child is accessible to the tser, s/he should be able to view the child. Else, the user should be able to place a transfer-in request for the child.

12.3. Child conflict alerts should be viewed and resolved by state users only.

13. Disabilities:

13.1. The disabilities of a child are independent of enrollment.

13.2. Users should be able to choose from a standard set of disabilities.

13.3. Two sets of disabilities are captured for a given child:

13.3.1. **Disabilities on entry:** This is the set of disabilities the child has on entry into the preschool special education program.

13.3.2. **Current disabilities:** This is the current list of child's disabilities.

14. Assessments:

14.1. The system supports entry, intermediate, and exit assessments.

14.2. Teachers, program center users, and LEA users should be able to add, edit, and view assessments.

14.3. State user, however, should be able to view only entry and exit assessments, but not intermediate assessments.

14.4. Each assessment consists of three outcome scores and their sources.

14.5. Outcome score is rated on a 1-7 scale. Users can select from predefined outcome scores and specify the sources.

14.6. The user should be able to enter an assessment date and select a teacher by name.

14.7. For intermediate and exit assessments, the user must specify if progress has been made for each of the outcomes.

14.8. Each child can have a maximum of one entry assessment and one exit assessment. However, there can be multiple intermediate assessments.

14.9. An entry assessment must be entered before an intermediate assessment or an exit assessment is entered.

14.10. For an intermediate assessment, the user can specify if it is “reportable” or “non-Reportable”. A non-reportable assessment should not be not visible to state users, nor should it affect reports.

14.11. A user needs the ability to enter the sources of information for each outcome in an assessment. The source can be selected from a predefined list and/or be entered as text.

2.4. Structural Analysis

Object-oriented analysis (OOA) looks at the problem domain, with the aim of producing a conceptual model of the information that exists in the area being analyzed. Analysis models do not consider any implementation constraints that might exist, such as concurrency, distribution, persistence, or how the system is to be built. Implementation constraints are dealt during object-oriented design (OOD) [5].

Figure 6 shows the analysis-level class diagram for the Utah Preschool Outcomes Data System. The class Child represents a child enrolled in the preschool special education program. This class stores the child's personal information such as name, SSID[1], etc., and information about child's enrollment status in a special education program.

A child has to be in a program center to avail special education services. However, the child sometimes can be in an LEA awaiting program center allotment. In such cases, the Child class has to be associated to both a program center and an LEA. The information about transfers of any child between LEAs and program centers is captured by the Transfer History class.

---

1 Utah SSID or Utah Statewide Student Identifier System is an identification number assigned to each child by the Utah State Office of Education (USOE). The USOE maintains a master SSID database of all students who enroll in a public school along with a few primary attributes (last name, first name, middle name, DOB, gender, school number and LEA student IDs) along with a history of what districts and schools that student has been enrolled in [6].

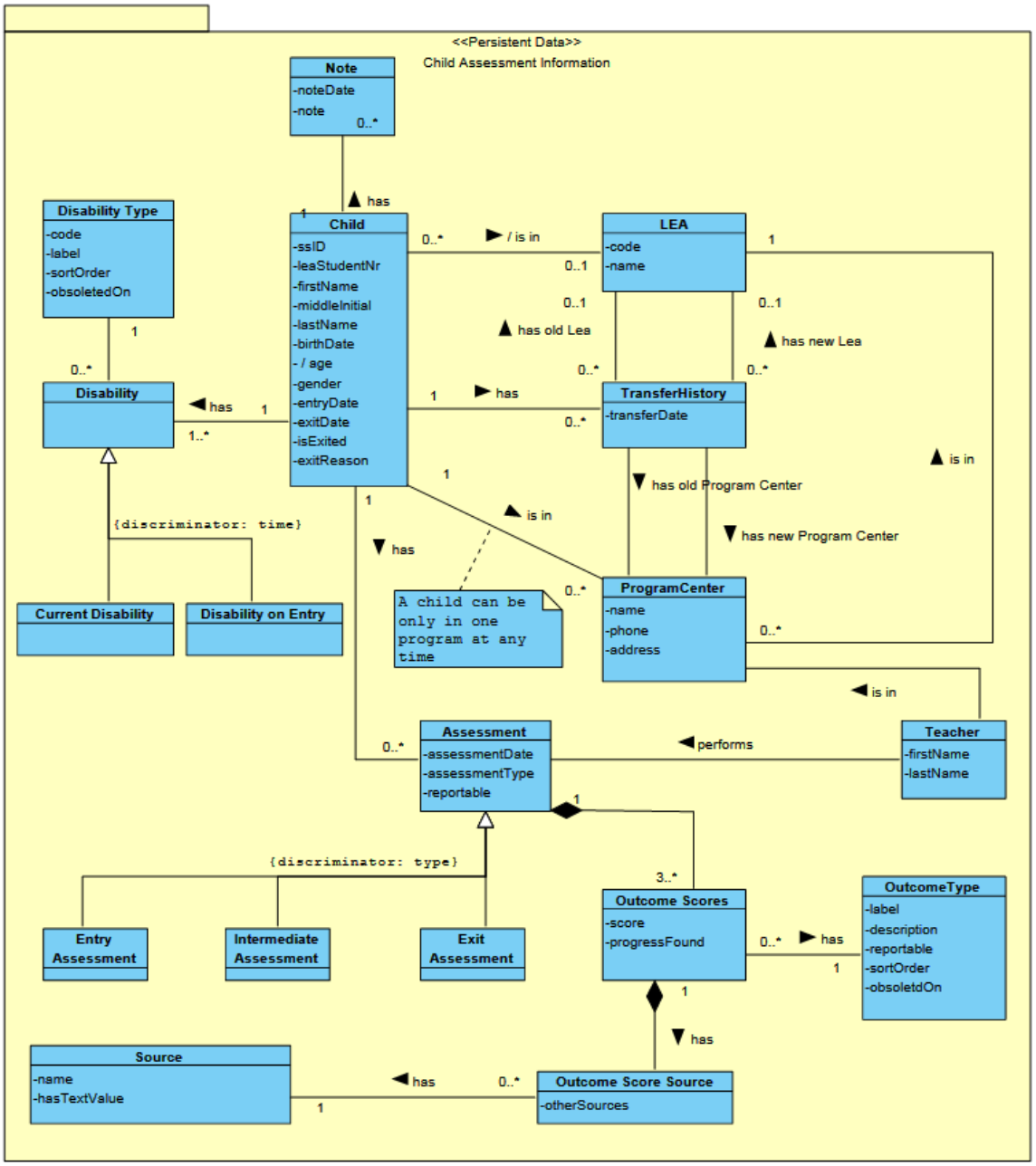

Figure 6: Analysis level class diagram for the system.

The child's developmental delay or disability is captured by the Disability class. There is a predefined set of disabilities that can be attributed to the child. Two sets of disabilities are recorded for each child:

1. Disabilities on entry.

2. Current disabilities.

Teachers assess a child's abilities from time to time. These assessments are classified as entry assessments, intermediate assessments, and exit assessments. Each assessment currently has three outcome scores – one for each outcome type. More outcome types are a possible enhancement to the system in the future. The teacher performing an assessment relies on various information sources for arriving at the outcome scores. Each outcome score has to include at least one outcome score source to capture the source of the information. There is a pre-defined set of outcome score sources. A category called "other" allows the users to enter text in the event that the pre-defined set does proves insufficient.

A LEA user, program center user, or a teacher can enter an assessment into the system. However, only a teacher can perform an assessment. So, the assessment is always associated with a specific teacher. A relation between a child and teacher is not required to associate an assessment with a teacher.

CHAPTER 3

ARCHITECTURAL DESIGN

The architecture of a program or computing system is the structure or structures of the system, comprising software components, the externally visible properties of those components, and the relationships between them. Documenting software architecture facilitates communication between stakeholders, documents early decisions about high-level design, and allows reuse of design components and patterns between projects [7].

The architecture of the UPOD system is depicted in Figure 7. It consists of three layers: presentation layer, application layer, and domain layer.

The presentation layer contains all GUI classes and has forms for managing all the data and generation of reports. This package uses .NET WebForms for most of the forms. The only exception is the form to add/view/edit assessments. It uses a custom control given that .NET WebForms does not have the required user interface features.

The application package contains the following components:

1. **Data transfer service**: This is a Windows service responsible for transfer of data between the TEDI and UPOD systems.
2. **Data transfer utilities**: This is collection of utility classes used by the data transfer service for accessing the database and transfer of data.
3. **User interface utilities**: This is a collection of classes used by the presentation layer. It contains classes for manipulating the DBObjects retrieved from database and data validation.

4. **Report generation utilities**: This is a collection of classes to be used for generating reports. Currently, report generation is not implemented, but a future goal is to use a PDF report generation features in Vitruvian to do so.

The database layer contains the DBObjects and DBLists corresponding to the tables in the database. It is a mapping of a relational database to the object model and is generated by using a wizard provided with Vitruvian.

All the components in the system use features from the Vitruvian framework and .NET framework

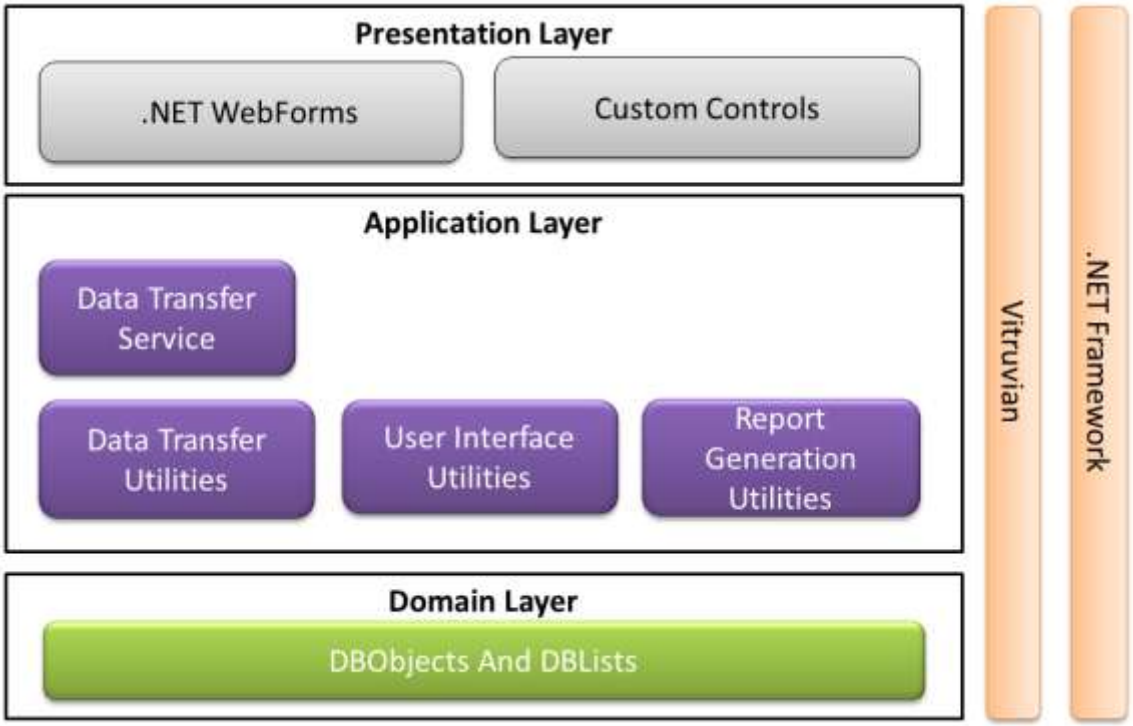

Figure 7: Architecture diagram of UPOD system.

CHAPTER 4

IMPLEMENTATION DETAILS

4.1. Introduction

To implement the UPOD System, we used C#, .Net Framework Version 3.5, and Sybase Adaptive Server Enterprise (ASE) 15.0.3. We also used Vitruvian's DBObjects for database support. USOE stipulates that Sybase must be the database manager since it uses Sybase for most of its other information systems.

4.2.Introduction to Vitruvian DBObjects

One of the problems encountered when mapping an object-oriented language, such as Java or C++, to a declarative language, like SQL, is impedance mismatch. Impedance mismatch is caused by the fact that one object in the application can contain data from multiple tables and multiple rows within a table [8].

There are several techniques for overcoming impedance mismatch. Typically, the developer writes classes for each of the tables or for each of the required objects. Doing so involves writing hundreds, possibly thousands of lines of code. This process is error prone and therefore requires writing lot of test cases. And then there is the added problem of maintaining the classes and their test cases.

Using object-relational mapping (ORM) is a more streamlined approach to overcoming impedance mismatch. ORM is a programming technique for converting data between incompatible type systems in relational databases and object-oriented

programming languages. This creates, in effect, a virtual object database that can be used from within the programming language [9].

We used Vitruvian DBObjects for ORM. The use of DBObjects minimizes, and in some cases eliminates, the need to access the database directly. The database is represented and maintained by DBObjects. Data transfers to and from the user interface are handled by DBObjects (See Figure 8). The DBObjects take care of reading and updating the database.

Figure 8: Relationships among user interface, DBObjects, and database.

Important features of DBObjects include the following:

1. Automatically generate classes for tables and views in the database.
2. Avoid or minimize writing SQL for create, read, update, and delete (CRUD) operations.
3. Navigate between related objects.
4. Lazy loading of objects.
5. Specify filters and sort order for loading DBObject lists from the database.

Vitruvian provides a wizard for generating DBObject classes and DBObject list classes for tables and/or views. Properties are generated in the classes for each of the columns in the corresponding tables/views. The relationships between tables are captured as properties in either or both the related classes. One-to-one relationships are represented as DBObjects while the one-to-many relationships are represented as DBObject lists. The

wizard allows us to choose the relationships to be represented in the generated classes. The user can customize the names of classes, their properties, and relationships.

Vitruvian provides the following methods for using the DBObjects:

1. **Load():** Load the data into the DBObject or DBList. Data can be filtered before loading.
2. **Reload()**: Load the new set of data from database.
3. **Save()**: Save the DBObject to the database.
4. **Delete()**: Delete the DBObject from the database.
5. **ResetValues()**: Reset the values (i.e., all properties) of a DBObject.
6. **RelationalSave()**: Save the DBObject and the children tables of the current DBObjects.
7. **RelationalDelete()**: Delete the DBObject and the children tables of the current DBObjects.

4.3.Experience with and Improvement of DBObjects

This section describes my experience with DBObjects and the improvements I made. I discovered the following problem or bugs and reported them to Brain Smith (Vitruvian developer) in order to be resolved.

Vitruvian initially only supported the use of a persistent connection to the database. This technique is efficient and removes the overhead associated with connection establishment each time we access the database. However, if the connection to the database failed due to some reason, Vitruvian had problems in reconnecting to the database. I reported the problem to Brian and he provided a new way to connect to the database on demand.

Vitruvian caches the recently accessed objects for efficiency. However, a bug in the implementation of the cache caused the application to crash randomly. I also reported this problem to Brian, and he fixed it.

During testing, I discovered SQL injection vulnerability in DBObjects. I reported the problem to Brian, and he gave me suggestions on how I can fix it. In Sybase, injection is prevented by enclosing String, Text, Char, Date, and DateTime data types in single quotes. The single quotes in the data should be replaced by two single quotes. I implemented this fix in the Vitruvian.

I enjoyed using Vitruvian DBObjects for ORM. In any application with many tables and views, developers tend to spend a significant amount of time and effort to overcome the impedance mismatch. Vitruvian DBObjects is a great tool to minimize or eliminate this problem. With ORM taken care of, I was able to concentrate on the more important and complex parts of the project. I also found that using Vitruvian DBObjects is much more reliable and maintainable than writing SQL statements.

4.4. Implementation Details and Challenges

This section describes the implementation details of the Utah Preschool Outcomes Data System, the challenges I faced, and my solutions to those problems.

Learning to work with Vitruvian, while keeping the code clean and structured, was a key goal. DBObjects and control binding are the features of Vitruvian I used in the project. The DBObject-generation wizard generates three files- [name].cs, [name].auto.cs and [name].relation.cs for each table/view. DBObjects need to be regenerated when the database changes. When this is done, the [name].auto.cs and [name].relation.cs files are

regenerated and overwritten. So, I wrote all the custom properties and methods in the [name].cs files.

In the Utah Preschool Outcomes Data System, a teacher may or may not have a user account. So, each time a new teacher is added, the system checks for an existing user account based on first name and last name in the same program center. If a match is found, the teacher is linked to the user account. When a user account for a teacher is added, the system similarly checks for a teacher. If a match is found, the teacher is linked to the teacher.

The implementation of add/view/edit assessment form was a challenge. The number of outcomes and the outcome score sources can change, and such changes must be displayed from the database. There was no .NET control that could handle this. I wrote a custom control-AssessmentFormCtl that generates HTML without using .NET controls. The NET framework incorporates some security checks that help prevent injection attacks. For example, when we use the DropDownList control in a form, .NET makes sure that the value submitted in the form is from the list of allowed values. The problem with generating HTML directly is that we have to incorporate checks to make sure the submitted form values are valid. I incorporated these checks into the custom control.

Most children enter the UPOD system through TEDI. I wrote a Windows service to read the data from the TEDI database and write the qualifying child data into the Utah Preschool Outcomes Database.

CHAPTER 5

SOFTWARE TESTING

5.1. Introduction

Software testing is essential to identify problems in the software, verify the fulfillment of requirements, and ensure the quality of the software. Unit testing, integration testing, and user acceptance testing were all conducted on the UPOD system.

5.2. Unit Testing

Unit testing is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application. In procedural programming, a unit may be an individual function or procedure. In object-oriented programming, a unit is usually an interface, such as a class [12]. Typically, unit test cases are written by developers in the early phases of development.

In the UPOD system, we performed unit testing using the NUnit unit testing framework. The use of DBObjects for ORM eliminated a significant amount of unit testing that would otherwise have been required in a typical web application.

I customized the DBObjects by adding functions and properties. This code was tested using unit test cases. The StringUtil class contains methods for processing and manipulating strings. Similarly, the DateUtil class has methods for processing and manipulating DateTime objects. I wrote test cases for thoroughly testing these classes.

5.3. Integration Testing

Integration testing is the level of testing done to ensure that the various components of a system interact and pass data correctly among themselves, as well as function cohesively [13].

During the integration testing of the UPOD system, the test cases focused on the classes supporting the import of data from TEDI system. I wrote test cases to verify the transfer of child information in the TEDI database to the UPOD database. This was done using the TEDI database populated with random test data.

I also wrote test cases to verify the generation of child conflict alerts to cope with the possibility of duplicate children entering the system while transferring data from TEDI.

5.4. System Testing

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements [14]. System testing is performed after integration testing to verify the fulfillment of the requirements.

In the initial states of system testing, I tested the system against the requirements. Later, the system was shown to select officials from USOE. The requirements identified for this release of the system were met. Some changes had to be done to the graphical user interface based on the feedback from USOE officials.

5.5. User Acceptance Testing

The objective of user acceptance testing is to confirm that the application under test (AUT) meets its business requirements and to provide confidence that the system works correctly and is usable before it is formally “delivered” to the end user(s).

The UPOD system is currently undergoing the user acceptance testing. The system is deployed on a test server, and a subset of the end users is testing the system. The database design and data integrity aspects are being evaluated by a team from the IT department of USOE.

CHAPTER 6

CONCLUSION AND FUTURE WORK

Presently, the Utah Preschool Outcomes Data System allows collection and viewing of data on individual children in the special education program. However, the system needs to be able to generate reports for evaluating the strengths and weaknesses of the program. So, the next step in development should be a feature to enable generation of customizable reports.

The child information in BTOTS and TEDI is currently kept in sync, even after the child transfers to TEDI. However, once the data is transferred from TEDI to the Utah Preschool Outcomes Data System, there is no feature to synchronize the data between the two systems. The data transfer Windows service needs to be enhanced to allow synchronization of data.

Working on this project, I was responsible for the analysis, design, and development of the software. The resulting product is software designed to be flexible to change and easy to maintain. Further, the product fuses streamlined and efficient database design with object-oriented programming.

REFERENCES

[1] U.S. Department of Education. Building the Legacy: IDEA 2004. http://idea.ed.gov/ (Accessed November 2011)

[2] Utah State Office of Education. UPOD Summary. http://www.schools.utah.gov/sars/DOCS/preschool/upodssummary.aspx (Accessed November 2011)

[3] Wikipedia. Use case. http://en.wikipedia.org/wiki/Use_case. (Accessed June 2011.)

[4] Laplante, P.A. *What Every Engineer Should Know about Software Engineering*. Taylor & Francis, 2007.

[5] Wikipedia. Object-oriented analysis and design. http://en.wikipedia.org/wiki/Object-oriented_analysis_and_design. (Accessed June 2011)

[6] Utah State Office of Education. Introduction to Utah Statewide Student Identifier System (SSID). http://www.schools.utah.gov/computerservices/Meetings-and-Conferences/May-2005/DOCS/SSIDIntroduction.aspx. (Accessed October 2011)

[7] Bass, L., Clements, P., and Kazman, R. *Software Architecture in Practice*, 2nd ed. Addison-Wesley Professional, 2003.

[8] Garmany, J., Walker, J., and Clark, T. *Logical Database Design Principles*, 1st ed. Auerbach Publications, 2005

[9] Wikipedia. Object-relational mapping. http://en.wikipedia.org/wiki/Object-relational_mapping (Accessed July 2011)

[10] U.S. Department of Education. *Idea 2004: Building the Legacy -- Part C (Birth – 2 Years Old)*. http://idea.ed.gov/part-c/search/new (Accessed October 2011)

[11] Eriksson, H-E., Penker, M., Lyons, B., and Fado, D. *UML 2 Toolkit*. Wiley Publishing, 2004.

[12] Wikipedia. Unit testing. http://en.wikipedia.org/wiki/Unit_testing (Accessed September 2011)

[13] Craig, R.D. and Jaskiel, S.P. *Systematic Software Testing*. Artech House, 2002

[14] Wikipedia. System testing. http://en.wikipedia.org/wiki/System_testing (Accessed September 2011.)