A Feature-Oriented Software Engineering Approach to Integrate ASSISTments with Learning Management Systems

by

Hien D. Duong

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

_____

May 2014

APPROVED:

_____
Professor George T. Heineman, Thesis Advisor

_____
Professor Gary Police, Thesis Reader

_____
Professor Craig Wills, Department Head

Abstract

Object-Oriented Programming (OOP), in the past two decades, has become the most influential and dominant programming paradigm for developing large and complex software systems. With OOP, developers can rely on design patterns that are widely accepted as solutions for recurring problems and used to develop flexible, reusable and modular software. However, recent studies have shown that Objected-Oriented Abstractions are not able to modularize these pattern concerns and tend to lead to programs with poor modularity. Feature-Oriented Programming (FOP) is an extension of OOP that aims to improve the modularity and to support software variability in OOP by refining classes and methods. In this thesis, based upon the work of integrating an online tutor systems, ASSISTments, with other online learning management systems, we evaluate FOP with respect to modularity. This proof-of-concept effort demonstrates how to reduce the effort in designing integration code.

Acknowledgements

It is my privilege to have Professor George Heineman as my advisor for my graduate research. Without his generous support, warm guidance and patience, this work would not have been possible. Thanks Professor Heineman for his advice and guidance on both academics and life.

Thanks to my thesis reader, Professor Gary Police, for his time and valuable suggestions.

Thanks to Professor Neil T. Heffernan, Cristina Heffernan and David Magid for detail workflow suggestions while I developed ASSISTments apps. Without your participations, ASSISTments apps would never be Live in Edmodo Store during the time I worked on my thesis.

Thanks to ASSISTments lab mates who help me a lot while doing the development work of ASSISTments.

Last but not least, I would like to thank my wife. This thesis is a product of her unconditional love, support, and encouragement throughout years.

Content

List of figures

List of tables

Chapter 1. Introduction

1.1. Motivation

Before the advent of mass production, the manufacturing process required handcrafted work and each product was unique, in the sense that is was built from scratch. During the age of industrialization, mass production based upon assembly lines used standard parts were constructed individually but then could be combined/assembled to create more complex products. The focus on standardized products reduced production costs and improved the quality of products and processes. Recognizing that different customers have different needs and wishes, manufacturers started to increase diversity in their product portfolios. In a way, mass production is similar to mass customization with just a few variations. Below is an example from the fast-food industry of mass production with individualized configuration [8].
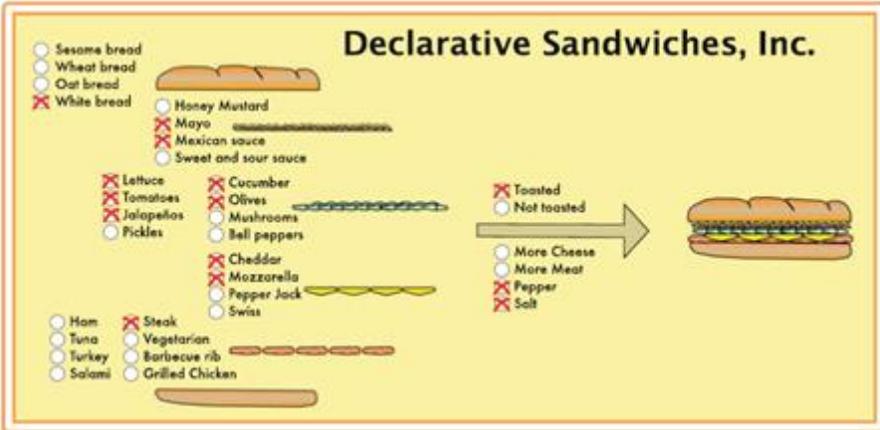
Figure 1.1. Modeling variability in a domain

1.2. Literature review on software integration perspective

We have found no other research targeting the use of feature oriented programming (FOP) in integration code at the time that this thesis was written. However, there is a similar engineering technique that aims to develop software systems by reusing pre-existing *software components* rather than features called Component-based software engineering (CBSE). Lau *et al* **Error! Reference source not found.** summarized that component-based approaches tend to use the concept of composition by taking two or more components then putting them together in some way. Component composition mechanisms fall into two main categories: direct message passing and indirect message passing. In general, with direct message passing scheme, there are two distinct role: the sender and the receiver. And when components are connected by direct message, data flow and control flow are mixed with the computation, and thus the message tends to "hard-wired" into component. It makes sender and receiver tightly couple together. Connection by indirect message passing typically happens with glue code that passes messages between components. To connect a component to another component, a connector is used, when notified by the former, invokes a method in the latter.
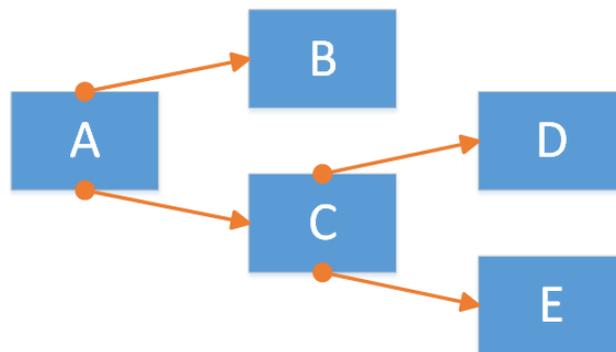
Figure 1.2. Connector with direct messaging
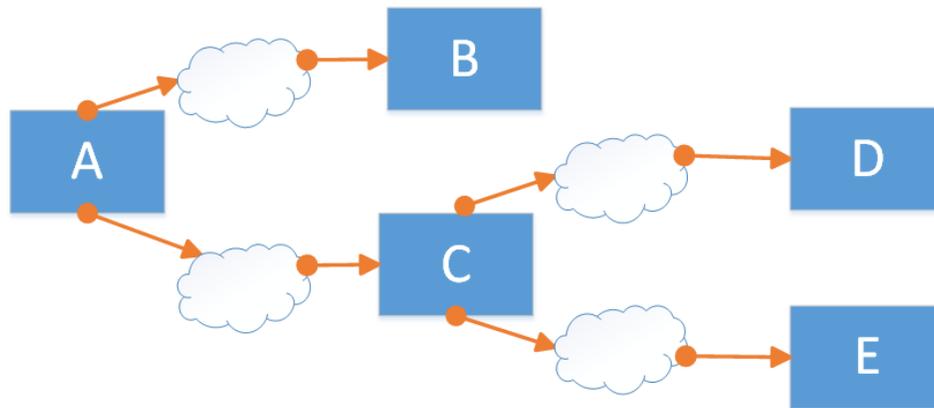
Figure 1.3. Connector with indirect messaging

Example of direct messaging are remote procedure calls (RPC). Models that adopt direct messaging include the CORBA Component Model (CCM) [17]. Models that adopt indirect message include *JavaBeans* [18].

1.3. The Need to Integrate Learning Management Systems

Learning mathematics in classroom's today is different than it was twenty years ago. While there is no definite proof for the one "right way" to teach mathematics, it increasingly important for teachers to adopt effective teaching strategies. Incorporating technology into the teaching of mathematics has proven to be an effective method of mathematics **Error! Reference source not found.**. Hadley and Sheingold suggest that technology is most valuable to teaching and learning once teachers integrate it as a tool into everyday classroom practice and into subject-matter curricula **Error! Reference source not found.**. The need to integrate between technologies emerges when teachers need to use a variety of teaching activities, while each integrated learning technology is designed only to deliver one particular set of instructional content.

Besides the purpose of integrating learning technologies for sharing content across, there are increasingly requirements for utilizing data between systems for researching purpose. As Harmelen and Workman identify, learning analytics refers to the interpretation of a wide range of data, which can be collected by outcomes data across a wide variety of learning tools [20]. For example, ASSISTments collect data from students such as assignment performance but social interactions, provided by Edmodo from its social learning platform, are not directly assessed as part of student's educational purpose.

Chapter 2. Background

2.1. Software Product Lines

Software product lines emerged since late of 1960s and gained more momentum in software industry from 1990s. The main idea is that software systems should be constructed from reusable parts instead of being developed from scratch. And instead of composing a software system always in the same way, it should be based upon the customer's requirements, where customers can choose from pool of configuration options. A clear example of a successful software product line is the Linux kernel which runs on a variety of platforms, such as embedded devices, desktop systems, and large-scale servers **Error! Reference source not found.**. Linux also supports different applications, from office software and games, to high-performance computing and server software. In able to efficiently supports all kinds of different platforms and application scenarios, Linux allows users to choose among large set of options (up to 10,000 features according to Tartler *et al.* [23]) to define the Linux kernel to fit their needs. The Figure below is a screenshot of Linux's configuration tool, called *Kconfig*.
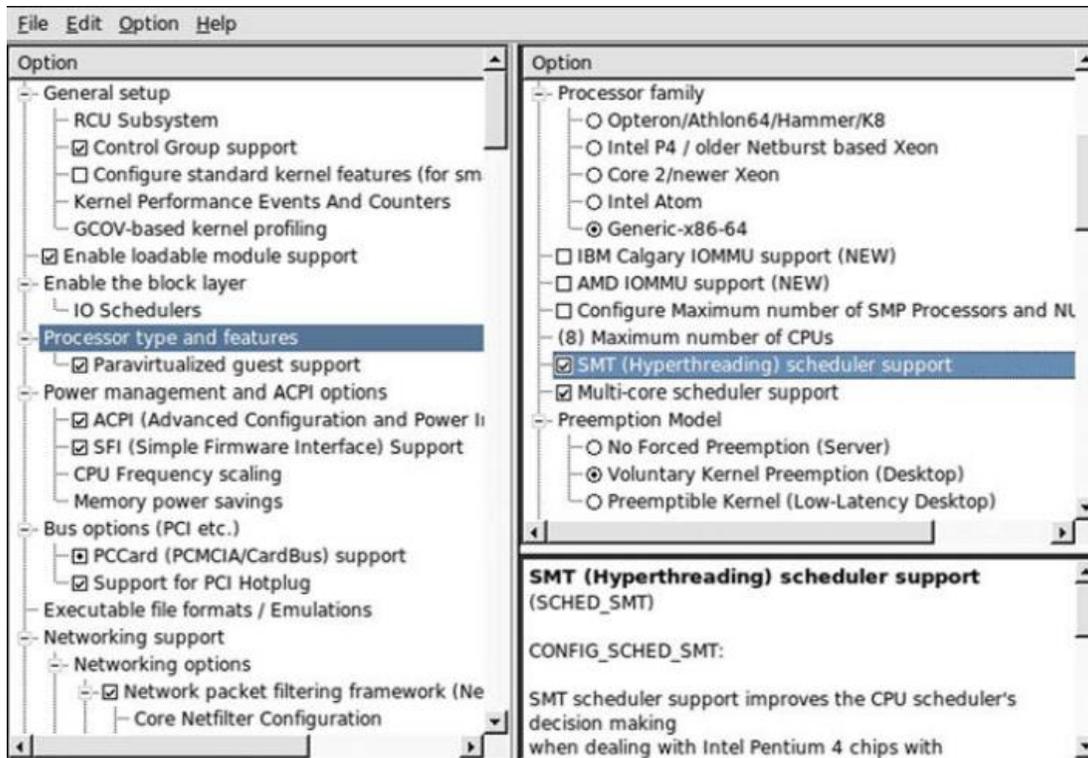
Figure 2.1. Linux's kernel configuration tool

Clearly, the industrialization of software development is facilitated by software product lines. Ideally, based on set of reusable parts, a software manufacturer can generate a software product that adapt to certain customer's requirements. The concept of feature is a core concept to distinguish the products of a product line. For example: some customers requires Email client that supports both IMAP and POP3 but others only need POP3.

2.2. What is Feature

Feature-Oriented Programming (FOP) is a paradigm for the construction, customization, and synthesis of large-scale software systems. FOP is the study of feature modularity and programming models that support feature modularity. The concept of a *feature* is at the heart of FOP. A feature is a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option. The basic idea of

14

FOP is to decompose a software system in terms of the features it provides. The goal of the decomposition is to construct well-structured software that can be tailored to the needs of the user. Typically, from a set of features, many different software systems can be generated that share common features and differ in other features. The set of software systems generated from a set of features is also called a software product line. In other words, A product line shares a common set of features developed from a common set of software artifacts [3][5][6].

According to Czarnecki, feature models, in their basic form, contain mandatory/optional features, feature groups, and implies and excludes relationships [8]. A feature model is a tree of features, whose root encapsulates the base feature, the minimum unit of functionality required for the existence of the system. Other nodes in the tree represent either solitary features, which can be optional or mandatory, or grouped features, which can be either exclusive-or groups or or-groups.

2.3. Eclipse FeatureIDE

FeatureIDE is an open-source solution tool for product line implementation, targeted primarily at researchers, teachers, and students. FeatureIDE is installed using the Eclipse plug-in mechanism. In FeatureIDE, the whole application is divided into parts representing different features. While this may sound similar to the concept of object-oriented classes there is an important difference. A feature in our sense always represents a certain aspect of the application.

Every feature can be related to an arbitrary number of software artifacts. In FeatureIDE, these artifacts can be classes, methods, fields or even single statements as well as additional resources like graphics or user-documentation. Especially, the option to change only parts of a method offers great flexibility in the design of features. In software product lines, not all

combinations of features are considered valid and lead to useful software systems. A feature model defines the valid combinations of features in a domain **Error! Reference source not found.**. Features models have a hierarchical structure, whereas each feature can have sub-features **Error! Reference source not found.**. The graphical representation of a sample feature model is feature diagram and example is shown below:
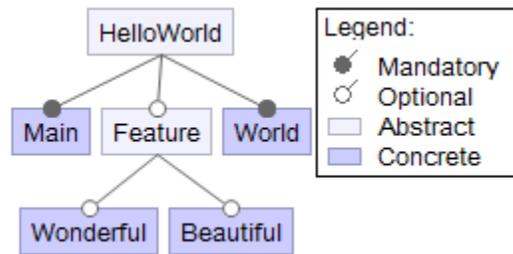
Figure 2.2. Sample Feature Model

The features **Hello** and **World** are mandatory and simply print the features name. The features **Wonderful** and **Beautiful** are not required. Connections between a feature and its group of sub-features are distinguished as: *and*, *or*, and *alternative* [25]. The children of and groups can be either mandatory or optional. A feature is abstract if it is not mapped to implementation artifacts and concrete otherwise [26]. A feature model may also have cross-tree constraints to define dependencies which cannot be expressed otherwise.

Feature models are a common notion for variability and their semantics is as follows: the selection of a feature implies the selection of its parent feature. Furthermore, if a feature is selected, all mandatory sub-features of an *and* group must be selected. In *or* groups, at least one sub-feature must be selected and in *alternative* groups, exactly one sub-feature has to be selected.

In FOP, classes are decomposed into feature modules, each implementing a certain feature. A feature module may contain methods and fields of several classes. Feature modules can be composed into a program based on a given configuration and order of the features.

Using the example in the figure above, here are the actual object-oriented classes contained within each of the designated features:

Hello Feature

```
public class Main {
     public void print() {
          System.out.print("Hello");
     }

     public static void main(String[] args) {
          new Main().print();
     }
}
```

Figure 2.3. Hello Feature Sample Code

Wonderful Feature

```
public refines class Main {
     public void print() {
          Super().print();
          System.out.print(" Beautiful");
     }
}
```

Figure 2.4. Wonderful Feature Sample Code

Beautiful Feature (similar to Wonder Feature)

World Feature.

```
public refines class Main {
     public void print() {
          Super().print();
          System.out.print(" World!");
     }
}
```

Figure 2.5. World Feature Sample Code

Example of valid configurations are shown below:
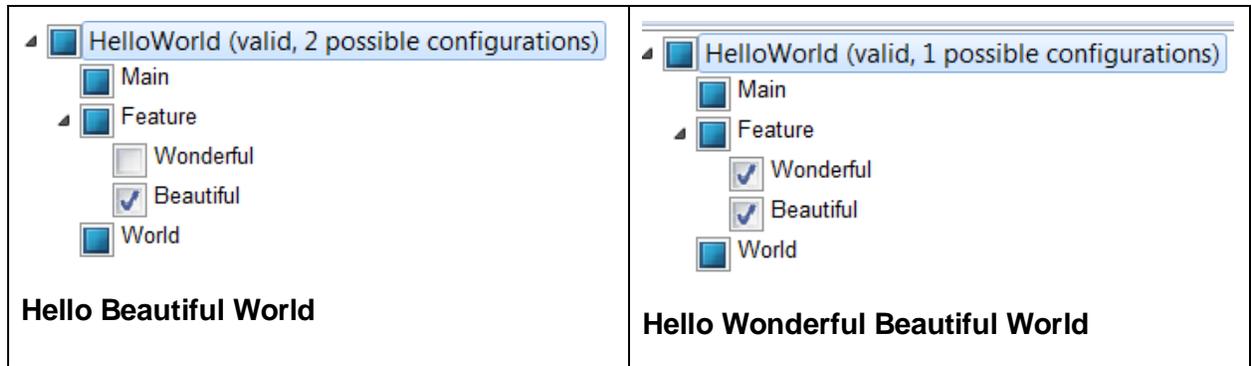
Figure 2.6. Valid HelloWorld Configuration

2.4. Integrating Systems

2.4.1. ASSISTments

Assistance and assessment are integrated in ASSISTments, a web-based math tutoring system for 7th-12th grade students which offers instruction to students while providing a detailed evaluation of their abilities to teachers. The ASSISTments System is being built to identify the difficulties individual students—and the class as a whole—are having, and teachers will be able to use this detailed feedback to tailor their instruction to focus on those difficulties. Unlike other assessment systems, the ASSISTments system also provides students with intelligent tutoring assistance while assessment information is collected. Tutorial help is given if a student answers the question wrong or asks for help. The tutorial help assists the student learn the required knowledge by breaking the problem into sub questions called scaffolding or giving the student hints on how to solve the question.

Figure below shows a screenshot of an ASSISTments problem with three scaffolding questions. Solving this problem involved understanding congruence, perimeter, and equation solving. If the student had answered correctly, she would have moved on to a new problem. However, she incorrectly answered 23, and the system responded with, "Hmm, no. Let me break

this down for you." It then presented the student with some questions that would help to isolate the skills with which she had difficulty and to tutor her so that she could figure out the correct actions. The tutor began by asking a scaffolding question that isolated the step involving congruence. Eventually she got the scaffolding question correct (by answering "AC") and then was given a question about perimeter. The figure shows that the student selected ½ * 8 * x as the formula for perimeter, and the system responded with a "buggy message" letting the student know she seems to be confusing perimeter with area. The student requested two hint messages, as shown at the bottom of the screen. The tutoring ends with a final question, which is actually the original question asked again. The student then will go on to do another math problem and will again get tutoring if she gets it wrong.
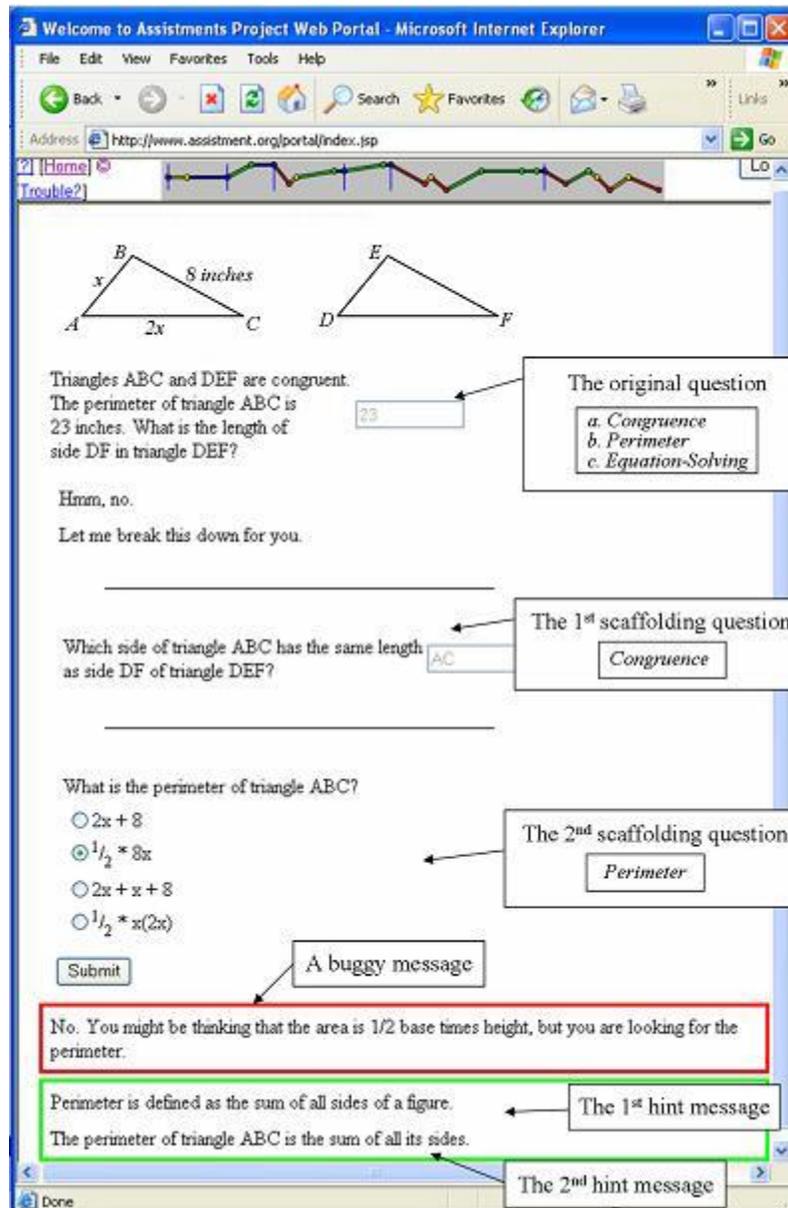
Figure 2.2. Sample ASSISTments screen

ASSISTments provides a set of RESTful APIs that allow external partners to integrate with the system. Some highlight APIs that allows to create users account, ask for access_token, and forward users to go inside ASSISTments without asking for credentials. Those APIs that allow ASSISTments to interact with other systems in seamless manner, without interruption, rely on Single Sign On capability.

Some examples ASSISTments APIs include:

User Login:

Request:	
Type:	GET
URL:	assistments.org/api2_helper/user_login ?partner=<partner id> &access=<access token> &on_success=<success callback> &on_failure=<failure callback>

Create User

Request:	
Type:	POST
URL:	assistments.org/api2/user
Request Headers and Key-Value Pairs:	
Header	assistments-auth
partner	Required: Partner ID (ASSISTments provided)

Payload:	{
	    "userType": "<principal | proxy>",
	    "firstName": "<text>",
	    "lastName": "<text>",
	    "displayName": "<text>",
	    "timeZone": "<UTC offset>",
	    "username": "<username>",
	    "password": "<password>",
	    "registrationCode": "<code>",
	    "email": "<email>"
	}

2.4.2. Edmodo

Edmodo can be incorporated into classrooms through a variety of applications including Reading, Assignments, and Paper-studying. Current uses include posting assignments, creating polls for student responses, embedding video clips, create learning groups, post a quiz for students to take, and create a calendar of events and assignments. Students can also turn in assignments or upload assignments for their teachers to view and grade. Teachers can annotate the assignments directly in Edmodo to provide instant feedback.

Parents can also view this website, either under their child's username or they may create their own account. The Parent accounts allow parents to see their children's assignments and grades. Teachers, subject to creating and maintaining parental records, could send alerts to parents about school events, missed assignments, and other important messages. Similarly, teachers can, subject to creating and maintaining class-participant data, generate printable class rosters. so if a teacher is going to have a substitute teacher in their classroom who needs a printed roster, they can print one from an Edmodo account.

Figure 2.3. Sample Edmodo screen

Student and, possibly parental, data is normally already maintained in a school's information management system and so would require ongoing effort and care to duplicate and maintain data on Edmodo outside the school's own security controls.

Edmodo, as with any social network, can be used as a place to post and critique work, facilitate collaboration, and post creative writing for an audience. Educational social networking

sites, like Edmodo, offer an opportunity to "connect with students and help them create norms and reflect on how different online actions will be interpreted. Edmodo and other social networking sites offer educators a chance to explore the use of social networks and use of media and online formats. Edmodo is used worldwide but mainly from the US. In Edmodo, teachers can put posts with attachments such as videos or pictures from their iPad, iPhone or computer and put it in a group folder in which pupils and teachers can access the post in a safe learning environment. It can be used to teach children how to create podcasts, posts and basic website designing.

With Edmodo's Apps API, Apps can integrate with Edmodo's core features such as: Students can turn in Edmodo assignments, teachers can upload grades into teachers' grade books, and content can be stored immediately into teachers' libraries

This figure below list an example Edmodo APIs:

Launch Request

Example Request

```
GET /launchRequests?api_key=<API_KEY>&launch_key=5c18c7
```

Example Response

```
{
    "user_type":"TEACHER",
    "access_token": "atok_abbd3e01",
    "user_token":"b020c42d1",
    "first_name":"Bob",
    "last_name":"Smith"
    "avatar_url":"https://edmodoimages.s3.amazonaws.com/default_avatar.png",
    "thumb_url":"https://edmodoimages.s3.amazonaws.com/default_avatar_t.png",
    "time_zone":"PST",
    "user_id":1234567,
    "groups":[
        {
            "group_id":379557,
            "is_owner":1
        },
        {
            "group_id":379562,
            "is_owner":1
        }
    ]
}
```

2.4.3. inBloom

inBloom funded by the Bill & Melinda Gates Foundation and the Carnegie Corporation, with mission is to provide a valuable resource to teachers, students and families, to improve education. inBloom makes it easier for teachers to see a more complete picture of individual student progress than what most currently have access to through its secure, single-access point. With this information teachers are able to better identify where each student needs extra attention, and to tailor education materials that maximize the one-on-one time they spend with

24

students. Further, currently it's difficult for teachers to find the many valuable instructional materials that exist across the country or even in their own school districts. Through its resource index, inBloom saves teachers valuable time by helping them more easily search for and share these materials.

inBloom provides REST API, full client-server Web service with features such as: Bult-in HTTP basic access authentication, support JSON data-interchange format and HTTP methods to exchange representation of resources.
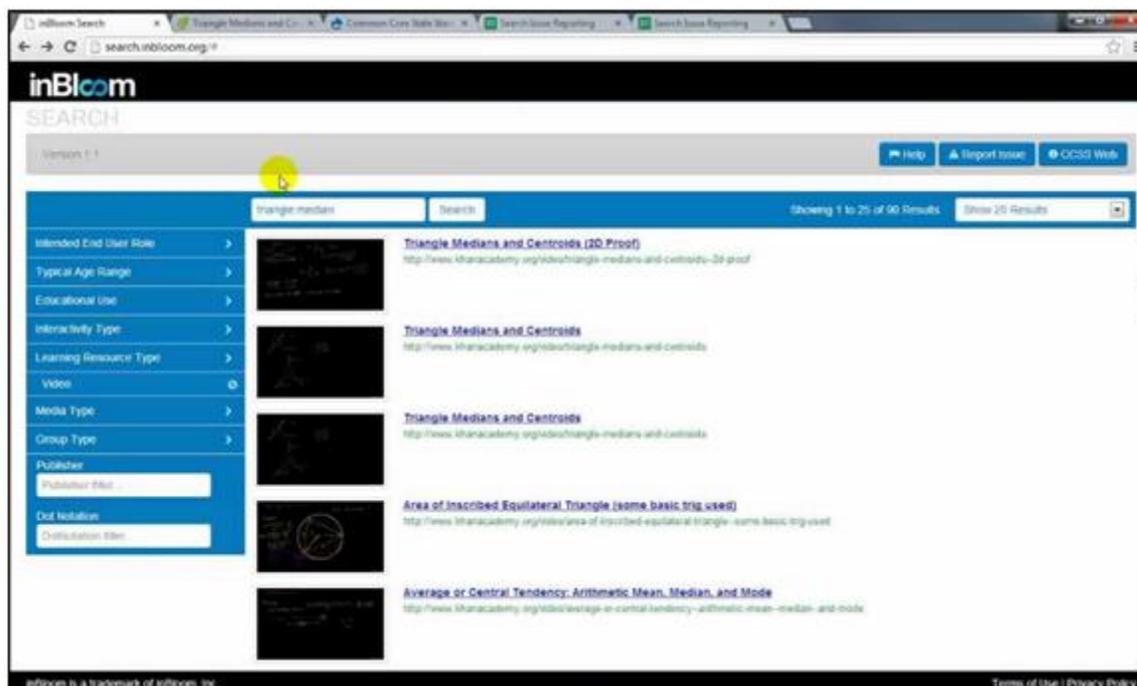
Figure 2.4. Sample inBloom screen

This figure below list an example inBloom APIs:

Example request/response for /schools/{id}

Request:

```
GET https://localhost/api/rest/v1.3/schools/{id}
```

Response:

```
{
  "id": "{id}",
  "schoolCategories": [
    "Junior High School"
  ],
  "organizationCategories": [
    "School"
  ],
  "gradesOffered": [
    "Sixth grade",
    "Seventh grade",
    "Eighth grade"
  ],
  "address": [
    {
      "nameOfCounty": "Wake",
      "streetNumberName": "111 Ave B",
      "postalCode": "10112",
      "stateAbbreviation": "IL",
      "addressType": "Physical",
      "city": "Chicago"
    }
  ],
  "educationOrgIdentificationCode": [
    {
      "identificationSystem": "School",
      "ID": "East Daybreak Junior High"
    }
```

Chapter 3. Case study

Heineman introduced an approach to construct encapsulate varieties via layers [9]. His undergraduate class developed dozen of plugin components for a card solitaire game engine with MVC (Model-View-Controller) design patterns. To maximize reusable code, he developed a set of layers that can be assembled to form solitaire variation plugins.
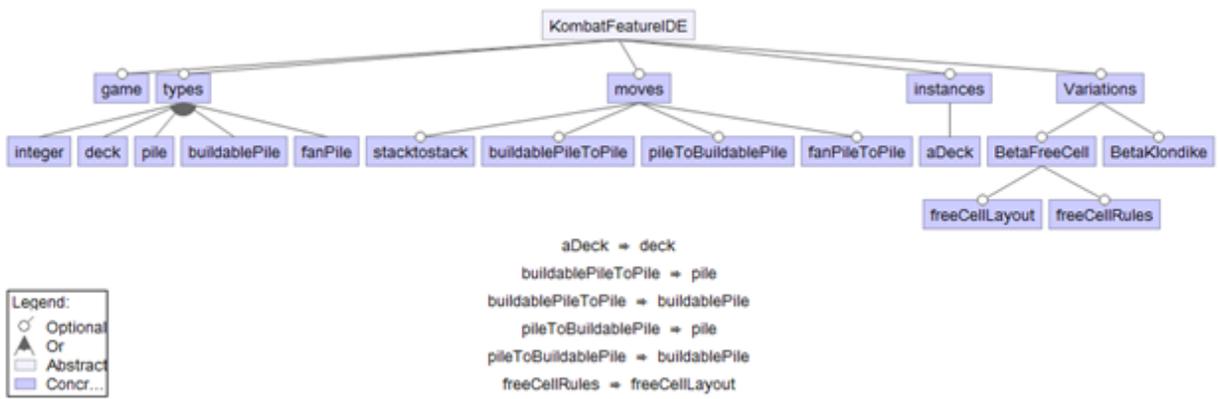
Figure 3.1. A feature diagram that captures Solitaire variations

In the figure above, a feature model that describes the variability of Kombat solitaire game. The variants of KombatSolitaire can be produced in this domain. The root of the tree denotes the concept that is modeled. All other boxes denotes features, where child features depend on parent features.

Table 3.1 below compares the reusability factor for generated layers of four solitaire plugin components against their hand-coded counterparts.

Table 3.1. Reusability Comparison

	Java	ACDK
	#Classes (#reused)	# Layers (#reused) %
Idiot	6 (0)	16 (13) 81%
Narcotic	7 (0)	17 (13) 76%
GrandFatherClock	6 (0)	31 (29) 93%
Klondike	11 (0)	31 (25) 80%

3.1. KombatSolitaire Design

Kombat Solitaire (KS) is a Java application that enables head-to-head competition of solitaire variations played simultaneously over the Internet. KS was developed as part of an undergraduate software engineering course. Each plugin represents a single solitaire variation. A rich set of model elements are already provided, as shown in Table 3.2.

Table 3.2. Classes within KS model hierarchy

Card	Single Card in a deck
Stack	Abstract representation of cards in sequence from bottom to top
Pile	Stack whose topmost card is visible
Deck	A stack of 52 cards forming a playing deck
Column	Stack of cards that reveals cards below the topmost card
BuildablePile	Pile of face down cards on top of which a Column can be built (as in Klondike)

Each model element shown (except for the abstract *Stack* class) has a corresponding view element that depicts the model element within the solitaire playing field. Each KS plugin is responsible for constructing a model of the game, which may include a deck, columns where cards are stacked, a running score, and waste piles. The plugin then defines the views for these model elements over a 2-dimensional playing field such that no two views intersect each other. Finally, a controller is registered with each view to manage mouse events (press, release, and click) and perform moves as allowed by the solitaire variation. The sum total of all the controllers enforces the rules of a solitaire variation.

The KombatSolitaire case study presented here is a microcosm of the overall thesis effort. That is, this thesis will demonstrate how to convert ordinary Java code into a set of features that represent the same functionality. In a straight object-oriented implementation of a solitaire variation, a programmer would construct objects from pre-existing classes (such as found in Table 3.2) to implement the required variation. For example, assume a developer wanted to implement the FreeCell solitaire variation, shown below:

Figure 52. Sample KombatSolitaire Klondike Screenshot

In this variation, there are four Free Cells in the upper left corner which can store a single face up card. In the upper right corner are four Base Cells which show face up cards, but they can contain up to thirteen cards each (one per suit). At the bottom of the game are eight Buildable Piles showing the arrangement of face up cards at play. A snippet of code in such a stand-alone object-oriented implementation would look something like the following:

```
public void initialize (int seed) {
   deck = new Deck ("deck");
   deck.shuffle(seed);
   for (int i = 0; i < columns.length; i++) {
       columns[i] = new Column("col" + (i+1));
   }

   for (int i = 0 ; i < freeCells.length; i++) {
       freeCells[i] = new Pile ("free" + (i+1));
   }

   for (int i = 0 ; i < baseCells.length; i++) {
           baseCells[i] = new Pile ("base" + (i+1));
   }
  ...
}
```

That is, the program would first create the model elements forming the structure of the game. Then they would create view objects to visually place these model elements on the screen:

```
  public void initializeView () {
    CardImages ci = getCardImages();
    int cw = ci.getWidth();
    int ch = ci.getHeight();

    for (int i = 0; i < freeCellViews.length; i++) {
        freeCellViews[i] = new PileView (freeCells[i]);
        freeCellViews[i].setBounds (10+15*i+i*cw, 20, cw, ch);
    }

    for (int i = 0; i < baseCellViews.length; i++) {
       baseCellViews[i] = new PileView (baseCells[i]);
       baseCellViews[i].setBounds (125+15*i+(i+4)*cw, 20, cw, ch);
    }

   int colH = 13*ch; // allow up to 13 cards
   for (int i = 0; i < 8; i++) {
       columnViews[i] = new ColumnView (columns[i]);
       columnViews[i].setBounds (45+15*i+i*cw, 40 + ch, cw, colH);
```

```
    }
```

Once all view elements are created, then the user needs to register the appropriate AWT

Mouse Listeners to react to mouse events over these objects

```
public void initializeControllers() {
 for (int i = 0; i < 4; i++) {
  freeCellViews[i].setMouseAdapter(new FreeCellFreeCellController (
                                   this, freeCellViews[i]));
 }
 for (int i = 0; i < 4; i++) {
  baseCellViews[i].setMouseAdapter(new FreeCellBasePileController (
                                   this, baseCellViews[i]));
 }

 for (int i = 0; i < 8; i++) {
  columnViews[i].setMouseAdapter(new FreeCellColumnController (
                                 this, columnViews[i]));
 }
```

These Controller classes contain the real logic of the FreeCell variation. Indeed, the

structure of two solitaire variations may be identical (both model and view elements) and the

only difference remains in the Controller objects.

The controller classes process low-level MouseEvent events and decides users actions by

interpreting the sequence of MousePress, MouseDrag and MouseRelease events. The following

controller snippet for the FreeCell Game processes MouseRelease events over one of the four

base piles:

```
public void mouseReleased(MouseEvent me) {
   Container c = theGame.getContainer();

   Widget w = c.getActiveDraggingObject();
   if (w == Container.getNothingBeingDragged())
      return;

   boolean changed = false;
   if (w instanceof CardView) {
      changed = processDraggingCardView ((CardView) w);
   } else if (w instanceof ColumnView) {
```

```
        changed = processDraggingColumnView ((ColumnView) w);
    }

    // try auto moves
    if (changed) {
        ((FreeCell)theGame).tryAutoMoves();
    }

    // release the dragging object and repaint everything
    c.releaseDraggingObject();
    c.repaint();
}
```

The reason for inserting this code here is to demonstrate the intertwined logic of these controllers. This one, for example, is able to process cards being dragged to the base pile from a free cell or one of the buildable piles. It also automatically advances any automatic moves that the game of FreeCell can determine to execute. We are omitting the additional detail found in the processDraggingCardView and processDraggingColumnView methods.

In poorly designed code, the Controller logic is intertwined with the Model and View classes, which reduces their reusability and overall coherence. However, when they are cleanly separated, the Controller classes quickly become overly detailed and complicated.

In Java – indeed in any object-oriented language – the unit of composition is an object (which really means the class used to define the object). Because the controllers contain the true logic, they often are not reusable themselves, as Heineman observed [9]. The irony is that the reusable model and view widgets have none of the Solitaire behaviors associated with them, whereas the non-reusable controllers contain all of the "important" code for the solitaire variation.

3.2. KombatSolitaire Feature-Based Design

Heineman observed that using MVC naturally leads to the inability to reuse controllers [9]. Domain experts have considerable expertise in using inheritance to capture the rich information to be stored in a model. HCI experts show how to build user interfaces that decouple the model from the view presented to the users. But the complex logic found in controllers can quickly be unmanageable because of the inherent limitations of the basic extension constructs in OO programming languages. Since business logic is encapsulated within controllers, MVC may actually be an impediment to the proper reuse or extension of business logic. For this reason, Heineman investigated how to convert the KombatSolitaire code base into features, using Batory's AHEAD tool suite. Ultimately the final system was realized using the FeatureIDE Eclipse Plugin.

The layers designed for this solution are intended to be the unit of composition, rather than individual classes. To achieve this goal, the premise is that as each additional layer is composed, there will always be a working solitaire implementation, albeit one with reduced functionality.

To represent a composition of layers, we use the dot ● notation. Starting from the **game** layer, one constructs a solitaire variation by the repeated composition of additional layers as designed in the solitaire solution space. In general, a solitaire variation is defined by equations of the following form, using features defined earlier.

variation = {**Variations**} ● {**moves**} ● {**types**} ● **game**

Let's propose the following as the simplest Solitaire variation.

There is a deck of cards and two piles. To deal a card from the deck to the empty

waste pile, click on the deck. Players can drag the card from the waste pile to the

home pile. Once all cards have been played, the game is over.

To assemble this variation using Features, you use existing layers and write three additional layers to describe the layout and the rules for this variation. Here is the final equation

simplest = { **SimplestRules**, **SimplestLayout**, **Simplest**, **Variations**} ● {**deckToStack**, **stacktostack**, **moves**} ● {**pile**, **deck**, **integer**, **types**} ● **game**

Most of these features are pre-existing and exist to support other solitaire variations as well. This exercise will demonstrate that it is possible to assemble an implementation of a solitaire variation predominantly from existing Feature layers.

The design of these layers support the model-view-controller (MVC) paradigm inherent in the underlying Java object-oriented implementation, however the true novelty appears when expanding (or contracting) the basic elements in a solitaire variation.

The best way to explain the logic is to show the full details for one of the layers, in this case, the **pile** layer, which contains the following JAK artifacts:

- **class** FlipPile **extends** Move
- refines **class** Game
- **class** PileAdapter **extends** MouseAdapter
- **class** PileManager
- **class** PileToPile **extends** MoveCardMove

The **pile** layer refines an existing class (Game) and introduces four new classes to deal with the behaviors associated with Solitaire piles. Specifically, you can view only the top card in a pile, although it may contain any number of cards. You can interact with a pile by removing its top card (pressing the left mouse button) or releasing a card (or a column of cards) onto the pile (by releasing the left mouse button over the pile).

The *PileAdapter* class extends *MouseAdapter* which allows it to be a drop-in replacement for any *MouseListener* interface.

<< heineman adds more detail on Simplest>>

Chapter 4. Methodology

Like in the construction industry, you would have to know what you are going to build before you can build it. So the first step in methodology is what software will be integrated and figure out integration method. We need to integrate between ASSISTments with other Learning Management Systems (LMSs) to utilize research in educational data mining and to share content across educational community. The contents of LMSs can be accessed immediately by all users (teachers, student, parents and administrators etc.) – all applications appear in one system, through a seamless online environment with a single sign-on learning portal.

The goal or scope of the thesis is to implement a system based layers that generates connector code for each of the system need to be integrated with ASSISTments.

4.1. Our Approach

K.Lee et al, 2000 proposed Feature-Based Object-Oriented Engineering, or Feature-Oriented Reuse Method (FORM), which instead identifying objects by popular method such as Keyword analysis [10][11], structured analysis [12], scenario-based analysis [5][6], but identify reusable objects by linking feature categories to object categories. FORM could extract important relationships between objects (aggregation and generalization) from feature model (composed-of, generalization, and implemented by). This leads FORM method favor object composition than class inheritance when design and development of reusable components (e.g., modules).
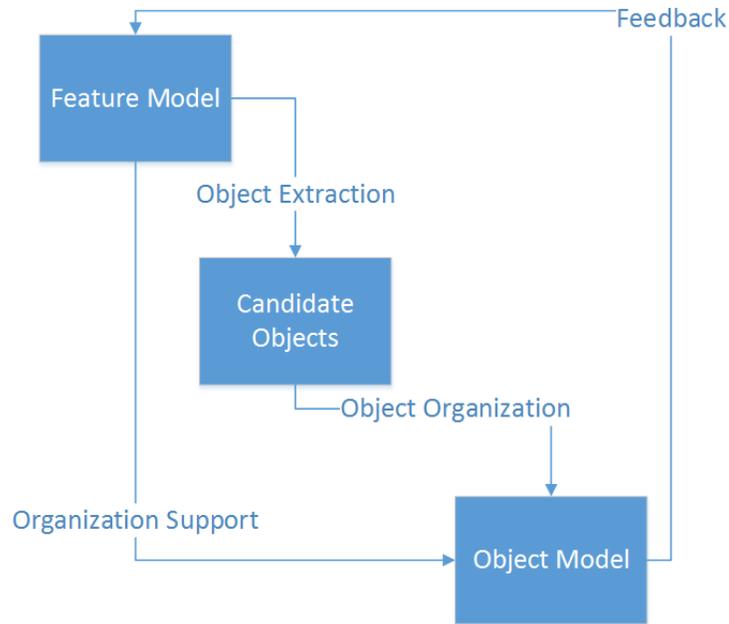
Figure 4.16. FORM Methodology

Our approach, however, does not start with feature model. What we believe is that even objected that properly abstracted and modeled for future reuses more likely subject to change than functions [15][16].
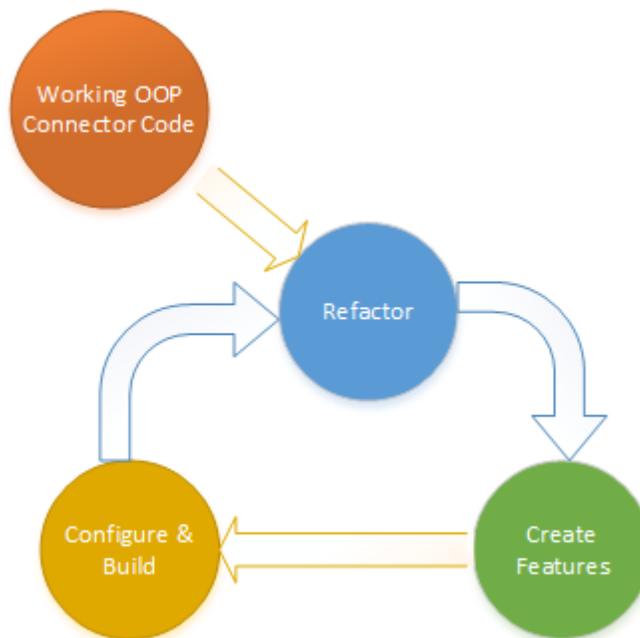

Figure 4.2. Our Methodology

To capture all the variants of each connector code, before going into integration layers design step, we programmed a prototype of integration between ASSISTments and inBloom. After that, there is one prototype need to be integrated with ASSISTments including: Edmodo. All two systems are what client requests to see the integration works. Then we can gather more functional requirements from the client based upon prototype system. All of the prototypes be programmed in Java.

After capturing all variants of integration between systems, but before writing any layer code, we first design integration based layers. Those layers could be assembled to satisfy all functional specification that describes in detail the functions to perform by the system. Moreover, those layers should be assembled easily enough to generate connector code.

Once all appropriate layers and specifications have been determined, the development of system based layers is started. During the development process, we perform unit test to verify new layers system reliability by comparing the result of integration with each prototype system. Finally, we can test the entire system by generating separate connector code for not only all inBloom and Edmodo, but also any system that desire.

4.2. Connection Model

Integration between two systems, even they are built for similar purpose, can be very complex process, since they can be very different in nature.

Each Learning Management System (LMS) provides its own interface, which is defined by an API (Application Programming Interface), to enable it to communicate with other systems according to a particular set of rules. Because of each own particular API, you cannot immediately "plug and play" one system to communicate with others; we need connector code. Figure 1 below is the flowchart describes connecting between systems:
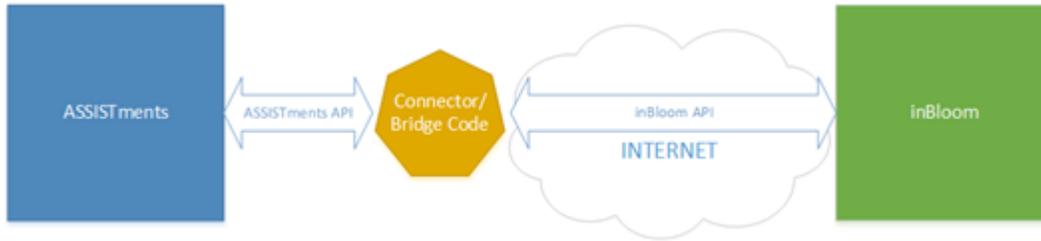
Figure 4.3. Sample integration between LMSs

The connector code above is specialized code that adapts to particular rules of both ASSISTments and inBloom to make them work together. When ASSISTments wants to connect with another LMS, Edmodo for example, this connection again requires writing special purpose connector code between two systems with independent APIs. Much of the effort in writing these connectors will be wasted because of the way that the object oriented code has to be written. The primary issue is the lack of modularity in object-oriented design patterns [1]. When code implements an interface, the internal (almost arbitrary) code written cannot be used and extended; rather this leads to copy/paste style reuse when attempting to bridge to multiple systems.

Figure 4.4 shows an example of creating grades using ASSISTments API with inBloom and Edmodo. Consider how we would export grades out of ASSISTments. In this figure, we shows the schema differences in the APIs of inBloom and Edmodo. Some of this functionality can be shared but there are noticeable differences.
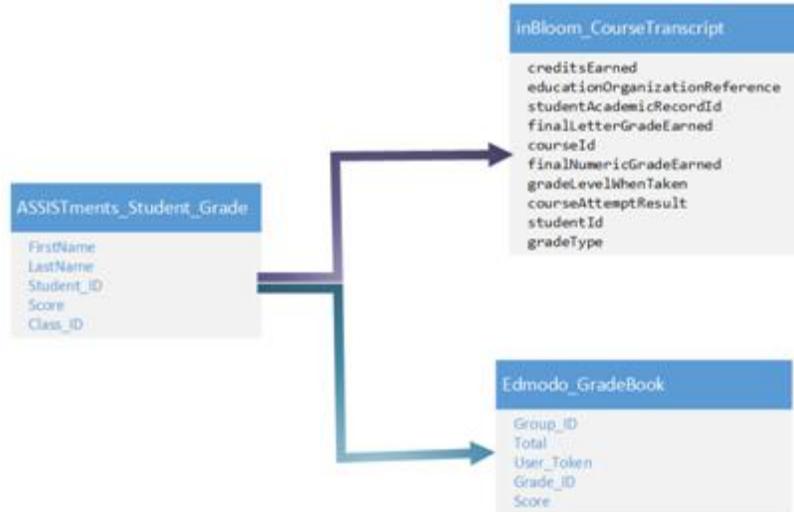
Figure 4.47. Multiple instances of connector code adapt to each system

The premise of this thesis is that we need to properly engineering "glue" or bridge code, and existing languages do little to help the reuse problem in this domain. Rather we must turn to a model that allows code to be woven together to achieve reuse. As shown in the figure above, the processing of extract data for sending and receiving is similar but data fields are different. These differences lead to diverse behaviors in object-oriented design. Recently, in Software Engineering, there are advanced programming techniques gain momentum to encapsulate variability such as Feature-Oriented Programming (FOP) or Aspected-Oriented Programming [2][8]. In this thesis work, we propose to design features that are not rigidly based upon class structure, and can be composed appropriately to create different connecter code as desire.

Chapter 5. Connector Code

5.1. Overview integration between ASSISTments and inBloom

Figure 5.1 is the scenario promoted for integrating between ASSISTments and inBloom. This has been coded and tested to work successfully in connecting two systems.

Two big rectangles represent two systems needed to be connected. Left hand side is inBloom and right hand side is ASSISTments. Each rectangle inside is an operation which triggered by the connector code. The arrows show data exchanged between two systems and managed by connector code. The down arrow in each system shows the previous step needed to be completed in order to continue. In this diagram, some required bean classes of connector code map to object of two systems not to be listed here. Those bean classes are required to build, serialize objects to *json* data and deserialize *json* data to objects. Example of the bean classes will be provided in later part of the thesis.

In this scenario, the upload back students grade into inBloom is optional. For the current release of connector code, ASSISTments does not support API to get student grades. The code of submitting back data to inBloom is done by screening ASSISTments HTML report, based upon some predefined text to obtain student performance result. Crawling large text file to extract desire data is not an efficient method.

Figure 5.1. Screening ASSISTments HTML Report
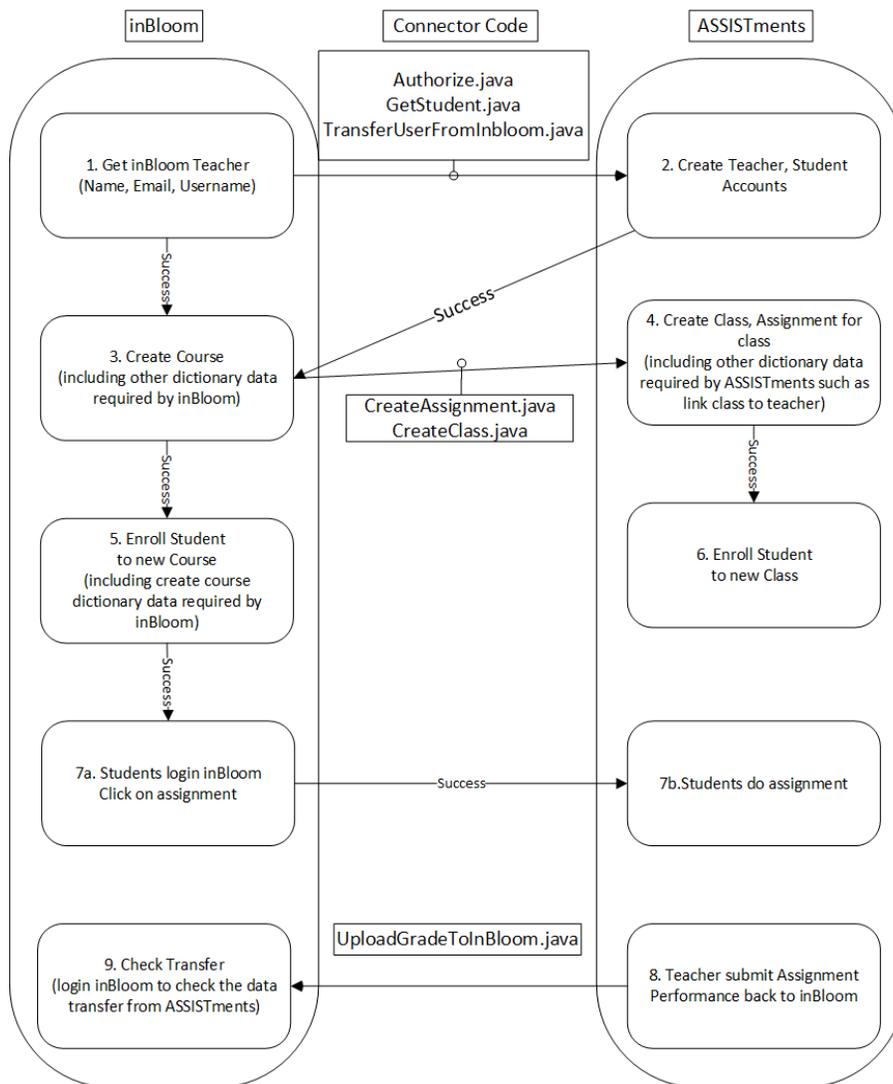
Figure 5.2. Promoted integration scenario with inBloom

5.2. Overview integration between ASSISTments and Edmodo

Below is the high level view of integration between ASSISTments and Edmodo. The scenario is very similar to inBloom. Then naturally, the class structure is similar but the code inside is different in some senses. For example, with inBloom, Authorize.java have to deal with OAuth2 security while with Edmodo is OAuth.
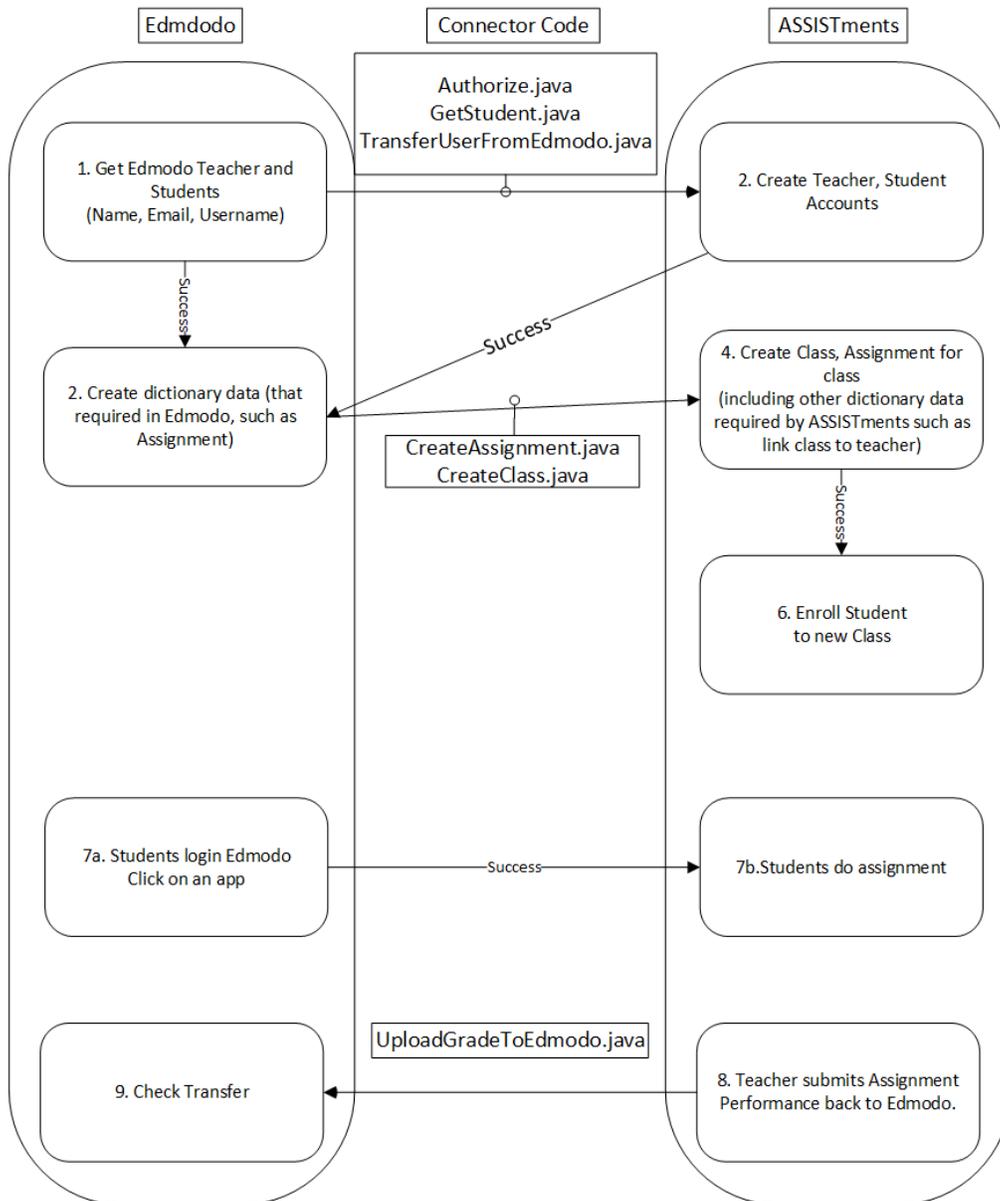
Figure 5.3. Promoted integration scenario with Edmodo

5.3. Detailed Integration Scenario with Edmodo

With all the scenarios below, the integration process is activated by Edmodo site. It means teachers and students have to have accounts in Edmodo site first.
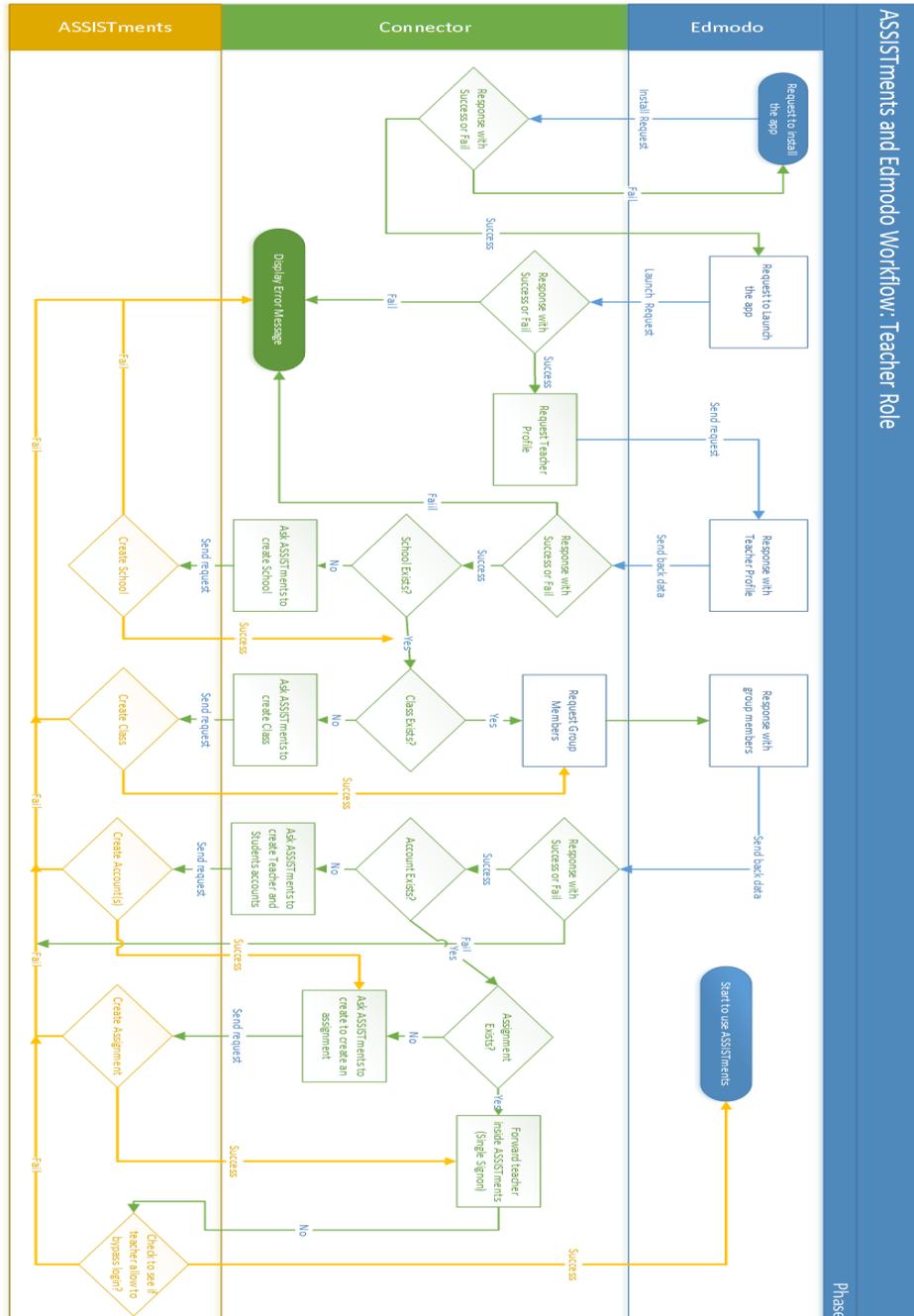
Figure 5.4. Edmodo Integration Flowchart

First of all, teachers in Edmodo would know about ASSISTments. They could search in Edmodo Store to install the app from ASSISTments.
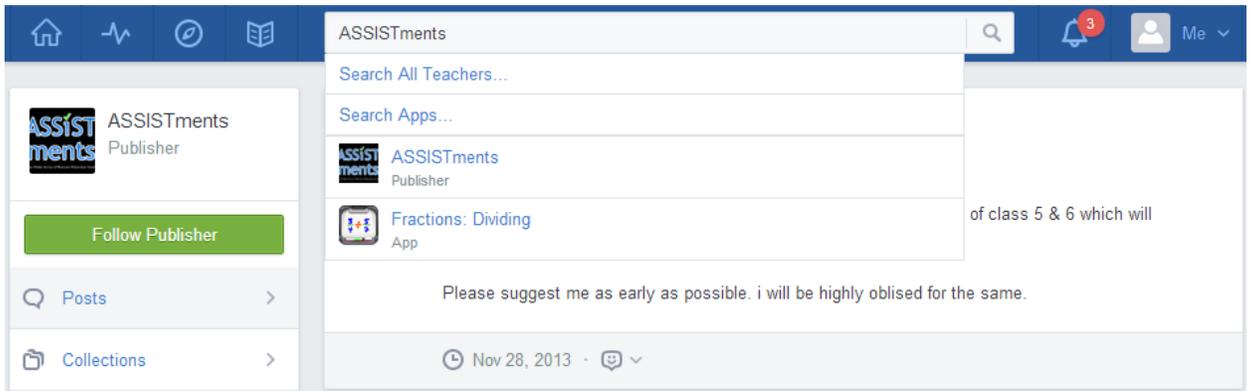
Figure 5.5. Teacher search for app
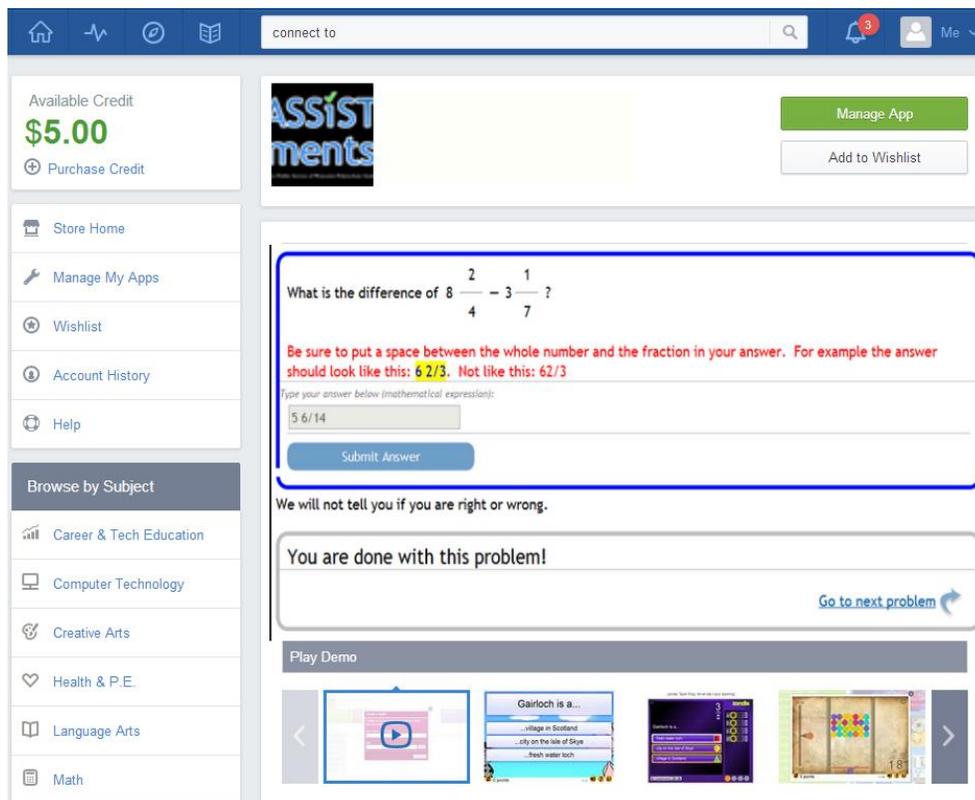
Figure 5.6. Teacher installs the app

The teacher then makes decision to install the app. Teacher has to pay the fee if needed. In case the app is free, teacher can immediately chooses which groups of students the app applied to. And the app then automatically appears in students view.
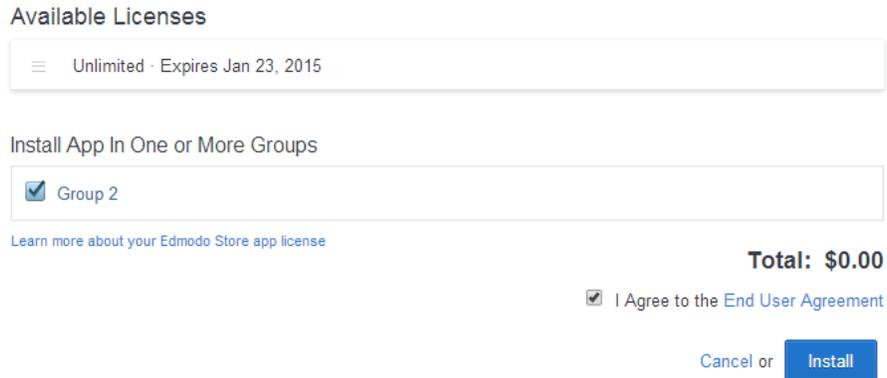
Figure 5.7. License and Group of students the app will be applied to

Please notice that each ASSISTments app is one problem set. So readers can assume that there would be hundreds of ASSISTments app in Edmodo. If the teacher first installs ASSISTments app, via Edmodo API, below are what information we can get:

Teacher	*user_token: unique number that identify user in the system.* *first_name* *last_name* *user_type: TEACHER* *time_zone* *groups: list of group that teacher are the owner.* *access_token: Authorizes and authenticates user login and permissions.*
Student	*user_type: STUDENT* *user_token* *first_name* *last_name*

Table 5.1. Example API return.

Based on those information get from Edmodo, ASSISTments automatically does following steps:

1. Create principal account for teacher with assume that:

   + login name: user_token@edmodo.com (538237@edmodo.com)

   + email: login name

   + First name: first_name

   + Last name: last_name

   + Password: randomize (then would reset later if teacher provides the real email address)

   + display name: first_name last_name

2. Create proxy account for students:

   + Username: first_namelast_name. Notice that student never have to login ASSISTments so Username just for tracing purpose.

   + First name: first_name

   + Last name: last_name

   + display name: first_name last_name

3. Automatically enroll teacher and students into "Edmodo School" (with assume that this school is created beforehand).

4. Automatically using teacher permission to create class. Class name is the combination of teacher name, skill builder name.

5. Enroll students in class.

6. Automatically create a new assignment and assigns to students.

   Now both students and teacher from Edmodo can be navigate to ASSISTments without any login required from ASSISTments.
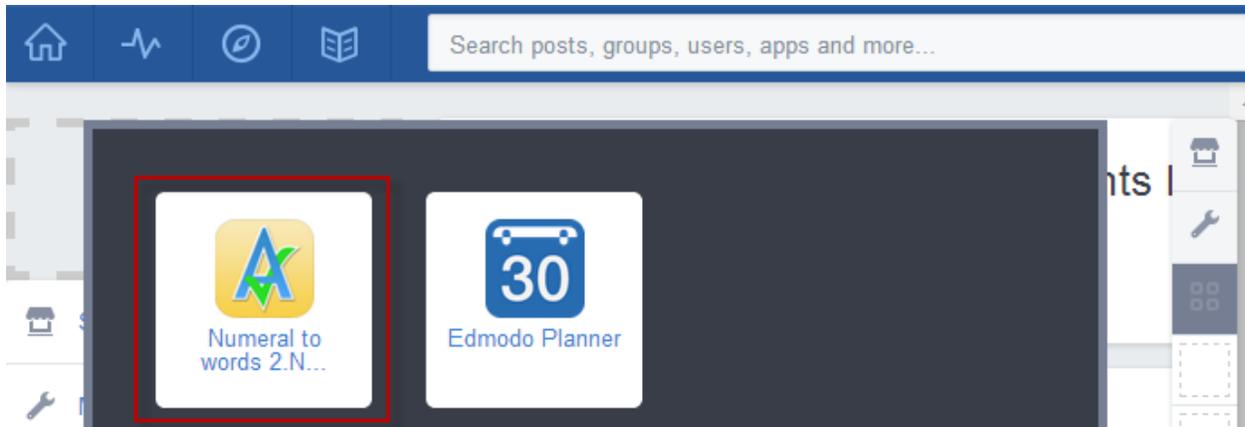
Figure 5.8. Teacher launches ASSISTment app. He/she automatically forward to ASSISTments without prompting ask for credentials

In case of students:
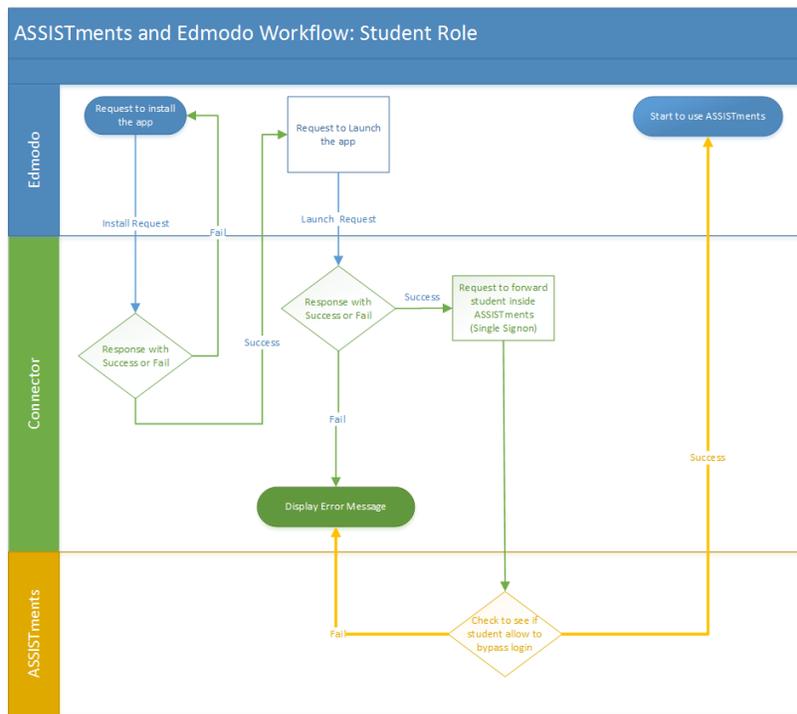
Figure 5.9. Integration detail steps between ASSISTments and Edmodo under Student Role

In both cases, if teachers or students launch the app successfully, they will be forwarded inside ASSISTments, which is embeded as an iFrame inside Edmodo.
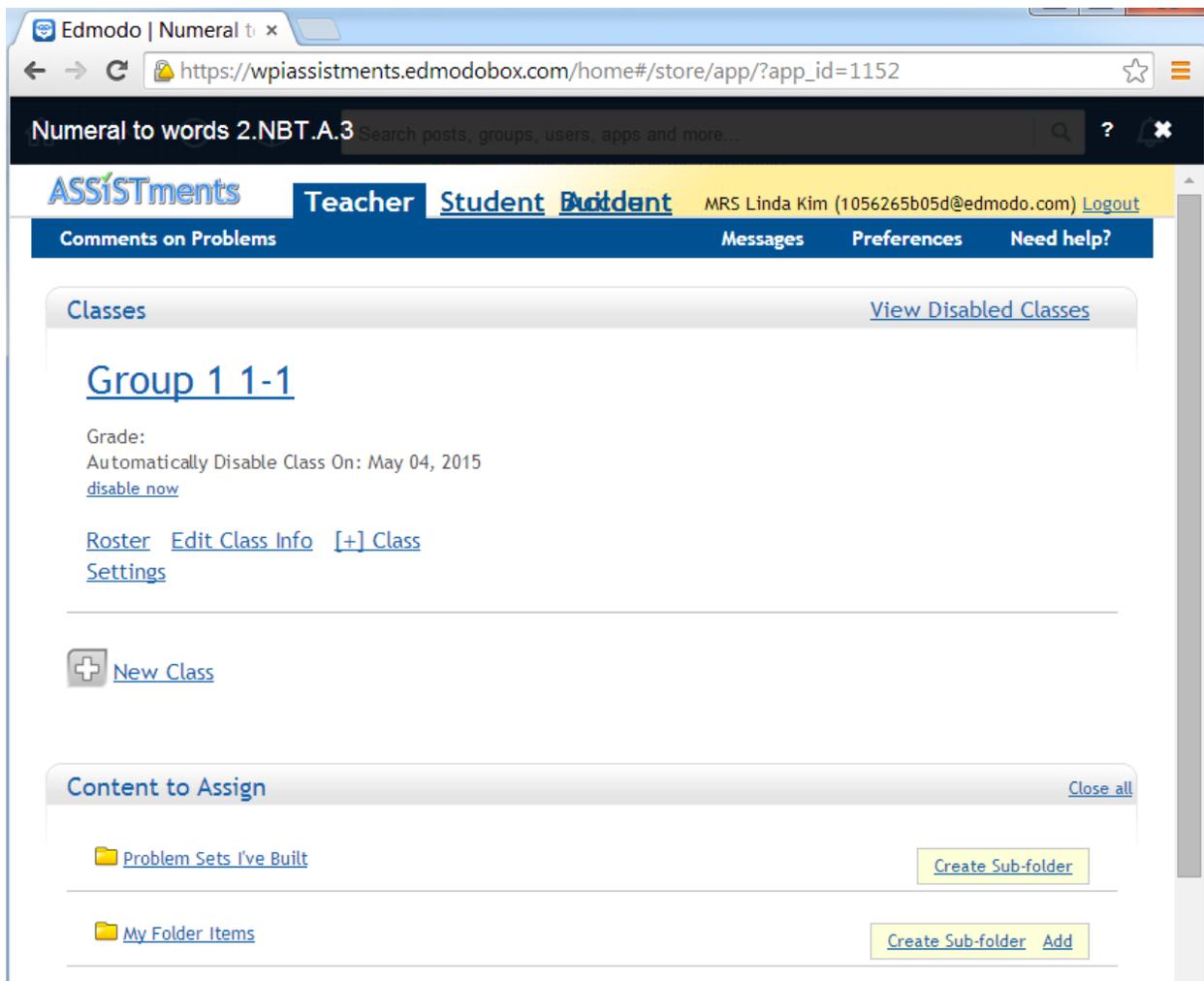
Figure 5.10. ASSISTments appears inside an iFrame in Edmodo

5.4. Object Oriented design and implementation

The bridge code is actually a servlet runs on Apache Tomcat server and waits for . It handles requests from source system, manipulate them to adapt to target system's API, sends request to target system, receives respond from target system, manipulate the respond and forward respond to source system.
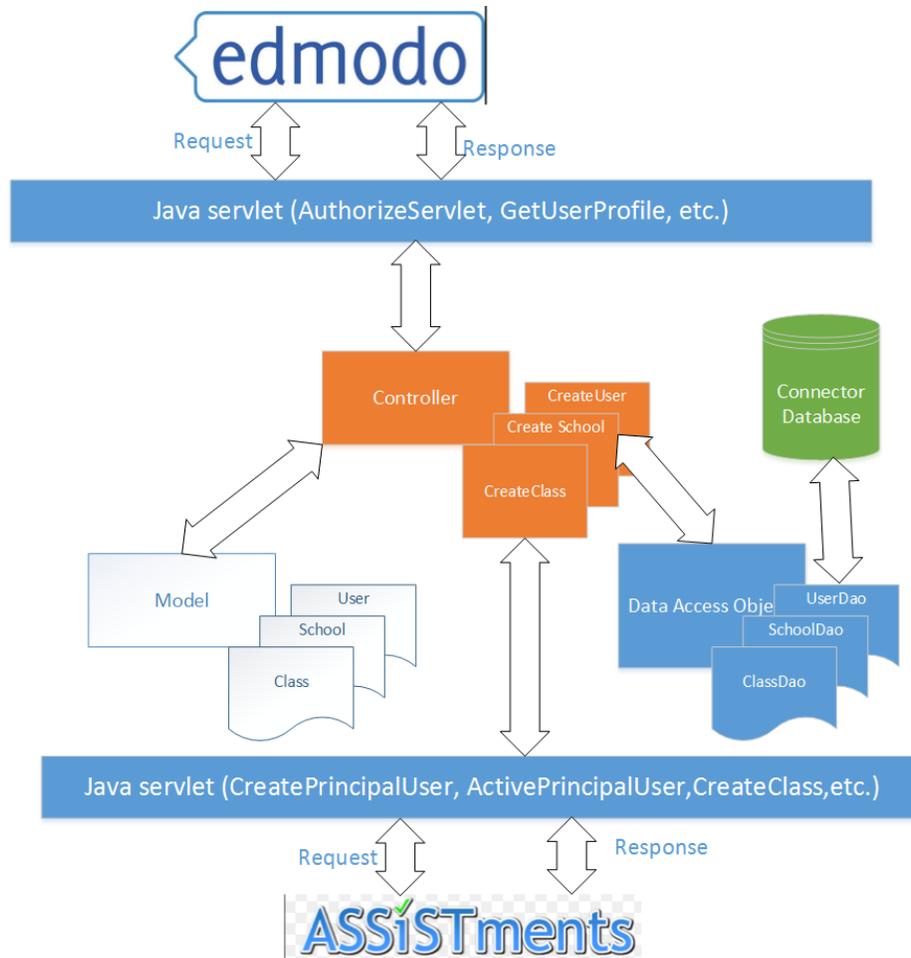
Figure 5.11. High level connector design

When we successfully did object-oriented approach for bridging ASSISTments with both inBloom and Edmodo, there are the structure of code in both package are very similar. Below is the packages in two solutions:

Table 5.2. Connector code packages listed

inBloom	Edmodo
assist.bean	assist.bean
assist.job	assist.job
global	global

inbloom.bean	edmodo.bean
inbloomJob	edmodoJob
inbloomLogin	edmodoLogin
utilities	utilities

Details of these package classes are listed in an Appendix C.

In the context of functionality, we can see the similar in code structure in both solution. Both solutions have 7 packages and each package pair inline exposed very similar functionalities.

The assist.bean package contains data objects that map to API payload of ASSISTments. It also contains Serializer object responsible for serial data object into Json format. And assist.bean also contains the data access object that map to data fields in database.

This Json object then be transferred between system via Internet.

```
package assist.bean;

public class Class {
      private String courseName;
      private String courseNumber;
      private String sectionNumber;
      public String getCourseName() {
            return courseName;
      }
      public void setCourseName(String courseName) {
            this.courseName = courseName;
      }
      public String getCourseNumber() {
            return courseNumber;
      }
      public void setCourseNumber(String courseNumber) {
            this.courseNumber = courseNumber;
      }
      public String getSectionNumber() {
            return sectionNumber;
      }
      public void setSectionNumber(String sectionNumber) {
            this.sectionNumber = sectionNumber;
      }
```

```
}
```

Figure 5.12. Sample code to present json object

```
public class ClassSerializer  implements JsonSerializer<Class> {

      @Override
      public JsonElement serialize(Class classObj, Type type,
                  JsonSerializationContext context) {

          final JsonObject jsonObject = new JsonObject();
          jsonObject.addProperty("courseName",
                        classObj.getCourseName());
          jsonObject.addProperty("courseNumber",
                        classObj.getCourseNumber());
          jsonObject.addProperty("sectionNumber",
                        classObj.getSectionNumber());
          return jsonObject;
      }
}
```

Figure 5.13. Sample code to serialize an json object

```
public class ClassDaoBean  implements Serializable {
      private String partner_refernce;
      private int external_refernce_type_id;
      private String external_refernce;
      private String user_access_token;
      private String partner_external_reference;
      private String user_connector_token;
      private String note;

      public String getPartner_refernce() {
          return partner_refernce;
      }
      public void setPartner_refernce(String partner_refernce) {
          this.partner_refernce = partner_refernce;
      }
      public int getExternal_refernce_type_id() {
          return external_refernce_type_id;
      }
      public void setExternal_refernce_type_id(int
                    external_refernce_type_id) {
          this.external_refernce_type_id = external_refernce_type_id;
      }
      public String getExternal_refernce() {
          return external_refernce;
      }
        (...)
}
```

Figure 5.14. Sample code to represent data access object

In the assist.job package, all classes are servlet and responsible for sending GET request to ASSISTments by using ASSISTments API. Each servlet is invoked automatically if dictionary data need to be created in ASSISTments side or manually if users trigger an event in external system side.

Invoked automatically if a new teacher from a new system wants to use ASSSITments. The bridge code will ask ASSISTments to automatically teacher into school without teacher notice.

Invoked manually if teacher requests such as he/she wants to create assignment in ASSISTments. Then the bridge code will ask ASSISTments to do so.

Figure 5.15. Example code invoked automatically.

```
String nces = ApplicationSettings.SchoolNCES;

School school = new School();
school.setNces(nces);

GsonBuilder gsonBuilder = new GsonBuilder();
gsonBuilder.registerTypeAdapter(School.class, new SchoolSerializer());
Gson gson = gsonBuilder.create();

String payloadJson = gson.toJson(school);
(...)
String resJson = ASSISTAPIUtilities.getJSONNotBehalf(APISchool,
payloadJson);

Gson gsonSchoolRef = new GsonBuilder().create();
SchoolRef schoolRef = gsonSchoolRef.fromJson(resJson, SchoolRef.class);
```

Global package

This package contains configuration information for the bridge code to work. Those global settings are used everywhere in the bridge code and convenient to access. The information such as ASSISTments API URLs are stored here.

External system bean

The external system beans, such as edmodo.bean or inbloom.bean, are very similar to assist.bean in structure of code. But their functionalities are map external system API object. The difference is all of the objects are transaction objects because dictionary data already exist in external system. We should notice again that external system is the trigger entity. And all the dictionary data needed to be set up beforehand to allow trigger to be fired from external system.

External system job

Similar to assist.job package, all classes under inbloom.job or edmodo.job are servlets and controllers. Those servlets receive requests, initiate correspond controllers then and waiting controllers to be invoked. Transactions are invoked manually with users notice or automatically. Example of invoked automatically such as connector code wants to get more detail information of user by sending extra API request to Edmodo.

External system login

The login functionality is grouped in separate package because of its natural complexity. Each external system has its own authentication and authorization methods to validate users. For example, with inBloom, it uses oAuth2 and with Edmodo it uses oAuth1a.

Utilities package

This package has classes responsible for constructing API requests to ASSISTments or Edmodo. Methods in those classes are static and are invoked by controllers when needed. It also handle the response returned. And when it finishes, it will return control to controllers with status of response is valid or exception occurs to let controllers what need to be done next.

```
public static String sendURLGet(String fullURL) {

        BufferedReader reader = null;
        String checkResponse = "";
```

```
            try {

                    URL url = new URL(fullURL);
                    HttpURLConnection connection = (HttpURLConnection) url
                                .openConnection();

                    connection.setRequestMethod("GET");

                    BufferedReader in = new BufferedReader(new
                    InputStreamReader(
                                connection.getInputStream()));

                    ApplicationSettings.setErrorStatusCode(
                    connection.getResponseCode());

                    String inputLine;
                    StringBuffer response = new StringBuffer();
                    while ((inputLine = in.readLine()) != null) {
                          response.append(inputLine);
                    }
                    in.close();
                    checkResponse = response.toString();

            } catch (Exception e) {
                    System.out.println("An error might occur with sendURLGet:
                    " + fullURL);
                    e.printStackTrace();
                    ApplicationSettings.setErrorTitle("Error while
                    communicating with Edmodo. We are sorry cannot proceed
                    further!");
                    ApplicationSettings.setErrorDetail("An error
                    might occur with sendURLGet: " + fullURL);
                    return null;
            }
            return checkResponse;
      }
}
```

Figure 5.16. Sample code to send API request.

```
public String getProfile(ArrayList<String> userTokens) {

            String request = "";
            setResources("profiles");

            Gson gson = new Gson();
            String userTokensJson = gson.toJson(userTokens);

            request = "/" + getResources() + "?" + "api_key=" +
getApi_key()+"&access_token="+getAccess_token()+"&user_tokens="+userTokensJs
on;
            request = ApplicationSettings.EdmodoAPIBase + request;
            return request;
```

```
        }
}
```

Figure 5.17. Sample code to construct API request.

Chapter 6. Featured-based Approach

P. Elizondo and K-K Lau proposes different approach than direct and indirect message connector approach [27]. As illustrated in the figure below, components do not call methods in other components. Instead, all method calls are initiated and coordinated by the connectors. The round dots denote the origins of the communication and coordination.
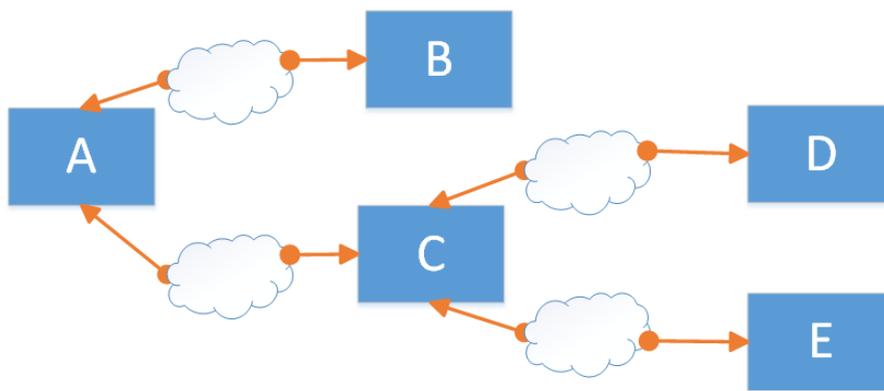
Figure 6.1. Proposed approach for connectors

This is clear contrast to both direct and indirect messaging technology since components originate communication and coordination. And they are convinced that their connector types support reusable not only at design but also at implementation phase.

And we found that our connector works as similar as Elizondo and Lau suggested. As describe before in connection workflow in Figure 5.4, users from Edmodo initiate the process then that is all. Subsequence steps are to be performed by the connector. The connector invokes ASSISTments or Edmodo services when it needs.

6.1. Refactoring OOP to extract features

A feature is a unit of functionality of a software system that satisfies a requirement. Therefore in attempt to refactor already working code, we move all the functionalities into features. However, we still have to keep the service handler in the object-oriented project. This is because for the time being, FeatureIDE cannot create a dynamic website project. So the OOP dynamic web application will delegate its work to correspond feature.

```
@WebServlet("/CreateAssignment")
public class CreateAssignment extends HttpServlet {
private features.CreateAssignment f_createAssignment = new
                                    features.CreateAssignment();

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException,
            IOException {
            f_createAssignment.doGet(request,response);
      }
}
```

Figure 6.2. OOP dynamic website invokes a feature

```
public class CreateAssignment {
      public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,IOException {
                    ….
}
```

Below is the model of Edmodo feature for the first attempt:
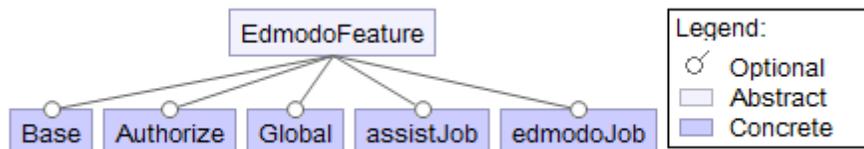
Figure 6.3. Edmodo Feature

6.2. Layers design and implementation

A layer is a set of files that define a feature of an application. Code representations are expressed as .jak files. For convenience, we list the integration scenario of inBloom and ASSISTments below for reference. In this scenario, we remove all the java classes represent controllers in OOP approach. We replace those controllers by layer classes. Detail of the designation is listed below.

And because of similar features design and implementation between feature-based connector between inBloom and Edmodo, we are going to list features of Edmodo connector code.
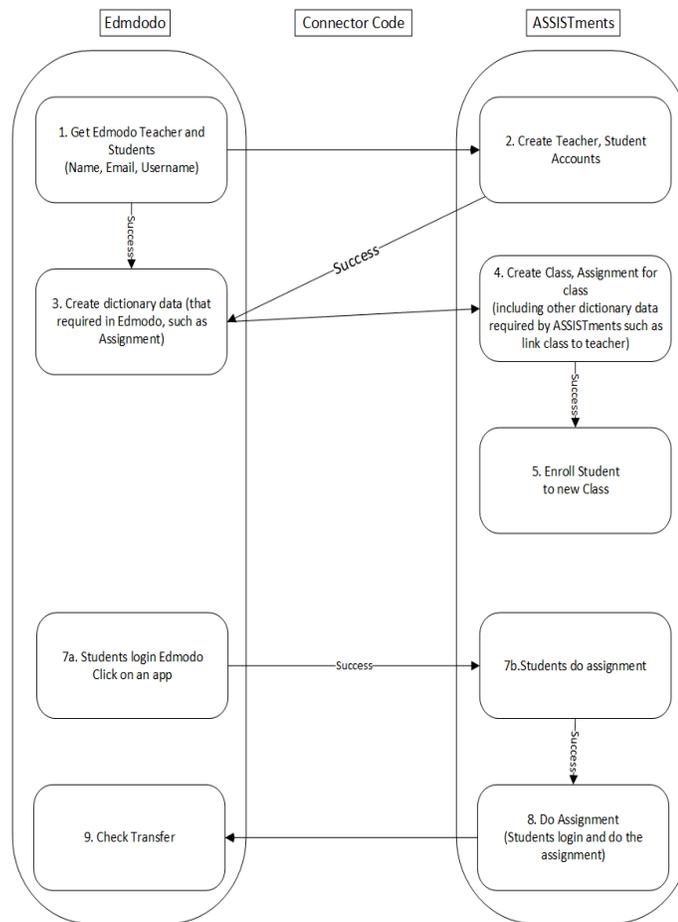
Figure 6.4. Edmodo Integration

6.2.1. Get Edmodo Teacher and Students

Feature: Authorize.jak.

Initiate entity: Edmodo users

Functionalities: An Edmodo user clicks one ASSISTments app. This app automatically send an API call to connector code requesting to launch ASSISTments app. Connector code handles the request but there is some security check happen there. The connector code has to make sure who is making the request by checking known api_key.

Connector code then has to check whether or not it is the user first launch ASSISTments app.

If it is the first launch then it has to save user information such as user_token from Edmodo into connector database and then automatically request ASSISTments to create user account. If a user is a teacher then students accounts in his group also created.

If users already have accounts then he/she is forward directly inside ASSISTments.

6.2.2. Create Teachers and Students accounts in ASSISTments

Feature: TransferUserFromEdmodo.jak.

Initiate entity: Automatically invoked after Authorize.jak

Functionalities: Right after authorize and authenticate users from Edmodo, connector sends request to create user accounts in ASSISTments. Normally, accounts will be created in ASSISTments. Teacher will be a principal account and student will be a proxy account. If there is something wrong with the process then Edmodo user will be saved in connector database without creating a new users in ASSISTments.

6.2.3. Create dictionary data in Edmodo

Feature: CreateAssignment.jak.

Initiate entity: Automatically invoked when TransferUserFromEdmodo.jak finishes.

Functionalities: This will be invokded when ASSISTments allows teacher to create an Edmodo assignment. This function maybe useful when teacher wants to transfer students performance from ASSISTments back to Edmodo.
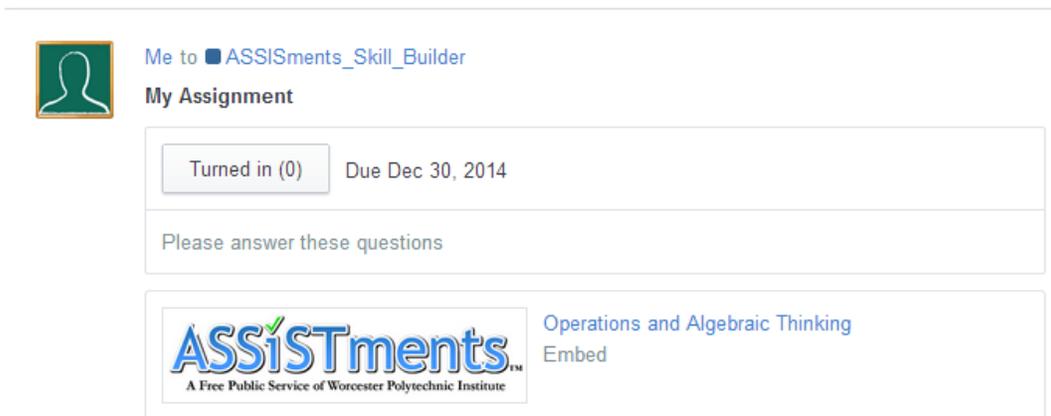
Figure 6.58. ASSISTments final screen

6.2.4. Create school data in ASSISTments

Feature: CreateSchool.jak

Initiate entity: Invoked automatically by connector after successfully create user accounts in ASSISTments.

Functionalities: In order for users do his/her normal job in ASSISTments, ASSISTments requires he/she needs to be assigned to a school. If user from Edmodo has valid school NCES code then new school automatically created if this school is not existed before. Otherwise, teacher and his students are in "Edmodo School".

6.2.5. Assign users to school.

Feature: AssignUserSchool.jak

Initiate entity: Invoked automatically by connector after school created in ASSISTments

Functionalities: Schools are required for every users in ASSISTments. This feature does assign new users to a school.

6.2.6. Create class

Feature: CreateClass.jak

Initiate entity: Invoked automatically by connector after assigning users to school in ASSISTments

Functionalities: Teacher needs class to organize his/her students. ASSISTments requires teacher access_token to behaves on his/her behalf to create class. One group in Edmodo correspond to one and only one class in ASSISTments. Number of groups in Edmodo will be created exactly the same number of classes in ASSISTments.

6.2.7. Enroll students in class in ASSISTments

Feature: EnrollStudentClass.jak

Initiate entity: Invoked automatically by connector after creating classes.

Functionalities: Students with the same group in Edmodo will be automatically enrolled into the same class in ASSISTments.

6.2.8. Create assignment in ASSISTments

Feature: CreateAssignmentASSIST.jak

Initiate entity: Invoked automatically by connector after assign users to school in ASSISTments

Functionalities: Connector code automatically send REST API to ASSISTments to request for creating an assignment in ASSISTments. Each ASSISTments app in Edmodo is one

assignment in ASSISTments. With this, students from Edmodo can login and do the assignment in ASSISTments.

6.2.9. Student login Edmodo to do assignment in ASSISTments.

Feature: Authorize.jak

Initiate entity: Students in Edmodo

Functionalities: Students from Edmodo click on an ASSISTments app and they are able to forward inside ASSISTments to do the assignment.

6.2.10. Transfer back assignment grade back to Edmodo

Feature: ScreenHTMLReport.jak and UploadGradeToEdmodo.jak

Initiate entity: Teachers in Edmodo

Functionalities: This functionality is optional for a teacher in ASSISTments when they want to send back the grade to Edmodo. First of all, the connector code will crawl generated HTML report in ASSISTments, reads each student performance. Then connector code sends request back to Edmodo to update or create new grade there.

Chapter 7. Conclusions and Future Work

7.1. Conclusion

In this thesis, we put forward the idea of using Feature-Oriented Programming technique in practice by developing connectors between systems. In the first part, we introduce the basic concepts of FOP with examples from academics simple program HelloWord to Linux configuration tool, an industrial complex program. Then we go more in-depth details on designing and implementing the connector code by using Objected Oriented Programming technique. This step is important since FOP is an extension of OOP. The successful of the OOP solution is illustrated by having many ASSISTments apps reviewed by Edmodo development teams and appears in Edmodo Store.

By having working OOP connector code, we can capture all the variations of each working solution between ASSISTments and inBloom, and ASSISTments and Edmodo. Then we refactor OOP connector code to develop working connector solution in FOP. However, during the time of the thesis work, FeatureIDE does not support for creating dynamics web project. So we have no choice to keep the servlet class structure from OOP and apply to FOP. Most of effort reduced is in development of controller classes in FOP ASSISTments side.

The main contribution work of this thesis is to represent a case study to integrate systems using FOP technique. There are no literature on integrating system using FOP but similar concepts in Component-based Software Engineering. Based upon this thesis work, we conclude that using FOP technique is totally possible and very promising on integrating systems.

7.2. Future Work

FeatureIDE currently one of the best academics tool supports for Feature-Oriented Software Development. However, this tool is still in early development stage and not all standard Java is supported such as ability to create dynamics web application. FeatureIDE is open-source and published under General Public License, so developers are encourage to extend it. We hope that in the near future, we could spend effort to the development of FeatureIDE by let it understand web application notation.

In this thesis work, we only have the connector code working for two prototype systems, Edmodo and inBloom. Since we are receiving increasing requests to integrate ASSISTments with Learning Tools Interoperability (LTI), we hope that in the near future, we could advance the thesis work by integrating seamlessly more learning management systems with ASSISTments.

Appendix A. Example of data return back from Edmodo

Teacher and student tokens

Teacher	user_token: unique number that identify user in the system (do not change). first_name last_name user_type: TEACHER time_zone groups: list of group that teacher are the owner. access_token: Authorizes and authenticates user login (change each time app launched)
Student	user_type: STUDENT user_token: unique number that indentify user in the system (do not change) first_name last_name access_token (change each time app lauched)

Teacher school profile

Profile	edmodo_school_id:123456 nces_school_id:ABC987654 name:Edmodo High address:1200 Park Place, Suite 350 city:San Mateo state:CA zip_code:94403 country_code:US

Group Info

Group	group_id:379557 title:Period 1 member_count:20, owners:[b020c42d1, 693d5c765], subject:Math, sub-subject:Algebra start_level:9th, end_level:9th

Appendix B: Connector code table

B1. API tables already exists in ASSISTments:

B.1.1. external_reference_types

This table provides type of reference created.

Field	Type	Foreign Key	Note
id	integer		
table_name	character[]		Values now: 1. users 2. schools 3. student class sections 4. class assignment 5. individual assignments 6. student classes.

B.1.2. external_references

This table will tell what type of data created via API by ASSISTments.

Field	Type	Foreign Key	Note
id	integer		
external_reference	character[]		Values in this column are automatically generated by ASSISTments correspond to request to create users, class, school via API
partner_id	integer		Point out who makes API request
type_id	integer	Yes	reference to id field in external_reference_types table
db_id	integer	Yes	reference to id field in correspond table. For example, if type_id = 1 (mean user) then this db_id reference to id field in users table.

B.1.3. access_tokens

Field	Type	Foreign Key	Note
id	integer		
user_id	character[]	Yes	point to id field of users table
partner_id	integer		
expiration	timestamp		
token	character[]		

B.2. A new bridge API tables

Table name: partner_external_references

Field	Type	Foreign Key	Note
id	integer		
partner_reference	character[]	Shows users belongs to which connecting system. Point to partner_refernece of api_partners table.	Point out who makes API request This is the partner's reference identifier provided by ASSISTments.
external_reference_type_id	integer		Specifies the object type of represented by this row. This is the id value found in the table external_reference_types
external_reference	character[]		This is an ASSISTments-provided unique identifier to an ASSISTments object. The type of object is specfied by the external_reference_type_id above.
user_access_token	character[]		When external_reference refers to an ASSISTments user, this is the access token granted to partner_reference to act on behalf of the ASSISTments user.

partner_extern al_reference	character[]		This value comes from the partner application (or site) and uniquely identifies an object in the partner's application. The type of object is specfified by the external_reference_type_id above.
user_connector _token	character[]		When partner_external_reference refers to a partner's user, this is the access token granted to act on behalf of the partner's user.
note	text		Data, specific to the partner application / site, about the object represented by this row.

Appendix C. Connector Classes

C1. Model Classes

- ▲ ⊞ assist.bean
  - ▷ Ⓙ Class.java
  - ▷ Ⓙ ClassSerializer.java
  - ▷ Ⓙ School.java
  - ▷ Ⓙ SchoolRef.java
  - ▷ Ⓙ SchoolSerializer.java
  - ▷ Ⓙ User.java
  - ▷ Ⓙ UserRef.java
  - ▷ Ⓙ UserSerializer.java
- ▲ ⊞ inbloom.bean
  - ▷ Ⓙ AcademicRecord.java
  - ▷ Ⓙ AcademicRecordSerializer.java
  - ▷ Ⓙ BirthDate.java
  - ▷ Ⓙ CourseOffering.java
  - ▷ Ⓙ CourseOfferingSerializer.java
  - ▷ Ⓙ CourseSection.java
  - ▷ Ⓙ CourseSectionSerializer.java
  - ▷ Ⓙ CourseTranscript.java
  - ▷ Ⓙ CourseTranscriptSerializer.java
  - ▷ Ⓙ CreditsEarned.java
  - ▷ Ⓙ CreditsEarnedSerializer.java
  - ▷ Ⓙ LinkSelfUserInfo.java
  - ▷ Ⓙ Name.java
  - ▷ Ⓙ SelfUserInfo.java
  - ▷ Ⓙ SessionCheck.java
  - ▷ Ⓙ Student.java

C2. Controller Classes

- ▲ ⊞ assistJob
  - ▷ AssignUserSchool.java
  - ▷ CreateAssignment.java
  - ▷ CreateClass.java
  - ▷ CreateSchool.java
  - ▷ EnrollStudentClass.java
  - ▷ LinkPrincipalUser.java
  - ▷ ScreenHTMLReport.java
  - ▷ TransferUserFromInBloom.java
  - ▷ UploadGradeToInBloom.java
- ▲ ⊞ inbloomJob
  - ▷ CreateCourseOffering.java
  - ▷ CreateCourseSection.java
  - ▷ CreateCourseTranscript.java
  - ▷ CreateStudentAcademicRecord.java
  - ▷ GetStudent.java
- ▲ ⊞ inbloomLogin
  - ▷ AccessToken.java
  - ▷ Authorize.java

References

[1] Hannemann, J., Kiczales, G., "Design Patterns Implementation in Java and AspectJ", Proceedings of the 17th ACM conference on Object-oriented programming, systems, languages, and applications (OOPSLA '02), Nov 2002.

[2] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), pages 220–242, 1997.

[3] D. Batory, J. Liu, and J. N. Sarvela. Refinements and Multi-Dimensional Separation of Concerns. In Proceedings of the International Symposium on Foundations of Software Engineering (FSE), pages 48–57, 2003.

[4] H. Ossher and P. Tarr. Hyper/J: Multi-Dimensional Separation of Concerns for Java. In Proceedings of the International Conference on Software Engineering (ICSE), pages 734–737, 2000.

[5] P. Clements and L. Northrop. Software Product Lines: Practices and Patterns. Addison-Wesley, 2002.

[6] K. Pohl, G. Böckle, and F. van der Linden. Software Product Line Engineering. Foundations, Principles, and Techniques. Springer-Verlag, 2005.

[7] C. Prehofer. Feature-Oriented Programming: A Fresh Look at Objects. In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), pages 419–443, 1997.

[8] Czarnecki, K.; Wasowski, A., "Feature Diagrams and Logics: There and Back Again," Software Product Line Conference, 2007. SPLC 2007.

[9] George T. Heineman, An Instance-Oriented Approach to Constructing Product Lines from Layers, WPI-CS-TR-05-06.

[10] Abbott R. Program design by informal English descriptions. Communications of the ACM 1983; 26(11).

[11] Wirfs-Brock R, Wilkerson B, Wiener L. Designing Object-Oriented Software. Prentice-Hall: Englewood Cliffs, New Jersey, 1990.

[12] Gomaa H. Software Design Methods for Concurrent and Real-Time Systems. Addison-Wesley: Reading, Massachusetts, 1993.

[13] Jacobson I, Christerson M, Jonsson P, Overgaard G. Object-Oriented Software Engineering. Addison-Wesley: Workingham, England, 1992.

[14] Beck K, Cunningham W. A laboratory for teaching object-oriented thinking. SIGPLAN Notices 1989; 24(10).

[15] M Kuhlemann, M Rosenmüller, S Apel, T Leich, On the duality of aspect-oriented and feature-oriented design patterns. Proceedings of the 6th workshop on Aspects, components, and patterns for infrastructure software, 2010

[16] Thüm et al, Applying Design by Contract to Feature-Oriented Programming, FASE 2012

[17] P. Velasco Elizondo and K.-K. Lau, A Catalogue of Component Connectors to Support Development with Reuse. The Journal of Systems and Software 83(1165-1178), 2010.

[18] Emmerich, W.; Kaveh, N., Component technologies: Java beans, COM, CORBA, RMI, EJB and the CORBA component model, ICSE 2002. Proceedings of the 24rd International Conference, 2002.

[19] Nastasi, B.K., & D.H. Clements. Motivational and social outcomes of cooperative education environments. Journal of Computing in Childhood Education, 1993

[20] Hadley, M., & Sheingold, K. Commonalties and distinctive patterns in teachers' integration of computers. American Journal of Education, 1993

[21] Harmelen and Workman, Analytics for Learning and Teaching, CETIS 2013.

[22] Sincero J, Schirmeier H, Schröder-Preikschat W, Spinczyk O, Is the Linux kernel a software product line? In: Proc. Int'l Workshop Open Source Software and Product Lines (SPLC-OSSPL), 2007.

[23] Tartler R, Lohmann D, Sincero J, Schröder-Preikschat W. Feature consistency in compiletime-configurable system software: Facing the linux 10,000 feature problem. In: Proc. Int'l EuroSys Conference (EuroSys). ACM Press, 2011.

[24] K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-021, 1990.

[25] K. Czarnecki and U. Eisenecker. Generative Programming: Methods, Tools, and Applications. Addison-Wesley, 2000.

[26] J. Liu, D. Batory, and S. Nedunuri. Modeling Interactions in Feature-Oriented Designs. In Proceedings of the International Conference on Feature Interactions in Software and Communication Systems (ICFI), 2005.

[27] Thum, T., Kastner, C., Erdweg, S., and Siegmund, N. Abstract features in feature modeling. In Proceedings of the 2011 15th International Software Product Line Conference, 2011.

[28] A catalogue of component connectors to support development with reuse P Velasco-Elizondo, KK Lau Journal of Systems and Software 83 (7), 2010.