CRANFIELD UNIVERSITY

Mateusz Golab

Forward integration of simultaneous ordinary differential equations with graphical output

School of Engineering

Software Engineering for Technical Computing

MSc THESIS

Academic Year: 2011 - 2012

Supervisor: Dr Peter Sherar, Prof Joanna Polanska

August 2012

CRANFIELD UNIVERSITY

School of Engineering

Software Engineering for Technical Computing

MSc THESIS

Academic Year 2010 - 2011

Mateusz Golab

Forward integration of simultaneous ordinary differential equations with graphical output

Supervisor: Dr Peter Sherar, Prof Joanna Polanska

August 2012

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science

This thesis is submitted in accordance with the Double Degree programme regulations. Home institution : Silesian University of Technology, Poland

© Cranfield University 2012. All rights reserved. No part of this publication may be reproduced without the written permission of the copyright owner.

ABSTRACT

Ordinary Differential Equations

Keywords:

ODE, Runge-Kutta, Modified Midpoint, Predictor - Corrector, Web development, GWT, AppEngine, Datastore, Unit testing.

ACKNOWLEDGEMENTS

Click here to enter acknowledgement text

TABLE OF CONTENTS

[ABSTRACT i](#_Toc332460125)

[ACKNOWLEDGEMENTS ii](#_Toc332460126)

[LIST OF FIGURES v](#_Toc332460127)

[LIST OF TABLES vi](#_Toc332460128)

[LIST OF EQUATIONS vi](#_Toc332460129)

[LIST OF ABBREVIATIONS viii](#_Toc332460130)

[1 Introduction 1](#_Toc332460131)

[1.1 Overview 1](#_Toc332460132)

[1.2 Aims and objectives 1](#_Toc332460133)

[1.3 Motivation 1](#_Toc332460134)

[1.4 Literature review 2](#_Toc332460135)

[1.5 Thesis structure 3](#_Toc332460136)

[2 Mathematical background 4](#_Toc332460137)

[2.1 Ordinary differential equations 4](#_Toc332460138)

[2.2 Euler’s method 5](#_Toc332460139)

[2.3 ODE numerical routines 5](#_Toc332460140)

[2.3.1 Runge-Kutta methods 5](#_Toc332460141)

[2.3.2 The modified midpoint 8](#_Toc332460142)

[2.3.3 Richardson extrapolation 9](#_Toc332460143)

[2.3.4 Rosenbrock 11](#_Toc332460144)

[2.3.5 Predictor- Corrector 11](#_Toc332460145)

[3 Technologies, Methodologies and Application Design 14](#_Toc332460146)

[3.1 AJAX approach 14](#_Toc332460147)

[3.2 Technologies used 15](#_Toc332460148)

[3.2.1 Google Web Toolkit 16](#_Toc332460149)

[3.2.2 Google AppEngine 17](#_Toc332460150)

[3.2.3 Datastore 18](#_Toc332460151)

[3.3 Architecture 21](#_Toc332460152)

[3.4 Design patterns 23](#_Toc332460153)

[3.5 The Prototyping Model 24](#_Toc332460154)

[3.6 Agile approach 25](#_Toc332460155)

[3.6.1 Feature Driven Development 26](#_Toc332460156)

[3.6.2 Test Driven Development 26](#_Toc332460157)

[4 Testing 28](#_Toc332460158)

[4.1 Unit testing 28](#_Toc332460159)

[4.1.1 GWT Unit Testing infrastructure 28](#_Toc332460160)

[4.1.2 TDD approach 28](#_Toc332460161)

[4.1.3 Test Cases 28](#_Toc332460162)

[4.2 Black-Box testing 31](#_Toc332460163)

[5 Implementation 33](#_Toc332460164)

[5.1 The anatomy of the project 33](#_Toc332460165)

[5.2 Model 34](#_Toc332460166)

[5.2.1 Equation 34](#_Toc332460167)

[5.2.2 System 34](#_Toc332460168)

[5.2.3 SystemEntity 35](#_Toc332460169)

[5.2.4 Solution 35](#_Toc332460170)

[5.3 Equation Parser 36](#_Toc332460171)

[5.3.1 ParserServiceImpl 36](#_Toc332460172)

[5.4 ODE Solvers 37](#_Toc332460173)

[5.4.1 Solver base class 37](#_Toc332460174)

[5.4.2 ODE Solver services 37](#_Toc332460175)

[5.5 Graph Viewer 38](#_Toc332460176)

[5.6 Equation Store 38](#_Toc332460177)

[5.7 User interface 39](#_Toc332460178)

[5.7.1 Input Panel 39](#_Toc332460179)

[5.7.2 Equation Panel 40](#_Toc332460180)

[5.7.3 Graph Panel 41](#_Toc332460181)

[6 Results 42](#_Toc332460182)

[6.1 Validation and verification results 42](#_Toc332460183)

[6.2 Application’s outputs 42](#_Toc332460184)

[7 Discussion and conclusion 43](#_Toc332460185)

[7.1 Solvers correctness 43](#_Toc332460186)

[7.2 Limitations 43](#_Toc332460187)

[7.3 Problems faced 43](#_Toc332460188)

[7.4 Quality of implementation 43](#_Toc332460189)

[7.5 Conclusion 43](#_Toc332460190)

[7.6 Future work 43](#_Toc332460191)

[REFERENCES 45](#_Toc332460192)

[APPENDICES 48](#_Toc332460193)

# LIST OF FIGURES

[Figure 2‑1 4th Order Runge-Kutta method 7](#_Toc332460194)

[Figure 2‑2 Richardson extrapolation used in the Burlisch-Stoer method with substep n = 2,4,6 10](#_Toc332460195)

[Figure 2‑3 Aitkens-Neville polynomial extrapolation tableau 10](#_Toc332460196)

[Figure 3‑1 Comparison of AJAX light traffic needs vs legacy HTML applications 15](#_Toc332460197)

[Figure 3‑2 An overview of GWT approach 16](#_Toc332460198)

[Figure 3‑3 Application architecture overview 22](#_Toc332460199)

[Figure 3‑4 Example of Command Pattern structure 23](#_Toc332460200)

[Figure 3‑5 Example of Composite Pattern structure 24](#_Toc332460201)

[Figure 3‑6 The prototyping lifecycle 25](#_Toc332460202)

[Figure 3‑7 Test Driven Development cycle 27](#_Toc332460203)

[Figure 5‑1 Input Panel 40](#_Toc332460204)

[Figure 5‑2 Equation Panel with example of ODE's system 40](#_Toc332460205)

[Figure 5‑3 2D graph presenting solution of sample system of ODE's 41](#_Toc332460206)

LIST OF TABLES

[Table 3‑1 Data types available in Datastore 20](#_Toc332460690)

[Table 3‑2 Datastore limitations 21](#_Toc332460691)

[Table 4‑1 Parser Test Case part 1 29](#_Toc332460692)

[Table 4‑2 Parser Test Case part 2 29](#_Toc332460693)

[Table 4‑3 Parser Test Case part 3 29](#_Toc332460694)

[Table 4‑4 Parser Test Case part 4 30](#_Toc332460695)

[Table 4‑5 Parser Test Case part 5 30](#_Toc332460696)

[Table 4‑6 Parser Test Case part 6 30](#_Toc332460697)

[Table 4‑7 Parser Test Case part 7 30](#_Toc332460698)

LIST OF EQUATIONS

[(2‑1) 4](#_Toc332460207)

[(2‑2) 4](#_Toc332460208)

[(2‑3) 5](#_Toc332460209)

[(2‑4) 5](#_Toc332460210)

[(2‑5) 5](#_Toc332460211)

[(2‑6) 5](#_Toc332460212)

[(2‑7) 6](#_Toc332460213)

[(2‑8) 6](#_Toc332460214)

[(2‑9) 6](#_Toc332460215)

[(2‑10) 6](#_Toc332460216)

[(2‑11) 6](#_Toc332460217)

[(2‑12) 8](#_Toc332460218)

[(2‑13) 8](#_Toc332460219)

[(2‑14) 8](#_Toc332460220)

[(2‑15) 8](#_Toc332460221)

[(2‑16) 8](#_Toc332460222)

[(2‑17) 9](#_Toc332460223)

[(2‑18) 9](#_Toc332460224)

[(2‑19) 10](#_Toc332460225)

[(2‑20) 11](#_Toc332460226)

[(2‑21) 11](#_Toc332460227)

[(2‑22) 11](#_Toc332460228)

[(2‑23) 12](#_Toc332460229)

[(2‑24) 12](#_Toc332460230)

[(2‑25) 13](#_Toc332460231)

[(2‑26) 13](#_Toc332460232)

LIST OF ABBREVIATIONS

|  |  |
| --- | --- |
| API  CU  GAE  GUI  GWT  FDD  JDO  JPA  JVM  ODE  PaaS | Application Programming Interface  Cranfield University  Google App Engine  Graphical User Interface  Google Web Toolkit  Feature Driven Development  Java Data Objects  Java Persistence API  Java Virtual Machine  Ordinary differential equation  Platform as a Service |
| RPC  SDK  SWT  TDD  UI  UT | Remote Procedure Call  Software Development Kit  Standard Widget Toolkit  Test Driven Development  User Interface  Unit Test |
|  |  |
|  |  |
|  |  |

# Introduction

The Introduction chapter focuses on descriptive background to the work and points aims and objectives of the thesis project. All requirements of the project are explained in this chapter. Moreover, main motivators to perform this specific topic are included as well. This chapter also contains literature review, and thesis structure.

## Overview

Series of simultaneously ordinary differential equations model physical systems that evolve over time. There is a wide variety of software solving ODE. However, it is hard to find an easy to use robust application providing good quality solutions with clear visualisation of the results. This thesis project aims to combine the most efficient and accurate numerical methods, solving ODE’s and systems of ODE’s, with robust web technologies providing modern solution visualisation . There are scientific and technological challenges present in this project. Furthermore, since main part of the thesis is about developing a specific application, issues concerning software development process are also significant.

## Aims and objectives

The main objective of this thesis project is to develop following application :

1. Parsing equations entered by the user
2. Solving series of 1st order simultaneous ordinary differential equations (linear or non-linear) for initial value problem.
3. Presenting the solution on the 2D graph.
4. Storing and loading equations along with parameters entered by the user.

## Motivation

In general, two main motivators of this project are :

1. Discovering most efficient numerical methods for solving series of ordinary differential equations.
2. Familiarise with AJAX applications development and Google AppEngine along with Datastore.

## Literature review

The literature review gives an insight into the numerical methods solving ODE’s, technologies like Google AppEngine along with Datastore and AJAX – based applications development with emphasis on GWT.

The three major types of practical numerical methods for solving ODE’s (Press, Teukolsky, Vetterling , Flannery 2007) are :

* Runge-Kutta methods
* Methods using Richardson extrapolation, especially Bulirsch-Stoer method.
* Predictor-corrector methods (also known as multistep methods)

Extensive study of GWT and AJAX approach in (Guermeur, Unruh 2010), (Chaganti 2007), (Cooper, Collins 2008) and (Google Developers online articles 2012) noted that usage of Google Web Toolkit is a good way to achieve robust web application.

Usage of Google App Engine, which is Google’s PaaS, to run web applications is noted to be very good idea in terms of reliability, scalability and robustness. Furthermore, availability of App Engine’s Datastore is described as well integrated and reliable storage (Guermeur, Unruh 2010).

In terms of software lifecycle models , the Prototyping Model seems the most suitable for this project (Barnes 2012) . From the Agile approach flavours , TDD and FDT are suitable to a certain degree for this thesis (Barnes 2012), (Nebulon Pty. Ltd 2003), (Wellman 2011)

Testing methodology applied for GWT applications is based on Unit Testing and cross-browser UI testing (Wellman November 2008)

## Thesis structure

The structure of further chapters of this thesis report are stated below:

* In chapter 2 the mathematical background to the methods implemented in the application is presented
* Chapter 3 contains descriptions of the technologies, methodologies and application design along with software lifecycle model.
* Chapter 4 concerns testing strategies and methodologies undertaken to verify and validate the developed software.
* In chapter 5 the implementation details of the application are presented.
* Chapter 6 is about results achieved in this thesis project
* Discussions and conclusions of the project are presented in chapter 7.

# Mathematical background

This chapter demonstrates knowledge about recommended practical numerical methods for solving ODEs. Presented literature review gives an insight into major types of numerical routines taking into account efficiency and accuracy. What is more, technologies supporting AJAX applications development along with App Engine are also presented in this chapter. It is important to perform detailed research about technologies essential to develop the project.

## Ordinary differential equations

General definition of the ordinary differential equation of order :

|  |  |
| --- | --- |
|  | (2‑1) |

Important issues involving ODEs are boundary conditions. In other words algebraic conditions on the values of the function . In general they divide into two broad categories :

* Initial value problems

All are given at some starting point and it is desired to find the at some final point or list of points with specified intervals, for example .

* Two-point boundary value problems

Where boundary conditions are specified at more than one point. For example at starting and final point.

General definition of the system of ODEs of order :

|  |  |
| --- | --- |
|  | (2‑2) |

|  |  |
| --- | --- |
|  | (2‑3) |

## Euler’s method

Euler’s method is a first-order numerical method for solving ODEs . It is not recommended method for practical use, however it is important conceptually for advanced methods.

|  |  |
| --- | --- |
|  | (2‑4) |

|  |  |
| --- | --- |
|  | (2‑5) |

## ODE numerical routines

Describes types of practical numerical methods for solving ODEs including Runge-Kutta methods, Bulirsch-Stoer, Rosenbrock metods and predictor-corrector rmethods.

### Runge-Kutta methods

Runge-Kutta methods propagate a solution over an interval by combining the data from several Euler-style steps. Each step involves one evaluation of the right-hand of the function .

|  |  |
| --- | --- |
|  | (2‑6) |

Thenusing the information obtained to match a Taylor series expansion up to higher order.

Developing higher – order methods made Runge-Kutta competitive with the other numerical methods in many cases. It is usually the fastest method when moderate accuracy is required (≤ ) and evaluation of the function is not too expensive. There are few kinds of Runge-Kutta methods : 2nd order method, (called midpoint method), 4th order method and also method with adaptive stepsize. [numerical recipes]

#### 4th Order Runge-Kutta

The most often used Runge-Kutta method is fourth – order formula . In general it is superior to 2nd order method, however high order does not always mean high accuracy. 4th order method requires four evaluations of the function. [numerical recipes]

|  |  |
| --- | --- |
|  | (2‑7) |

|  |  |
| --- | --- |
|  | (2‑8) |

|  |  |
| --- | --- |
|  | (2‑9) |

|  |  |
| --- | --- |
|  | (2‑10) |

|  |  |
| --- | --- |
| Figure 2‑1 4th Order Runge-Kutta method | (2‑11) |

During single step, the derivative is evaluated four times. Once at initial point, (1) twice at midpoints (2)(3) and once at trial endpoint (4). Final function value is calculated on the basis of these derivatives. [numerical recipes]

Each step in the sequence of steps is treated in an identical manner, so prior behaviour of the solution is not used in its propagation. Such approach is mathematically proper, since any point along the trajectory of an ODE can be an initial point. [numerical recipes]

#### Runge-Kutta with adaptive stepsize

The purpose of adaptive stepsize method is to achieve predetermined accuracy in the solution with minimum computational effort. It is possible to face very smooth interval , while performing Runge-Kutta steps. Few great strides instead of small steps should speed through such undifferentiated interval, what may result in significant gain in efficiency. The idea of adaptive stepsize method is to control the size of the step and increase it when possible maintaining required level of accuracy. It is important to estimate truncation error to control accuracy level while increasing step size. Obviously the calculation of this information will add to the computational overhead, however it is profitable investment in terms of efficiency.

### The modified midpoint

Modified midpoint method is a second order method like 2nd order Runge-Kutta, however with the advantage of requiring only one derivative evaluation per single step instead of two evaluations present in Runge-Kutta. This method generates the solution as a vector of values from a point  to a point by a sequence of substeps each of size . Where

|  |  |
| --- | --- |
|  | (2‑12) |

The total number of function evaluations required by this method is . The formulas essential to provide the solution for are as follows

|  |  |
| --- | --- |
|  | (2‑13) |

|  |  |
| --- | --- |
|  | (2‑14) |

|  |  |
| --- | --- |
|  | (2‑15) |

|  |  |
| --- | --- |
|  | (2‑16) |

The is the final approximation to whereas represents intermediate approximations calculated along in steps of .

### Richardson extrapolation

Richardson extrapolation bases on idea of extrapolating a computed value to the value that would have been obtained if the stepsize had been remarkably smaller than it actually was. The practical numerical method using this idea is called Bulirsch-Stoer method.

#### Burlisch – Stoer

The idea of Burlisch-Stoer method is to perform iterations of modified midpoint method . Each iteration uses various number of substepsfor modified midpoint method ends up with polynomial extrapolation of the given values. Burlish and Stoer originally proposed following sequence of substeps :

|  |  |
| --- | --- |
|  | (2‑17) |

However sequence discovered by Deuflhard is usually more efficient :

|  |  |
| --- | --- |
|  | (2‑18) |

In terms of number of iterations usually 8 gives satisfactory results.



Figure 2‑2 Richardson extrapolation used in the Burlisch-Stoer method with substep n = 2,4,6

We use Aitkens-Neville algorithm in order to perform extrapolation, which is described by the following tableau :



Figure 2‑3 Aitkens-Neville polynomial extrapolation tableau

The first column of the tableau is formed by modified midpoint first iteration with n = 2 .

|  |  |
| --- | --- |
|  | (2‑19) |

Where is computed with the stepsize

Successive columns can be filled by using following recurrence :

|  |  |
| --- | --- |
|  | (2‑20) |

The final solution which is can be achieved after performing each successive iteration.

### Rosenbrock

Rosenbrock methods are competitive with other numerical ODEs integrators in terms of moderate accuracies (tolerances of order . Moreover these methods remain reliable for more stringent parameters . The formula of Rosenbrock method is as follows :

|  |  |
| --- | --- |
|  | (2‑21) |

Where corrections are found after solving following linear equations :

|  |  |
| --- | --- |
|  | (2‑22) |

The coefficients are fixed and Jacobian matrix is denoted by . [ numerical recipes]

### Predictor- Corrector

Predictor – Corrector methods are a subcategory of methods called “multistep” and “multivalue”. These methods have had long historical run. It is said that that predictor-corrector integrators have had their day. For high precision applications and right-hand side expensive evaluations Bulirsch-Stoer method dominates. For moderate precision problems Runge-Kutta methods dominates. However there is possibly one exceptional case where predictor-corrector dominates. It is the case of high-precision solutions of very smooth equations with complicated right-hand side evaluations.

Considering multistep approach it is important to realize the difference between integrating an ODE and finding the integral of a function. For a function , the integrand has a dependence on the independent variable . However for an ODE, the “integrand” (which is right-hand side) depends both on and dependent variables . So in order to advance the solution of from to we have :

|  |  |
| --- | --- |
|  | (2‑23) |

According to a multistep approach is approximated by a polynomial passing through several previous points and possibly through .

The formula that is evaluating the integral (2-18) at is then of the form:

|  |  |
| --- | --- |
|  | (2‑24) |

Where

There is a method called which solves an implicit formula of the form (2-19) for . Such method is called . The idea of this method is to take initial guess for , then insert it into the right-hand side of (2-19) and get updated value of . In order to get initial value of we have to extrapolate the polynomial fit to the derivative from the previous points to the new point . The next stage of solving process is made by which is using the prediction step’s value of to the derivative. In conclusion Predictor-corrector method comprises of three separated processes :

* Predictor step
* Evaluation of the derivative from the latest value of .
* Corrector step

#### Adams-Bashforth-Moulton

Probably the most popular method is method. This method has good stability properties . The predictor part is called The Adams-Bashforth :

|  |  |
| --- | --- |
|  | (2‑25) |

The Adams-Moulton part is the corrector :

|  |  |
| --- | --- |
|  | (2‑26) |

# Technologies, Methodologies and Application Design

This section is a combination of methodologies chosen for this thesis project and Design part. This chapter also includes description of software lifecycle model which was chosen for this project. Moreover, description of agile approach applied in this thesis is also described.

Literature review concerning technologies is mainly focused on web application point of view. AJAX approach is taken into account along with Google Web Toolkit. Also Google App Engine’s possibilities for maintaining web-based applications are described.

## AJAX approach

AJAX which stands for Asynchronous JavaScript and XML refers to a set of web-development techniques including the use of JavaScript, CSS and asynchronous HTML requests. The biggest advantage of this set of techniques is bringing desktop-like experience to the client-side of a web application. AJAX applications asynchronously request only the necessary information from a server. That is why they reduce the traffic and the server load comparing to non-AJAX web applications , which synchronously request whole web pages from their servers. Good example to realize it is following situation. Users are trying to log in to their e-mail application. After authentication process the only real job of the application is to replace a login link by some user welcoming message. Following figure presents how web traffic is happening for AJAX and a HTML-only non-AJAX applications for described scenario. [ gwt + appeng]



Figure 3‑1 Comparison of AJAX light traffic needs vs legacy HTML applications

## Technologies used

Part of literature review concerning technologies helped with making decisions about which technologies, tools and frameworks to use. The project is designed as an AJAX application using **Google Web Toolkit 2.4** .

**Google App Engine** (SDK version 1.6.4) is responsible for running application and storing data on the Datastore. It is also responsible for automatic scaling and load balancing.

**Objectify** as an lightweight and effective framework is used to simplify working with Datastore.

**Exp4j** library is applied in ODE Solvers implementation. It is capable of evaluating expressions and functions in the real domain . This library is very useful for evaluation of right-hand-side functions in numerical routines implemented in the ODE Solvers.

In order to provide good quality 2D chart presenting solution , **Google Chart Tools 1.1** library is used. It is also called and provides wide variety of charts and graphs.

In order to provide rich GUI , **Ext GWT** (**GXT) 2.2.5** library was used. It is a Java library for building internet application with GWT.

### Google Web Toolkit

Google Web Toolkit is a development toolkit that enables you to build AJAX applications using the Java language which is compiled to JavaScript. GWT minimizes the cross-browser issues and enables productive development of high-performance web applications. It encapsulates the object API and provides a set of ready-to use interface components and widgets. Google Web Toolkit provides framework for building RPC services, which provide certain functionalities that can be accessed asynchronously from web application.



Figure 3‑2 An overview of GWT approach

GWT consists of four main components listed below, which provide functionalities for writing AJAX applications.

* **GWT Java to JavaScript Compiler**

GWT compiler compiles and optimizes GWT applications written in Java to JavaScript. That is why an application can be deployed to a web container.

* **GWT Hosted Web Browser**

This component enables you to run and execute GWT applications in JVM , in hosted mode, without compiling to JavaScript . GWT provides such possibility by embedding a special SWT browser control, that contains hooks into the JVM.

* **JRE emulation library**

Contains JavaScript implementations (for the client-side implementation ) of most widely used packages in Java standard class library like , which are used on the client-side. On server-side implementation you are free to use entire Java class library.

* **GWT Web UI class library**

This provides a set of interfaces and classes that enable you to create various UI components and widgets. Moreover you are free to use ready-to-use widgets in your applications.

### Google AppEngine

Google App Engine is a PaaS cloud computing platform for developing and hosting web applications using Google’s servers and infrastructure. Supports Java, Python and Go runtime environments. GAE as a platform is designed for scalability , robustness and performance. It is very well integrated with Google Web Toolkit. Once you develop application using GWT, you can deploy it on the GAE.

App Engine distributes requests for applications across multiple web servers to prevent application from interfering with another. In order to achieve it applications run in a restricted “sandbox” environment. Applications in this environment can execute code, store and load data from App Engine’s Datastore and examine web requests.

In terms of limitations , an App Engine application cannot :

* Write to the filesystem. All operations connected with storing and quering data have to be done using Datastore.
* Access another host directly or open a socket.
* Make some other kind of system call

### Datastore

The Google App Engine’s Datastore is a non-centralized persistent store, based on Google’s BigTable technology. It provides robust and scalable storage for web applications running on App Engine. Datastore is not a relational database based on join queries, it is rather a property-value store holding specified objects known as . It was designed to manage scaling to very large data sets in distributed architecture. For this reason Datastore is different than traditional relational databases.

Java Datastore API provides low-level operations on entities. Google App Engine SDK includes implementations of JDO and JPA interfaces for modelling and persisting data. It is possible to use Datastore API directly in your applications, however there is also an option to use a framework which simplifies Datastore usage for Java developers. There are three frameworks recommended by Google App Engine team.

* **Objectify**

Very simple and convenient interface to the App Engine Datastore. Helps with avoiding some complexities of JDO/JPA and low-level Datastore.

* **Twig**

This framework provides a configurable object persistence interface that improves support for inheritance, polymorphism and generic types. Similarly to Objectify helps you to avoid Datastore complexities.

* **Slim3**

Slim3 is not limited only to Datastore. It is a MVC framework which can be used for wide variety of App Engine functions.

As mentioned at the beginning of this chapter, framework chosen for this project is .

Each has one or more named properties . Each property can have one or more values. The available data types defined for Datastore are presented in Table 3-1.

Table 3‑1 Data types available in Datastore



There are following limitations concerning Datastore :

Table 3‑2 Datastore limitations

|  |  |
| --- | --- |
| Limit | Amount |
| Maximum entity size | 1 megabyte |
| Maximum transaction size | 10 megabytes |
| Maximum number of values in all indexes for an entity | 5000 |
| Text string (short) | Up to 500 Unicode characters |
| Text string (long) | Up to 1 megabyte |
| Byte string (short) | Up to 500 bytes |
| Byte string (long) | Up to 1 megabyte |
| Floating-point types | 64-bit double precision |

## Architecture

From the very beginning of this thesis, the idea was to develop it as a web application. The main reason is to make it accessible from any web browser and ready to use without installation. Application is divided on server-side and client-side implementation.

Application is divided into several modules :

* **Equation Parser**

This module is responsible for parsing user’s input into ODE objects or System of ODEs objects .

* **ODE Solver**

Contains implementations of ODE numerical routines. Solver operates on objects returned by the Equation Parser.

* **Graph Viewer**

This component is responsible for presenting Solution object, returned by the ODE Solver, as a 2D graph with intervals specified by the user.

* **Equation store**

This module is a Datastore connector, which is responsible for storing and quering Equations on the Datastore.



Figure 3‑3 Application architecture overview

According to good software design and development practices, Equation Parser, Equation Store and ODE Solver are processed on the server-side of the application. The client-side implementation contains User Interface and Graph viewer.

## Design patterns

The whole structure of the application is kept in MVC pattern to maintain separation of concerns.

Command and Composite are design patterns used in GWT.

Command pattern encapsulates a request as an object. In fact this pattern lets toolkit turn the request itself into an object, which can be stored and passed around like other objects. In other words Command objects can be thought of as “tokens” that are created by one client that knows what need to be done and passed to another client that has required resources to perform this operation.



Figure 3‑4 Example of Command Pattern structure

Composite is a structural pattern, which composes objects into tree structure to represent whole-part hierarchies. The idea of this pattern is to compose objects in such a way , that client sees many of them as single objects. The advantage of applying Composite pattern is using some operations for an object in the same way as for group of objects.



Figure 3‑5 Example of Composite Pattern structure

## The Prototyping Model

Prototyping lifecycle model is an example of Evolutionary model. In general prototype means a version of the software used to test different solutions to the problem. Such approach allows better understanding of the ways solution might work and could be useful to choose the best solution for the final version of the software. The most important factor in prototyping is the feeding back of experience from each iteration into improving the design.



Figure 3‑6 The prototyping lifecycle

During development process of this thesis project, features of the application were developed according to the prototyping model. At the beginning of the cycle, requirements are gathered . Then quick design phase is performed. Another step is to develop prototype according to the prepared design. The next step is evaluation of the developed prototype in agreement with thesis supervisor. Another step is gathering comments about prepared prototype and refining it by repeating the cycle from design part. This process is repeated as long as feature requires improvements and corrections .

## Agile approach

The key ideas of Agile approach are reacting on changes in project quickly and loose grouping of similar software process models. In other words, developing model sets up lightweight structure of the project with practical hands-on approach and intensive usage of evolutionary and iterative methods. Moreover, in this approach customer plays an active part in a development process.

### Feature Driven Development

FDD is an agile approach which treats project as a set of features and can be used on almost any project regardless of size and technology.

The first process in FDD is to develop overall model of an application. The idea of this stage is to gain good understanding of the problem domain.

The second process is to prepare a list of features. Each feature is a small client-valued requirement which typically takes 1-3 days to implement.

The third and last process is called “Plan by Feature” . Taking into account risks and dependencies of the features, the order of the features to develop is prepared.

### Test Driven Development

Test Driven Development is also one of the Agile methodologies used in this project. In traditional testing approach tests are created after developing some piece of software. TDD turns traditional approach around. Instead of writing functional code first and then testing code as an afterthought , you write test code before functional code. Furthermore , the best way to perform TDD is by making small steps, one test and one small bit of corresponding functional code. It sounds simple in principle, however requires great discipline. It is easy to “slip” and write functional code avoiding new tests.



Figure 3‑7 Test Driven Development cycle

# Testing

Testing is an important part in a software development process. In terms of GWT applications, testing infrastructure is based on classes.

## Unit testing

Represents lowest level of testing in validation and verification process. The aim of single UT is to check correctness of the smallest component of source code, usually a method of a class.

### GWT Unit Testing infrastructure

Since a GWT application is written in Java, you are free to use JUnit as a Unit Testing framework. Google Web Toolkit provides which is a subclass of JUnit’s . This class is necessary to test native JavaScript code. Such UT can be performed on hosted-mode browser, provided by GWT. In fact there is no difference for a developer between using standard class and class, only is an additional method which must be implemented. This method returns a string containing the name of GWT code module as defined in module configuration file of the application. Running test cases extending starts up hosted-mode browser and then evaluates prepared tests against it. Such testing infrastructure allows invoking asynchronous RPC calls, run native JavaScript functions and render widgets.

### TDD approach

TDD approach is used in this thesis project to a small extent. The vast majority of unit tests prepared for ODE Solvers was written before implementation phase. The idea was to define set of specific ODEs checking correctness of the solvers.

### Test Cases

There are several Test Cases prepared to verify correctness of the developed software :

#### ParserTestCase

Contains tests concerning Equation Parser component. There is a wide variety of tests checking parser reactions for a correct and incorrect input as well.

Table 4‑1 Parser Test Case : correct ODE’s

|  |  |
| --- | --- |
| Test name | Parser input |
| testCorrectEquation1 | y' = y + x, y = 0 |
| testCorrectEquation2 | y''' = y + x, y=0 |
| testCorrectEquation3 | y ' ' ' = y+x, y = 0 |
| testCorrectEquation4 | y''' = y'' + y' + y + x = 5, y = 0 |
| testCorrectEquationFunction | y''' = y '' + y\*y' - y + x + 4, y = 0 |

Set of tests checking parser for correct ODE’s. Presented tests differs from each other in terms of equation’s complexity and order.

Table 4‑2 Parser Test Case : incorrect ODE’s

|  |  |
| --- | --- |
| Test name | Parser input |
| testNoODE | y + x |
| testNoODEWithInitVariable | y = tre , y = 0 |

Tests performed within this Test Case provide characters put together which are not constituting a differential equation.

Table 4‑3 Parser Test Case : equation’s incorrect variables

|  |  |
| --- | --- |
| Test name | Parser input |
| testToManyVariables | y’ = y + x + z, y = 0 |
| testNoIndependentVariable | y''' = y '' + y\*y' - y + 4, y = 0 |
| testTooManyIndependentVariables | y''' = y '' + y\*y' - y + 4 + x + u, y = 0 |

This Test Case verifies parsing ODE’s containing incorrect and correct states of variables in an equation. There are tests performed containing too many variables in an equation. Moreover equation which has no independent variables is verifies in terms of correctness for the parser.

Table 4‑4 Parser Test Case : case sensitivity

|  |  |
| --- | --- |
| Test name | Parser input |
| testCaseInsensitivity | y'' = y + x + 2\*X + 3/Y', y = 0 |

This test checks whether equation is correctly parsed when characters are case insensitive.

Table 4‑5 Parser Test Case part 7

|  |  |
| --- | --- |
| Test name | Parser input |
| testMathExpressions | y'=sin(x)+cos(x)+exp(x)+log(x)+tan(x)+sqrt(x), y = 0 |

Test verifies if mathematical expression are supported by the parser.

Table 4‑6 Parser Test Case : initial values

|  |  |
| --- | --- |
| Test name | Parser input |
| testWrongInitVariable | y' = y +3, u =4 |
| testNoInitVariable | y' = x + 4 |

This Test Case is verifying if exception is being thrown when initial values of the equations are omitted. Additionally another test is performed to check situation when initial values concerns irrelevant variable.

Table 4‑7 Parser Test Case : system of ODE's

|  |  |
| --- | --- |
| Test name | Parser input |
| testSystemOfODE | "y'= y + x, y = 0" ,  "z'= z + y + x, z = 0" |

The idea of this test is to verify parsing systems of ODE’s.

#### SolverTestCase

The is a base class for all Test Cases prepared to verify correctness and acuracy of ODE Solvers implemented. In order to get correct solutions, which are essential for comparison in unit tests, system was used. It is using an online mathematical system called , which is very precise and efficient when it comes to ODEs solving. All ODE Solver Test Cases extend and perform the same tests. The only difference is an error tolerance, which depends on the method implemented.

Table 4‑8 Solver Test Case : 1st order linear ODE

|  |  |
| --- | --- |
| Test name | Equation content |
| test1OrderEquation | y'=-2\*y+x+4 |

Tests verifies result of the Solver within a range [0.0 , 0.2] and step = 0.01 for the linear ODE. The correct answer was provided from analytic solution of the ODE. Tests checks number of solution values included in object and value at the point 0.2.

Table 4‑9 Solver Test Case : 1st order non-linear ODE

|  |  |
| --- | --- |
| Test name | Equation content |
| test1OrderMathPowerEquation | y'=y+x^2 |
| test1OrderMathSinCosEquation | y'=sin(x)+cos(x) |

This set of tests verifies accuracy of non-linear ODE’s. Both tests are performed with the domain [0.0 , 1.0] and step 0.1. Solution compared in these unit tests is achieved using ‘s .

Table 4‑10 Solver Test Case : 1st order system of ODE's

|  |  |
| --- | --- |
| Test name | System of ODE’s |
| test1OrderSystem | y'= y + x  z'= z + y + x |
| test1OrderSystem3 | y'= y + x  z'= z + y + x  u'= u + x + 4 |

Tests provided in this Test Case verify solution accuracy for systems of ordinary differential equations. The and check accuracy of a system which comprises of respectively two and three ODE’s. The domain for all equations is defined as [0.0 , 1.0] with step = 0.1 . All initial values for the functions , and (in the second system) are 0.0 .

#### PersistenceTestCase

TestCase responsible for testing Equation store component. Tests performed are checking operations of storing and querying data on Datastore.

<TODO>

## Black-Box testing

The idea of black-box testing is to test software according to requirements specification, without any knowledge of internal structure. According to this approach application is treated as a “black-box” and testing process is basically based on performing use cases of the application.

Since vast majority of use cases can be covered with unit tests for this project, black – box tests concern user interface tests mainly.

Functionalities of the application were tested according to black-box testing strategy on different web browsers :

* Google Chrome 20.0
* Mozilla Firefox 14.0.1
* Microsoft Internet Explorer 8.0

# Implementation

Implementation chapter is divided into four parts according to main application’s modules . As mentioned before, implementation is organised according to MVC pattern. Furthermore, the structure of the project is kept as a typical GWT application structure.

## The anatomy of the project

This project is structured according to GWT application convention. The most important parts of project are as follows :

* **client package**

This package contains client-side implementation. This code will be cross-compiled to JavaScript. Includes

* **server package**

Server-side implementation, contains Equation Parser, ODE Solvers and Persistence services.

* **shared package**

This package contains all classes used on server-side and client-side implementation. It includes and packages, which contain classes used in both implementations.

* **ThesisAE.gwt.xml**

GWT module definition file. Declares several primary elements : inherited modules, servlet deployments, compiler plugins and entry points.

* **web.xml**

Contains mappings for servlets specified in the .

* **ThesisAE.java**

Main class of the application, implements . It is a starting class invoked by the module.

## Model

According to MVC pattern which is used in this project there is separated model implementation located in package. That is why all model classes are available for client-side and server-side implementation. Model comprises of the following classes:

### Equation

It is a very important class, which represents equation in the application. Equation is created after parsing stage, and ODE solvers are operating on it during solving process. This class representation is based on following data :

**private** List<Double> initValues;

**private** String equationContent;

**private** **char** independentVariable;

**private** **char** functionVariable;

**private** **int** order;

initValues stores equation’s initial values, equationContent is a content of the equation presented as a string, independentVariable and functionVariable are represented as characters and order is an order of the equation stored as int.

All class methods are standard data accessors, which for Java programming language are so called setters and getters. Moreover several constructors were implemented to instantiate an object depending on the data available to create such an object .

### System

System class represents system of equations in the application, which are stored in a list :

**private** List<Equation> equations

Except standard setters and getters, class provides :

**public** **void** addEquation(Equation equation)

which enables scenario when you creates object and adds equations one by one.

### SystemEntity

This class is an entity class represented in AppEngine’s Datastore. It is an expanded version of the in terms of information stored as stated below :

@Id

**private** String name;

**private** List<String> equations;

**private** **double** min;

**private** **double** max;

**private** **double** step;

**private** Date date;

Except list of equations there is ,inter alia, name variable, which is an entity’s identifier in a Datastore. Furthermore, min and max indicates the borders of the interval and step represents a step which a system will be solved with. Finally there is date which stores the date when was last modified on the Datastore.

This class is also used to store single objects in Datastore. Such entity simply contains only one in the list.

### Solution

The class is a representation of the result provided by the ODE Solver. The structure of the data represented by this class is following :

**private** List<Double> results

**private** **double** min

**private** **double** max

**private** **double** step

List of the values provided by the solver is stored in results. Other variables min, max and step are necessary for presenting a solution on the graph.

## Equation Parser

Equation Parser is implemented as the service. This class includes set of methods necessary to perform parsing operations on user input.

### ParserServiceImpl

**public** Equation parseEquation(String input) **throws** IncorrectODEEquationException

The main method responsible for parsing ODE, given parameter is a input provided by the user. Method returns object which includes all equation data necessary for the ODE Solver.

There are several steps of parsing process. At the beginning, input is separated into equation content and initial values part. Then following methods of are executed to finish the process :



Parses initial values



Retrieves character which stands for a function in parsed equation.

Retrieves character which stands for an independent variable of the function in the equation.

**public** System parseEquationsSystem(List<String> inputs) **throws** IncorrectODEEquationException

Method responsible for parsing System of Equations, parameter is a list of equations in a form provided by the user. As a result, method returns object including data necessary for the Solver module.

Parsing system of equations seems like performing a loop of equation parsing, however the process of evaluating independent and functional variables of the system has to be done regarding all equations. It means that and methods are performed respectively for all equations, but overloaded method is executed at the end with a list of all equations in a system.

- is the exception which is thrown when some part of equation or system of equations is incorrect during the parsing process. This exception includes message which informs about the reason of the exception.

## ODE Solvers

There are a few important classes which form ODE Solver module.

### Solver base class

This is a base class for all specific numerical methods solving ODE’s. Contains set of methods which are common for all implemented solvers. Those methods are basically operations on lists and vectors like addition of vectors along with multiplying by some specific .

### ODE Solver services

The 4th order Runge – Kutta method is implemented as . The Modified Midpoint method is implemented as and Predictor-corrector as . All specific solvers extend mentioned class and provide two methods. One for solving ODE’s and second for solving systems of ODE’s.

**public** Solution solve(Equation equation, **double** step, **double** start, **double** stop) **throws** IncorrectODEEquationException, UnknownFunctionException, UnparsableExpressionException.

This method is responsible for solving ODE given as a parameter. Other parameters : and define domain of the equation. Solving process is controlled by the parameter, which influences on the level of accuracy of the solution. Method returns object which contains results.

## Graph Viewer

This module belongs to the client-side implementation. The objective of this component is to provide graph, presenting solution of the given equation or system of equations. The class which implements inter alia Graph viewer module is . It contains private classes : and which are callbacks for ODE Solver’s methods and respectively. Those private classes set graph properties and values received from ODE Solver as object. Then class from library is created in order to provide visualisation of the 2D graph presenting the solution.

## Equation Store

This module is responsible for data persistence in the application. is a service providing set of methods for storing and querying systems of equations in Datastore. Implementation is based on the Objectify framework which significantly simplifies coding part of this service. provides following interface :

**public** **void** persist(SystemEntity system)

This method stores system of equations in the Datastore.

**public** SystemEntity get(String name)

Gets system of equations specified by the parameter, since is identified by field.

**public** List<SystemEntity> getAll()

This method provides all stored systems of equations as a list. Especially useful for presenting available systems to load for the user.

**public** String remove(String name)

This method enables removing from Datastore, specified by parameter.

**public** **void** removeAll()

Removes all stored systems of equations from Datastore.

## User interface

In order to provide rich and robust user interface Ext GWT (GXT) along with standard GWT components were used. GUI comprises of three panels : and .

### Input Panel

This panel comprises of several subpanels. Combined all together create interface for the user to provide, load or save equation or system of equations. Furthermore user defines domain borders, step and chooses one of three implemented methods solving ODE’s . Additionally some sort of help is included in Input Panel. The user can familiarize himself with format of equations and systems of equations by checking examples. Moreover list of available mathematical expressions is also provided. Finally after providing all necessary information user can press “Compute” button at the bottom of the panel.

In terms of implementation details class extends from Ext GWT library. Vast majority of components, events and listeners in this class is based on GXT library. The structure of subpanels maintained by the from standard GWT library. Moreover, to provide unlimited text fields for the equations was used. This specific table enables creating cells on demand. Such functionality is used to add another text fields when user provides another equations into the input.



Figure 5‑1 Input Panel

### Equation Panel

The objective of this panel is to present equations after parsing process. It is important from the user’s point of view to check how the provided input was interpreted.

Implementation of this panel is based on is used to store rows containing parsed equations, which are represented as objects including equations contents. These images with equations are created with charts API, supported by standard GWT library.



Figure 5‑2 Equation Panel with example of ODE's system

### Graph Panel

This class includes 2D graph utilization, provided by Google Chart Tools 1.1 library. Furthermore, implementation contains quite a few important inner callback classes which drive the main process of the application. and classes implement mechanism of applying received solutions to the object.

Similarly to previously described panels uses a mixture of GXT and GWT components and listeners. Additionally, package (Google Chart Tools library) is widely used in terms of 2D graph implementation.



Figure 5‑3 2D graph presenting solution of sample system of ODE's

# 

# Results

This chapter presents application’s results, with emphasis on Unit testing results. The key part of the project is ODE Solver implementation of numerical methods. However, other components covered with Test Cases are also important in terms of functional requirements of the application.

## Parser Test Case results

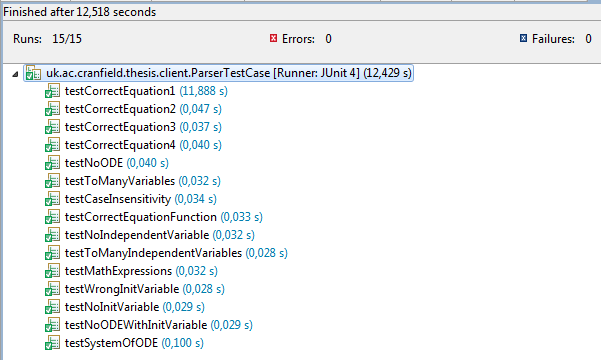


Figure 6‑1 ParserTestCase results

## Solver Test Cases

### RungeKuttaSolverTestCase results

### ModifiedMidpointSolverTestCase results

### PredictorCorrectorSolverTestCase results

## PersistenceTestCase results

## Application’s outputs

# Discussion and conclusion

This chapter reviews all stages of the project along with closing remarks and future work ideas.

## Results analysis

In order to evaluate

## Solvers correctness

The level of correctness of the implemented ODE solvers can be estimated by the level of the accuracy set for Test Cases.

## Quality of implementation

## Limitations

### App Engine’s Limitations

## Problems faced

## Conclusion

## Future work

### N-th order solver

The most important improvement for the future is probably adapting ODE Solvers to -th order systems and equations solving.

### Model – View – Presenter architecture

### Datastore optimization

REFERENCES

[1] Press, William H.; Teukolsky, Saul A.; Vetterling, William T.; Flannery, Brian P. (2007), “Numerical recipes: the art of scientific computing”, Third edition, Cambridge University Press, Cambridge

[2] Kirpekar, Sujit, (2003), “Implementation of the Bulirsch Stoer extrapolation method” Department of Mechanical Engineering, University of California, Berkeley

[3] Guermeur, Daniel; Unruh, Amy; (2010), “Google App Engine Java and GWT Application Development”, Packt Publishing, Birmingham.

[4] Chaganti, Prabhkar, (2007), “Google Web Toolkit : GWT Java Ajax Programming”, Packt Publishing, Birmingham.

[5] <https://developers.google.com/web-toolkit/overview> (accessed 3rd of August 2012)

[6] <https://developers.google.com/appengine/docs/java/runtime#The_Sandbox> (accessed 3rd of August 2012)

[7] <https://developers.google.com/appengine/docs/java/datastore/overview> (accessed 4th of August 2012)

[8] <https://developers.google.com/appengine/docs/java/datastore/entities> (accessed 4th of August)

[9] Barnes, Stuart 2012, Advanced Software Engineering : Software lifecycle models, course notes, Cranfield University.

[10] Nebulon Pty. Ltd, Martin, (2003) “ FDD & Web Development “ <http://www.featuredrivendevelopment.com/node/550> (accessed 5th of August 2012)

[11] Google Developers, (2012), “Command Pattern” <https://developers.google.com/java-dev-tools/codepro/doc/features/patterns/command_pattern>

(accessed 5th of August 2012)

[12] Google Developers, (2012), “Composite Pattern” <https://developers.google.com/java-dev-tools/codepro/doc/features/patterns/composite_pattern> (accessed 5th of August 2012)

[13] Source Making, “Command Design Pattern”, <http://sourcemaking.com/design_patterns/command> (accessed 6th of August 2012)

[14] Cooper, Robert T.; Collins, Charlie E. (2008), “GWT in Practice”, Manning Publications Co. Greenwich.

[15] Source Making, “Composite Design Pattern”, <http://sourcemaking.com/design_patterns/composite> (accessed 6th of August 2012)

[16] Palmer, S. 2009, “An introduction to Feature-Driven Development” <http://agile.dzone.com/articles/introduction-feature-driven> (accessed 6th of August 2012)

[17] Ambler, Scott W. 2011, “Introduction to Test Driven Development”, <http://www.agiledata.org/essays/tdd.html#WhatIsTDD> (accessed 6th of August 2012)

[18] Wellman Daniel, November 2008, “Google Web Toolkit : Writing Ajax Applications Test-First”, published in Better Software magazine.

[19] Barnes, Stuart 2012, Advanced Software Engineering : Validation and Verification, course notes, Cranfield University.

[20] Google Developers, 2012 , “Using Ext GWT (GXT)” <https://developers.google.com/web-toolkit/tools/gwtdesigner/features/gwt/gxt> (accessed 10th of August 2012)

[21] Singh, Yogendra , 2010, ”Google Web Toolkit (GWT) : Uses and Limitations”, <http://yogendrakrsingh.blogspot.com/2010/03/google-web-toolkit-gwt-uses-and.html> (accessed 11th of August 2012)

[22]

APPENDICES

Whilst Heading 1 to Heading 6 can be used to number headings in the main body of the thesis, Heading styles 7–9 have been modified specifically for lettered appendix headings with Heading 7 having the ‘Appendix’ prefix as shown below.

Datastore

Appendix Section (Use Heading 8)

Appendix Subsection (Use Heading 9)

Creating captions in Appendices

If you have chosen to include chapter numbers in your captions then follow the instructions given here to apply the same format to the captions in your appendices. This section explains how to caption the figures and tables in your Appendices, assuming that Heading 7 is numbered “Appendix A” and that the Figures and Tables are going to be labelled ‘Figure A-1’, ‘Figure A-2’, ‘Table B-1’ etc.

You will have to create new, separate labels that look like the ‘Figure’ and ‘Table’ labels you used in the main body of your thesis.

1. Select the **References** tab on the Ribbon then click on **Insert Caption**
2. Click **New Label**. Type **Figure\_Apx** then click **OK**
3. You now have two labels for figures, called **Figure** and **Figure\_Apx**  
   Repeat for table captions.
4. In the **Caption** box, type your caption text
5. Click **Numbering**. Tick **Include chapter numbering** and choose **Heading 7** from the drop-down list of styles and click **OK** twice
6. Your caption should look something like this:

**Figure\_Apx A‑1 This is the caption text for a Figure in the Appendix**

1. Delete the extraneous ‘\_Apx’ from the caption label so it reads:  
   **Figure A‑1 This is the caption text for a Figure in the Appendix**  
   **TIP:** Instead of deleting each ‘\_Apx’ individually use **Find & Replace** to modify all the labels at once.

Creating Lists of Figures and Tables for Appendices

This template already includes a List of Figures and a List of Tables, however you will have to create two new lists for the ‘Figure\_Apx’ and the ‘Table\_Apx’ labels.

1. Place the insertion point on a blank row after the existing List of Figures
2. Select the **Insert Table of Figures** command on the **References** tab of the Ribbon
3. Set the **Caption Label** box to ‘**Figure\_Apx**’ and click **OK**  
   **Note:** Word will put a single blank line between the original and new lists preventing it from appearing as one seamless list. However if you select the blank paragraph between the tables you can hide it by opening the Font dialog box from the Home tab and selecting **Hidden**.
4. Click after the List of Tables and repeat for the Caption Label ‘Table\_Apx’