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

ABSTRACT

Ordinary differential equations are used to model physical and mathematical systems that evolve over time. ODEs are present in many branches of science especially in pure and applied mathematics. Many models base on these equations, so efficient solving methods are essential.

The investigation of numerical routines for solving ODEs is presented in this report. It comprises of the explanation and comparison of the most efficient practical numerical methods available.

This thesis report contains also description of the design, testing and implementation of a web application based on AJAX technologies, solving the systems of ordinary differential equations. Moreover the application devises 2D graphical representation of a solution and enables the user to store and load systems of ODEs. Part of this report is dedicated to technologies used to develop the application, with emphasis on Google Web Toolkit and Google AppEngine . Moreover, details of the software lifecycle model and agile approaches used are referenced as well.

Finally, numerical methods and technology choices are discussed . Additional analysis is provided for the application's results along with problems faced during the development process. The final conclusion is followed by ideas for future improvements.

Keywords:

ODE, Runge-Kutta, Modified Midpoint, Adams-Bashforth-Moulton, Web development, GWT, AppEngine, Datastore, Unit testing, Test Driven Development, Prototyping

ACKNOWLEDGEMENTS

The author would like to thank the supervisor of this thesis project, Dr Peter Sherar. His opinions, advice and especially his guidance were very helpful during the project development. He is also a very cheerful person, so our meetings were pleasant and productive.

Special thanks to Prof Joanna Polanska, second supervisor of this thesis project at Silesian University of Technology. Her commitment to coordination between both Universities enables the fulfilment of the double degree programme.

The author would like to express deep gratitude to his family for their support during his work on this project. Without their support he would not be where he is now.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

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

1. Introduction

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

1.1 Overview

A series of simultaneously ordinary differential equations model physical systems that evolve over time. There is a wide variety of software available for solving ODEs. 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 ODEs and systems of ODEs, 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 the software development process are also significant.

1.2 Aims and objectives

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

I. Parsing ODEs and systems of ODEs entered by the user
II. Solving a series of 1st order simultaneous ordinary differential equations (linear or non-linear) for initial value problem, given along with user defined step and domain.
III. Presenting the solution as a 2D graph.
IV. Storing and loading equations along with parameters entered by the user.

1.3 Motivation

In general, two main motivators of this project are :

I. Discovering the most efficient numerical methods for solving a series of ordinary differential equations.
II. Familiarisation with AJAX applications development and Google AppEngine along with Datastore.

1.4 Literature review

The literature review gives an insight into the numerical methods solving ODEs, 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 ODEs [16], [19] are :

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

An extensive study of GWT and the AJAX approach [4], [5], [10], [15], [20] noted that utilization 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 a very good idea in terms of reliability, scalability and robustness [15]. Furthermore, availability of App Engine's Datastore is described as well integrated and with reliable storage [8], [9] .

In terms of software lifecycle models , the Prototyping Model seems the most suitable for this project [2] . From the Agile approach flavours , TDD and FDT are suitable to a certain degree for this thesis [1], [3], [17], [18], [23].

Design decisions were made based on articles from the Google Developers page, containing detailed descriptions of applications developed using Google technologies. [11], [13], [14], Moreover, design patterns and architectural details of Google Web Toolkit were taken into account as well. [6], [7], [21], [22].

The testing methodology applied for GWT applications is based on Unit Testing and cross-browser UI testing [12], [23].

1.5 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 the results achieved in this thesis project
- Discussions and conclusions of the project are presented in chapter 7.

2 Mathematical background

This chapter demonstrates knowledge about recommended practical numerical methods for solving ODEs. The presented literature review gives an insight into major types of numerical routines taking into account efficiency and accuracy.

2.1 Ordinary differential equations

An ordinary differential equation is an equation that contains function of one independent variable and its derivatives. General definition of the order :

$$y^{(n)} = f(x, y, y', y'', y''', \ldots, y^{n-1}) \qquad \textbf{(2-1)}$$

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 [19] :

- Initial value problems
  All y_i are given at some starting point x_0 and it is desired to find the $y_i's$ at some final point x_n or list of points with specified intervals, for example $x_i = x_o + ih$.
- Two-point boundary value problems
  Where boundary conditions are specified at more than one point. For example at starting and final point.

The general definition of the system of ODEs of order :

$$\boldsymbol{y}^{(n)} = \boldsymbol{F}(x, \boldsymbol{y}, \boldsymbol{y}', \boldsymbol{y}'', \boldsymbol{y}''', \ldots, \boldsymbol{y}^{\boldsymbol{n-1}}) \qquad \textbf{(2-2)}$$

$$\begin{pmatrix} y_1^{(n)} \\ y_2^{(n)} \\ . \\ . \\ . \\ y_m^{(n)} \end{pmatrix} = \begin{pmatrix} F_1(x\,, y, y', y'', y''', \dots, y^{n-1}) \\ F_2(x\,, y, y', y'', y''', \dots, y^{n-1}) \\ . \\ . \\ . \\ F_m(x\,, y, y', y'', y''', \dots, y^{n-1}) \end{pmatrix} \quad \textbf{(2-3)}$$

2.2 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 .

$$\mathrm{y}'(\mathrm{x}) = f\big(x\,, y(x)\big)\,, \qquad y(x_0) = y_0 \quad \textbf{(2-4)}$$

$$y_{n+1} = y_n + hf(x_n, y_n) \quad \textbf{(2-5)}$$

2.3 ODE numerical routines

This section describes types of practical numerical methods for solving ODEs including Runge-Kutta methods, Bulirsch-Stoer, Rosenbrock methods and Predictor-corrector methods.

2.3.1 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 f function .

$$y' = f(x, y), \qquad y(x_0) = y_0 \quad \textbf{(2-6)}$$

Then using 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 ($\leq 10^{-5}$) and evaluation of the f function is not too expensive. There are a few kinds of Runge-Kutta methods : 2nd order method, (called midpoint method), 4th order method and also method with adaptive stepsize. [19]

2.3.1.1 4th Order Runge-Kutta

The most often used Runge-Kutta method is the fourth order formula. In general it is superior to 2nd order method, however high order does not always mean high accuracy. The 4th order method requires four evaluations of the f function. [19]

$$k_1 = hf(x_n, y_n) \tag{2-7}$$

$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_1) \tag{2-8}$$

$$k_3 = hf(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_2) \tag{2-9}$$

$$k_4 = hf(x_n + h, y_n + k_3) \tag{2-10}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5) \tag{2-11}$$

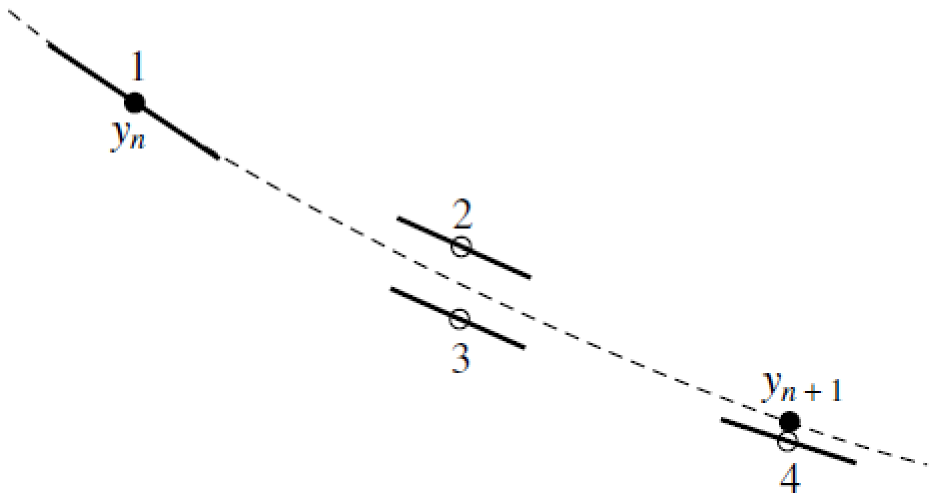


Figure 2-1 4th Order Runge-Kutta method

During a single step, the derivative is evaluated four times. Once at the initial point, (1) twice at the midpoints (2)(3) and once at the trial endpoint (4). The final function value is calculated on the basis of these derivatives. [19]

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. [19]

2.3.1.2 Runge-Kutta for a system of differential equations

The algorithm of 4th order Runge-Kutta method for solving systems of ODEs comprises of the following steps :

$$y' = f(x, y(x), z(x), \dots), \qquad z' = g\big(x, y(x), z(x)\big), \qquad \dots \tag{2-12}$$

$$k_1 = hf(x_n, y_n, z_{n,}, \dots), \qquad l_1 = hg(x_n, y_n, z_{n,}, \dots), \qquad \dots \tag{2-13}$$

$$k_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_1, z_n + \frac{1}{2}l_1, \ldots\right),$$ **(2-14)**

$$l_2 = hg(x_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}k_1,\ z_n + \tfrac{1}{2}l_1, \ldots\ ),\ \ldots$$

$$k_3 = hf\left(x_n + \frac{h}{2},\ y_n + \frac{1}{2}k_{2,}\ z_n + \frac{1}{2}l_2, \ldots\right),\ l_3 = hf\left(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_2,\ z_n + \frac{1}{2}l_2, \ldots\right), \ldots$$ **(2-15)**

$$k_4 = hf(x_n + h, y_n +\ k_3, + l_3 + \cdots),\ l_4 = hf(x_n + h, y_n +\ k_3, + l_3 + \cdots), \ldots$$ **(2-16)**

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 +\ k_4 + \cdots) + O(h^5),$$ **(2-17)**

$$z_{n+1} = y_n + \frac{1}{6}(l_1 + 2l_2 + 2l_3 +\ l_4\ + \cdots) + O(h^5), \qquad \ldots$$

2.3.1.3 Runge-Kutta with adaptive stepsize

The purpose of the 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. Taking a few great strides instead of small steps should speed through such undifferentiated interval, what may result in significant gain in efficiency. The idea of the adaptive stepsize method is to control the size of the step and increase it when possible maintaining the required level of accuracy. It is important to estimate the truncation error to control the accuracy level while increasing the step size. Obviously the calculation of this information will add to the computational overhead, however it is a profitable investment in terms of efficiency. [19]

2.3.2 The modified midpoint

The modified midpoint method is a second order method like the 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 $y(x)$ values from a point x to a point $x+H$ by a sequence of n substeps each of size h. [19] Where

$$h = \frac{H}{n} \tag{2-18}$$

The total number of function evaluations required by this method is $n+1$. The formulas essential to provide the solution for $y\ (x+H)$ are as follows

$$k_0 = y(x_0) \tag{2-19}$$

$$k_1 = k_0 + hf(x, k_0) \tag{2-20}$$

$$k_{m+1} = k_{m-1} + 2hf(x+mh, k_m) \qquad for \quad m = 1,2,\dots,n-1 \tag{2-21}$$

$$y(x+H) \approx y_n = \frac{1}{2}\ [\, k_n + k_{n-1} + hf(x+H, k_n)] \tag{2-22}$$

The y_n is the final approximation to $y(x+H)$ whereas z_m represents intermediate approximations calculated along in steps of h.

2.3.2.1 Modified midpoint for a system of differential equations

The form of the Modified midpoint method for solving system of ODEs is presented below

$$k_0 = y(x_0), \ \ l_0 = z(x_0), \qquad \dots \tag{2-23}$$

$$k_1 = k_0 + hf\big(x, k_0, l_{0,} \dots\big), \qquad l_1 = l_0 + hg\big(x, k_{0\,,} l_{0,} \dots\big), \qquad \dots \tag{2-24}$$

$$k_{m+1} = k_{m-1} + 2hf(x + mh, k_m, l_m, \dots)\ , \tag{2-25}$$

$$l_{m+1} = l_{m-1} + 2hg(x + mh, k_m, l_m, \dots), \qquad \dots \quad for \quad m = 1,2, \dots, n-1$$

$$y(x+H) \approx y_n = \frac{1}{2}\ [\,k_n + k_{n-1} + hf(x+H, k_n)]\ , \tag{2-26}$$

$$z(x+H) \approx z_n = \frac{1}{2}\ [\,l_n + l_{n-1} + hg(x+H, l_n)], \qquad \dots$$

2.3.3 Richardson extrapolation

The 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.

2.3.3.1 Bulirsch – Stoer

The idea of Burlisch-Stoer method is to perform I iterations of the modified midpoint method. Each iteration uses various number of substeps (n) for the modified midpoint method and ends up with polynomial extrapolation of the given values. [16], [19] Bulirsh and Stoer originally proposed the following sequence of substeps :

$$n = 2, 4, 6, 8, 12, 16, 24, \dots, [n_j = 2n_{j-2}] \tag{2-27}$$

However the sequence discovered by Deuflhard is usually more efficient :

$$n = 2, 4, 6, 8, 10, 12, 14, \dots, [n_j = 2j] \tag{2-28}$$

In terms of number of iterations (I) usually 8 gives satisfactory results.

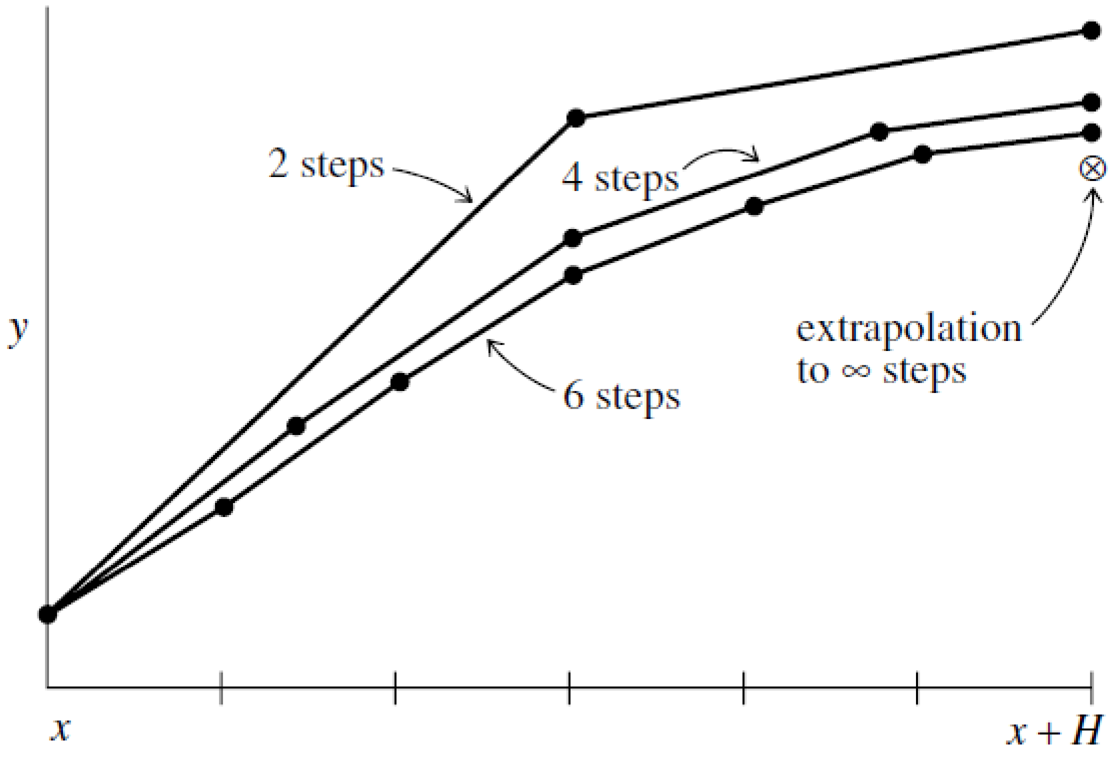


Figure 2-2 Richardson extrapolation used in the Burlisch-Stoer method with substep n = 2,4,6 [19]

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

$$\begin{array}{llll}
T_{11} & & & \\
T_{21} & T_{22} & & \\
T_{31} & T_{32} & T_{33} & \\
\ldots & \ldots & \ldots & T_{kk}
\end{array}$$

Figure 2-3 Aitkens-Neville polynomial extrapolation tableau [19]

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

$$T_{j,1} = y_j \tag{2-29}$$

Where y_j is $y(x_n + H)$ computed with the stepsize $h_j = {}^{H}/_{n_j}$

Successive columns can be filled by using following recurrence :

$$T_{j,k} = T_{j,k-1} + \frac{T_{j,k-1} - T_{j-1,k-1}}{(\frac{n_j}{n_{j-k+1}})^2 - 1} \qquad \textbf{(2-30)}$$

The final solution which is $T_{k,k}$ can be achieved after performing each successive iteration [16]

2.3.4 Rosenbrock

Rosenbrock methods are competitive with other numerical ODE integrators in terms of moderate accuracies (tolerances of order $10^{-4} - 10^{-5}$). Moreover these methods remain reliable for more stringent parameters [19] . The main formula of the Rosenbrock method is as follows :

$$y(x_0 + h) = y_0 + \sum_{i=1}^{s} b_i k_i \qquad \textbf{(2-31)}$$

Where k_i corrections are found after solving following s linear equations :

$$(1 - \gamma h f') * k_i = hf(y_0 + \sum_{j=1}^{i-1} \propto_{ij} k_j) + hf' \sum_{j=1}^{i-1} \gamma_{ij} k_j \, , \qquad i = 1, \dots, s \qquad \textbf{(2-32)}$$

The coefficients $\gamma,\ b_i, \propto_{ij}\ and\ \gamma_{ij}$ are fixed and Jacobian matrix is denoted by f'. [19]

2.3.5 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 the Bulirsch-Stoer method dominates. For moderate precision problems Runge-Kutta methods dominate. 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 [19].

Considering the 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 x. However for an ODE, the “integrand” (which is right-hand sided) depends both on x and dependent variables y [19]. So in order to advance the solution of $y' = f(x, y)$ from x_n to x we have :

$$y(x) = y_n + \int_{x_n}^{x} f(x', y)\, dx' \tag{2-33}$$

According to $f(x, y)$ a multistep approach is approximated by a polynomial passing through several previous points x_n , $x_{n-1}, \dots$ and possibly through x_{n+1} .

The formula that is evaluating the integral (2-18) at $x = x_{n+1}$ is then of the form:

$$y_{n+1} = y_n + h(\beta_0 y'_{n+1} + \beta_1 y'_n + \beta_2 y'_{n-1} + \beta_3 y'_{n-2} + \cdots) \tag{2-34}$$

where $y'_n = f(x_n, y_n)$

There is a method called $functional\ iteration$ which solves an implicit formula of the form (2-19) for y_{n+1} . Such method is called a $predictor\ step$. The idea of this method is to take initial guess for y_{n+1} , then insert it into the right-hand side of (2-19) and get updated value of y_{n+1}. In order to get initial value of y_{n+1} we have to extrapolate the polynomial fit to the derivative from the previous points to the new point x_{n+1} . The next stage of the solving process is made by $corrector\ step$ which is using the prediction step’s value of y_{n+1} to $interpolate$ the derivative [19]. In conclusion Predictor-corrector method comprises of three separate processes :

- Predictor step
- Evaluation of the derivative y'_{n+1} from the latest value.

- Corrector step

2.3.5.1 Adams-Bashforth-Moulton

Probably the most popular $predictor - corrector$ method is $Adams - Bashforth - Moulton$ method. This method has good stability properties [19] . The predictor part is called The Adams-Bashforth :

$$y_{n+1} = y_n + \frac{h}{12}(23f(x_n, y_n) - 16f(x_{n-1}, y_{n-1}) + 5f(x_{n-2}, y_{n-2})) \tag{2-35}$$

The Adams-Moulton part is the corrector :

$$y_{n+1} = y_n + \frac{h}{12}(5f(x_{n+1}, y_{n+1}) + 8f(x_n, y_n) - f(x_{n-1}, y_{n-1})) \tag{2-36}$$

In order to perform this method it is necessary to evaluate y_1 and y_2 having initial value (y_0) . To achieve this you have to use other numerical routine to evaluate two steps. Usually 4th Order Runge-Kutta method is used for this.

For systems of ordinary differential equations, predictor and corrector steps are transformed into following equations :

Predictors :

$$\begin{aligned} y_{n+1} = y_n + \frac{h}{12}\big(&23f(x_n, y_n, z_n, u_n, \ldots) - 16f(x_{n-1}, y_{n-1}, z_{n-1}, u_{n-1}, \ldots) \\ &+ 5f(x_{n-2}, y_{n-2}, z_{n-2}, u_{n-2}, \ldots)\big) \end{aligned} \tag{2-37}$$

$$\begin{aligned} z_{n+1} = z_n + \frac{h}{12}\big(&23f(x_n, y_n, z_n, u_n, \ldots) - 16f(x_{n-1}, y_{n-1}, z_{n-1}, u_{n-1}, \ldots) \\ &+ 5f(x_{n-2}, y_{n-2}, z_{n-2}, u_{n-2}, \ldots)\big) \end{aligned}$$

$$\begin{aligned} u_{n+1} = u_n + \frac{h}{12}\big(&23f(x_n, y_n, z_n, u_n, \ldots) - 16f(x_{n-1}, y_{n-1}, z_{n-1}, u_{n-1}, \ldots) \\ &+ 5f(x_{n-2}, y_{n-2}, z_{n-2}, u_{n-2}, \ldots)\big) \end{aligned}$$

…

Correctors :

$$y_{n+1} = y_n + \frac{h}{12}\Big(5f\big(x_{n+1}, y_{n+1}, z_{n+1}, u_{n+1}, \ldots\big) + 8f\big(x_n, y_n, z_n, u_n, \ldots\big) - f(x_{n-1}, y_{n-1}, z_{n-1}, u_{n-1}, \ldots)\Big) \qquad \textbf{(2-38)}$$

$$z_{n+1} = z_n + \frac{h}{12}\Big(5f\big(x_{n+1}, y_{n+1}, z_{n+1}, u_{n+1}, \ldots\big) + 8f\big(x_n, y_n, z_n, u_n, \ldots\big) - f(x_{n-1}, y_{n-1}, z_{n-1}, u_{n-1}, \ldots)\Big)$$

$$u_{n+1} = u_n + \frac{h}{12}\Big(5f\big(x_{n+1}, y_{n+1}, z_{n+1}, u_{n+1}, \ldots\big) + 8f\big(x_n, y_n, z_n, u_n, \ldots\big) - f(x_{n-1}, y_{n-1}, z_{n-1}, u_{n-1}, \ldots)\Big)$$

…

3 Technologies, Methodologies and Application Design

This section is a combination of methodologies chosen for this thesis project and design part. The chapter presented here includes a description of the software lifecycle model which was chosen for this project. Moreover, a description of the agile approach applied in this thesis is also given. The AJAX approach is taken into account along with Google Web Toolkit. Also Google App Engine’s possibilities for maintaining web-based applications are described.

3.1 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 a 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. A good example of this is when users are trying to log in to their e-mail application. After the authentication process the only real job of the application is to replace a login link by some user welcoming message. The following figure presents what happens with the web traffic for AJAX and a HTML-only non-AJAX applications in the described scenario. [4], [5], [15]

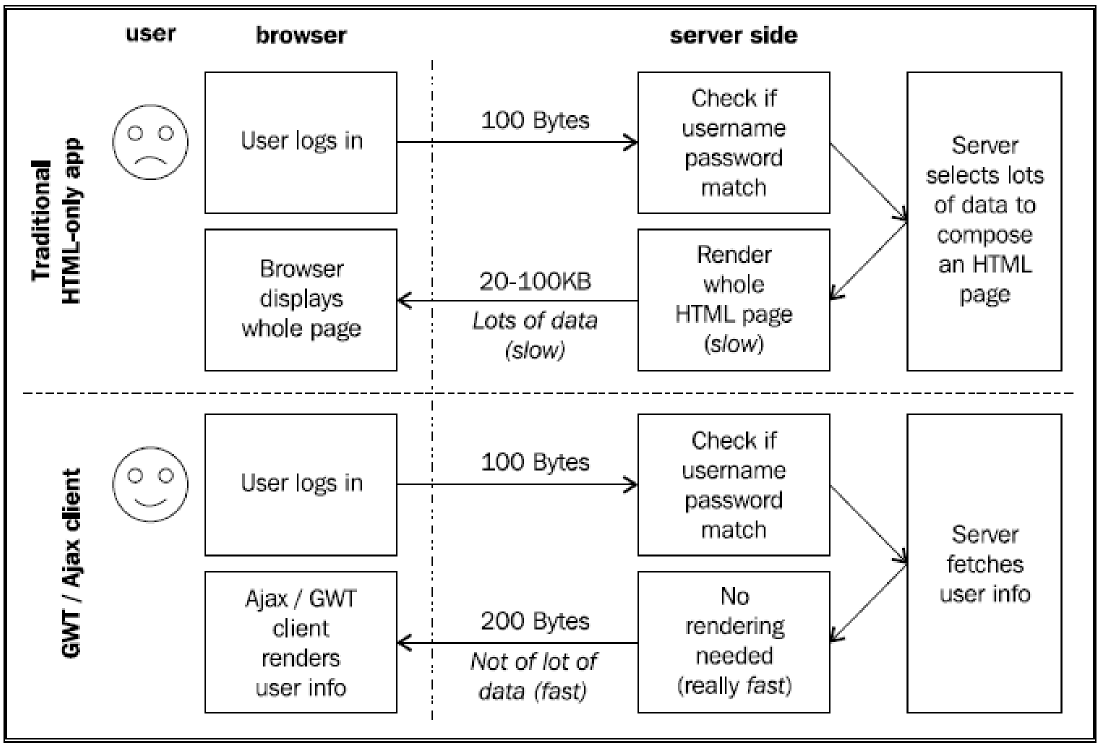

user
browser
server side
Traditional HTML-only app
User logs in
100 Bytes
Check if username password match
Server selects lots of data to compose an HTML page
Browser displays whole page
20-100KB
Lots of data (slow)
Render whole HTML page (slow)
GWT / Ajax client
User logs in
100 Bytes
Check if username password match
Server fetches user info
Ajax / GWT client renders user info
200 Bytes
Not of lot of data (fast)
No rendering needed (really fast)

Figure 3-1 Comparison of AJAX light traffic needs versus legacy HTML applications [15]

3.2 Technologies used

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

The **Google App Engine** (SDK version 1.6.4) is responsible for running the application and storing data on the Datastore. It is also responsible for automatic scaling and load balancing [4], [5], [15].

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

The **Exp4j** library is applied in the implementation of ODE Solvers. 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 $GWT - Visualisation$ 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.

3.2.1 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 $XMLHttpRequest$ 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 a web application [4], [5], [15].

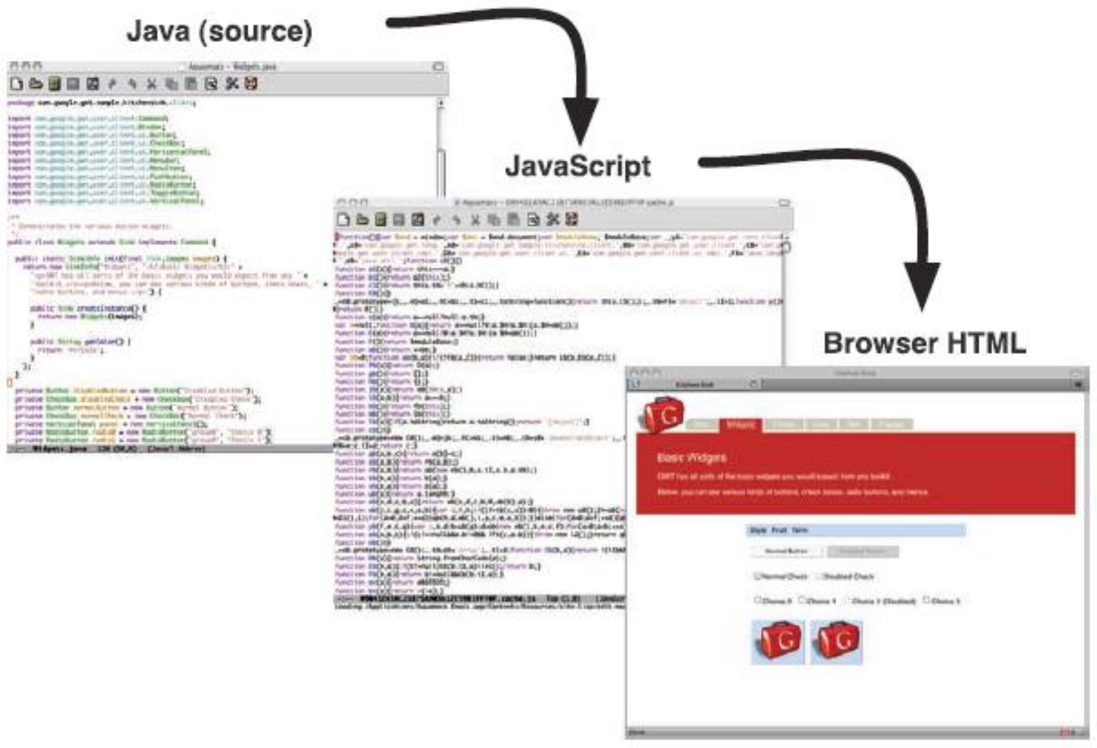


Figure 3-2 An overview of GWT approach

GWT consists of four main components listed below, which provide functionalities for writing AJAX applications [4], [5], [15].

- **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 $java.lang$, $java.util$, 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.

3.2.1.1 GWT Remote Procedure Calls

In GWT, client-server communication is enabled using $Remote\ Procedure\ Call$ mechanism. Each side of the process implements GWT service interface for sending and receiving special GWT serialized data. The server side exposes resources and client side invokes those resources asynchronously [5].

3.2.2 Google AppEngine

The Google App Engine is a PaaS cloud computing platform for developing and hosting web applications using Google’s servers and infrastructure. This 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 [15].

The App Engine distributes requests for applications across multiple web servers to prevent applications from interfering with one 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 [15].

In terms of limitations , an App Engine application cannot :

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

3.2.3 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 *entities*. It was designed to manage scaling to very large data sets in distributed architecture. For this reason Datastore is different than traditional relational databases [8].

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 [8], [9].

- **Objectify**

Very simple and convenient interface to the App Engine Datastore. This 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, the framework chosen for this project is $Objectify$.

Each $entity$ has one or more named properties . Each property can have one or more values. The available data types defined for Datastore are presented in Appendix A.

There are following limitations concerning Datastore :

Table 3-1 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

3.2.4 App Engine Administration console

Google AppEngine provides Administration console for every AppEngine account. It is very useful for managing and monitoring deployed applications. Moreover it is possible to provide basic configurations, set performance options and manage versions of deployed applications. Furthermore Datastore configuration options are also available there. An example of such console view is available in Appendix B.

3.3 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. The application is divided on the server-side and client-side implementation.

The application is divided into several modules :

- **Equation Parser**
  This module is responsible for parsing the user’s input into ODE objects or System of ODEs objects .
- **ODE Solver**
  Contains implementations of ODE numerical routines. The Solver operates on objects returned by the Equation Parser.
- **Graph Viewer**
  This component is responsible for presenting the 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 querying Equations on the Datastore.

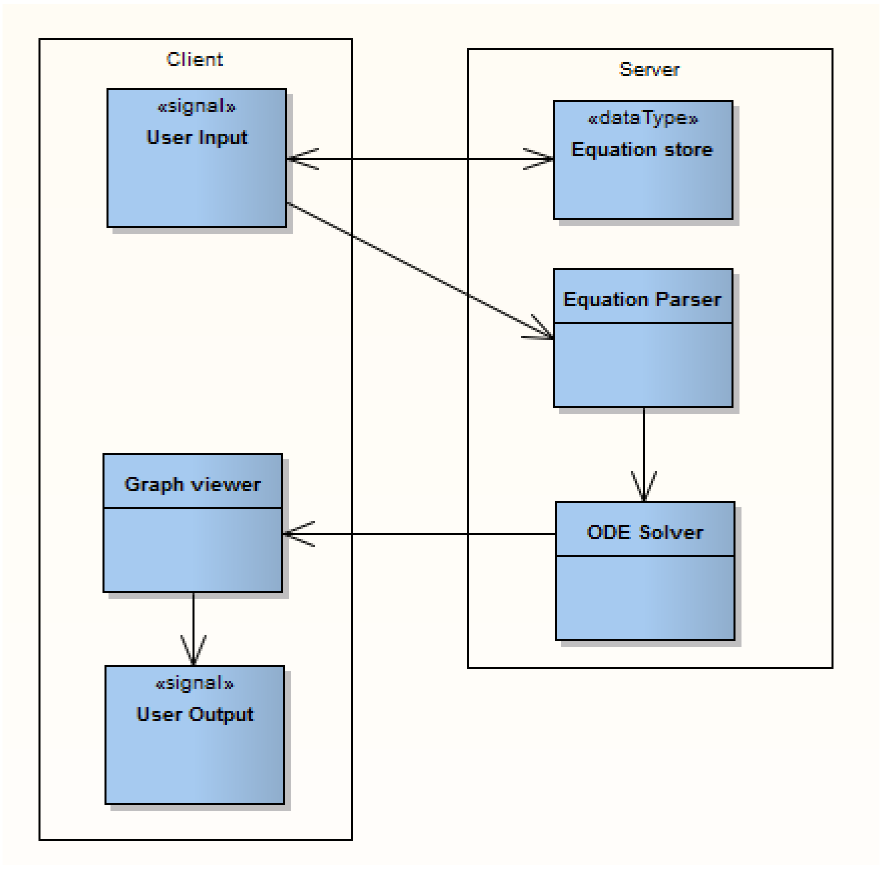


Figure 3-3 Application architecture overview

According to good software design and development practices, the 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.

3.4 Design patterns

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

Command and Composite are design patterns used in GWT.

The 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 needs to be done and passed to another client that has required resources to perform this operation.[6], [21]

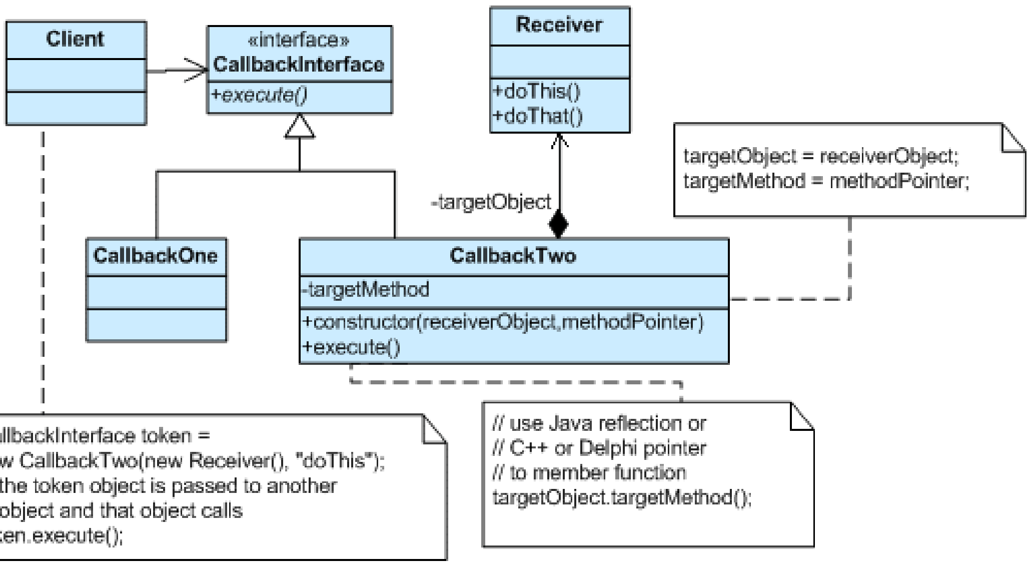

Client
«interface»
CallbackInterface
+execute()
Receiver
+doThis()
+doThat()
targetObject = receiverObject;
targetMethod = methodPointer;
-targetObject
CallbackOne
CallbackTwo
-targetMethod
+constructor(receiverObject,methodPointer)
+execute()
CallbackInterface token =
new CallbackTwo(new Receiver(), "doThis");
... the token object is passed to another
... object and that object calls
token.execute();
// use Java reflection or
// C++ or Delphi pointer
// to member function
targetObject.targetMethod();

Figure 3-4 Example of Command Pattern structure [21]

Composite is a structural pattern, which composes objects into a 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. [7], [22]

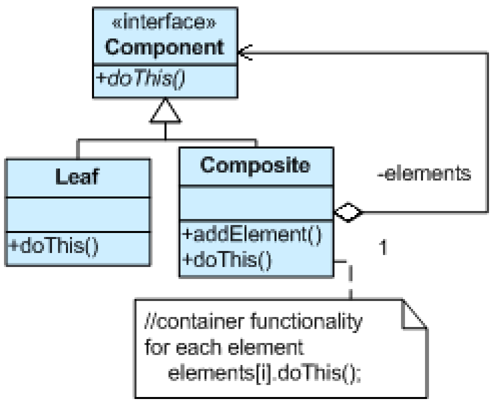


Figure 3-5 Example of Composite Pattern structure [22]

3.5 The Prototyping Model

The prototyping lifecycle model is an example of an Evolutionary model. In general a prototype means a version of the software used to test different solutions to the problem. Such an approach allows better understanding of the ways solutions 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 [2] .

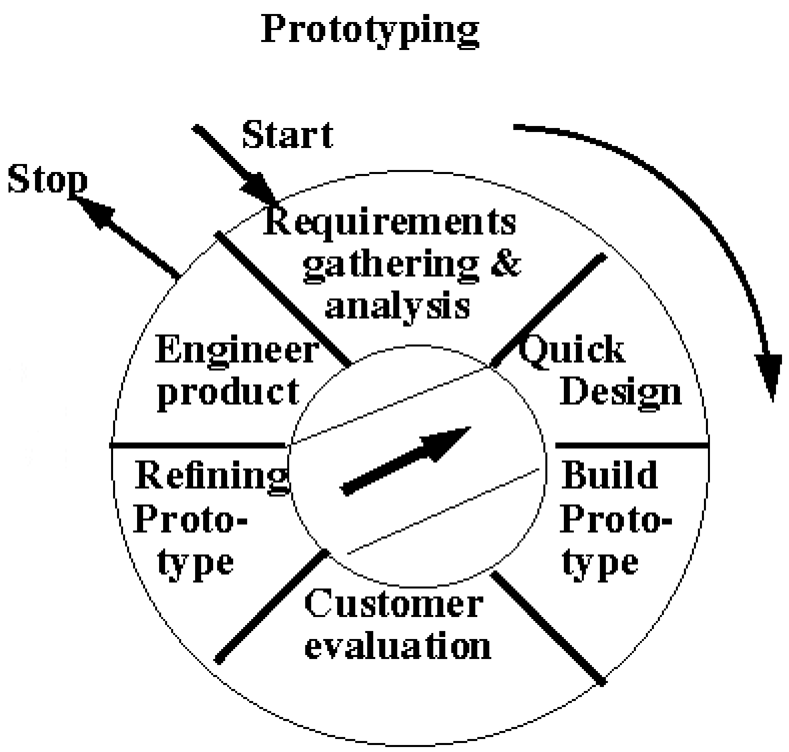


Figure 3-6 The prototyping lifecycle

During the 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 a quick design phase is performed. Another step is to develop a prototype according to the prepared design. The next step is the evaluation of the developed prototype in agreement with the thesis supervisor. Another step is gathering comments about the prepared prototype and refining it by repeating the cycle from the design part. This process is repeated as long as feature requires improvements and corrections .

3.6 Agile approach

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

3.6.1 Feature Driven Development

FDD is an agile approach which treats the 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 the 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. [17]

3.6.2 Test Driven Development

Test Driven Development is also one of the Agile methodologies used in this project. In the traditional testing approach tests are created after developing some piece of software. TDD turns traditional approach around. Instead of writing the functional code first and then testing the code as an afterthought , you write test code before the 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 it requires great discipline. It is easy to “slip” and write the functional code avoiding new tests. [1], [2]

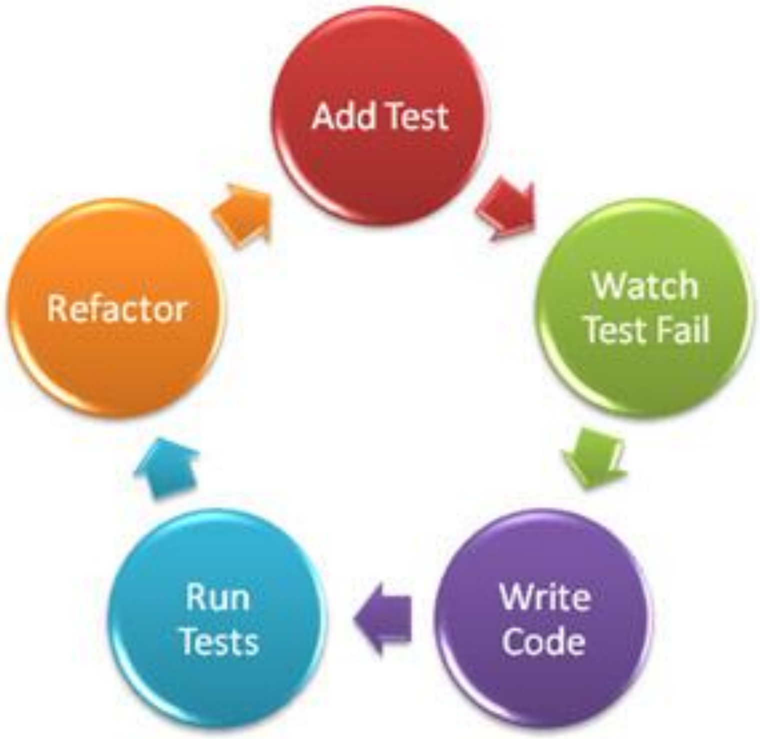


Figure 3-7 Test Driven Development cycle

3.7 Software versioning

In terms of software versioning and revision control, whole development process was supported by Google Project Hosting, which provides fast, reliable and open source hosting service. The *Eclipse* IDE was used to develop the software and *Subclipse* plugin was supporting Subversion system within this IDE.

4 Testing

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

4.1 Unit testing

This represents the lowest level of testing in a 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.[3]

4.1.1 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 $GWTTestCase$ which is a subclass of JUnit's $TestCase$. 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 $TestCase$ class and $GWTTestCase$ class, only $getModuleName$ is an additional method which must be implemented. This method returns a string containing the name of the GWT code module as defined in module configuration file of the application. Running test cases extending $GWTTestCase$ 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. [23]

4.1.2 TDD approach

The 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 the implementation phase. The idea was to define set of specific ODEs checking the correctness of the solvers. [1]

4.1.3 Test Cases

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

4.1.3.1 ParserTestCase

This 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 ODEs. 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 ODEs containing incorrect and correct states of variables in an equation. There are tests performed containing too many variables in an equation. Moreover an equation which has no independent variables is verified 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 the 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`

This 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 an exception is being thrown when initial values of the equations are omitted. Additionally another test is performed to check the situation when initial values concern irrelevant variables.

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 ODEs.

4.1.3.2 SolverTestCase

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

```
public void gwtSetUp()
```

Which contains specific Solver's initialization and accuracy setting.

In terms of $SolverTestCase$, it comprises of the following tests :

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

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

Tests verify 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 the analytic solution of the ODE. Tests check number of solution values included in $Solution$ 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 the accuracy of non-linear ODEs. Both tests are performed with the domain [0.0 , 1.0] and step 0.1. The solution compared in these unit tests is achieved using $WolframAlpha$'s $Mathematica$.

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 $test1OrderSystem$ and $test1OrderSystem3$ check accuracy of a system which comprises of respectively two and three ODEs. The domain for all equations is defined as [0.0 , 1.0] with step = 0.1 . All initial values for the functions , z and u (in the second system) are 0.0 .

4.2 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 the application is treated as a “black-box” and testing process is based on performing use cases of the application.

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

The 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

4.2.1 Persistence tests

This set of tests is responsible for testing $Equation\ Store$ component. The tests performed check operations of storing and querying data on Datastore. There is a reason why these tests are not performed as a separated TestCase comprising of unit tests. It is possible to perform local testing of Datastore by using $LocalServiceTestHelper$ class from $appengine - testing - 1.4.0.jar$ library. However those tests can be performed on Datastore interface only. It means that it is not possible to perform implemented services using Datastore and objectify locally. This is the reason why $Equation\ store$ module is tested using black-box methodology. Several tests are performed to cover all possible scenarios related to Datastore usage as stated below :

- store an equation
- load an equation
- store system of equations
- load system of equations
- list all systems of equations
- remove system of equations
- remove all systems of equations

5 Implementation

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

5.1 The anatomy of the project

This project is structured according to the 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 $model$ and $exception$ packages, which contain classes used in both implementations.

- **ThesisAE.gwt.xml**
  GWT module definition file. This declares several primary elements : inherited modules, servlet deployments, compiler plugins and entry points.

- **web.xml**
  Contains mappings for servlets specified in the $ThesisAE.gwt.xml$.

- **ThesisAE.java**

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

5.2 Model

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

5.2.1 Equation

It is a very important class, which represents the equation in the application. The equation is created after the parsing stage, and ODE solvers are operating on it during solving process. This class representation is based on the 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 *Equation* object depending on the data available to create such an object.

5.2.2 System

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

```
private List<Equation> equations
```

Except standard setters and getters, the $System$ class provides :

```
public void addEquation(Equation equation)
```

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

5.2.3 SystemEntity

This class is an entity class represented in AppEngine's Datastore. It is an expanded version of the $System$ 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 the 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 $SystemEntity$ was last modified on the Datastore.

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

5.2.4 Solution

The *Solution* 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
```

The 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.

5.3 Equation Parser

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

5.3.1 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 *Equation* object which includes all equation data necessary for the ODE Solver.

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

- *parseInitialValues*

  Parses initial values

- *parseFunctionVariable*

  Retrieves character which stands for a function in parsed equation.

- *parseIndependentVariable*

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

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

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

Parsing a system of equations seems like performing a for 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 $parseInitialValues$ and $parseFunctionVariable$ methods are performed respectively for all equations, but the overloaded $parseIndependentVariable$ method is executed at the end with a list of all equations in a system.

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

5.4 ODE Solvers

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

5.4.1 Solver base class

This is a base class for all specific numerical methods solving ODEs and contains a 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 h .

5.4.2 ODE Solver services

The 4th order Runge – Kutta method is implemented as $RungeKuttaSolverServiceImpl$. The Modified Midpoint method is implemented as $ModifiedMidpointSolverServiceImpl$ and Predictor-corrector as $PredictorCorrectorSolverServiceImpl$. All specific solvers extend mentioned $Solver$ class and provide two methods. One for solving ODEs and second for solving a systems of ODEs.

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

This method is responsible for solving ODE given as an $equation$ parameter. Other parameters : $start$ and $stop$ define domain of the equation. The solving process is controlled by the $step$ parameter, which influences the level of accuracy of the solution. The method returns $Solution$ object which contains results.

5.5 Graph Viewer

This module belongs to the client-side implementation. The objective of this component is to provide a graph presenting the solution of the given equation or system of equations. The class which implements inter alia Graph viewer module is $GraphPanel$. It contains private classes : $EquationSolverCallback$ and $SystemSolverCallback$ which are callbacks for ODE Solver's methods $solve$ and $solveSystem$ respectively. Those private classes set graph properties and values received from ODE Solver as $Solution$ object. Then $LineChart$ class from $Google\ Chart\ Tools$ library is created in order to provide visualisation of the 2D graph presenting the solution.

5.6 Equation Store

This module is responsible for data persistence in the application. $SystemPersistenceServiceImpl$ is a service providing a 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. $SystemPersistenceServiceImpl$ provides the 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 $name$ parameter, since $SystemEntity$ is identified by $name$ field.

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

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

```
public String remove(String name)
```

This method enables removing $SystemEntity$ from Datastore, specified by $name$ parameter.

```
public void removeAll()
```

Removes all stored systems of equations from Datastore.

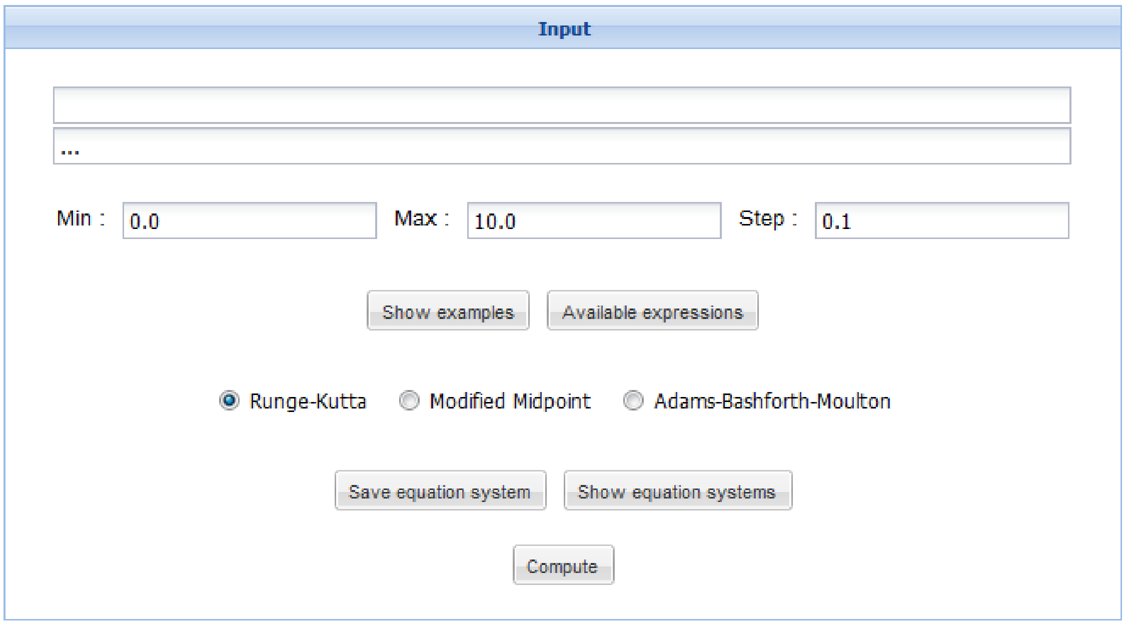
5.7 User interface

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

5.7.1 Input Panel

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

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


Input

...

Min : 0.0 Max : 10.0 Step : 0.1

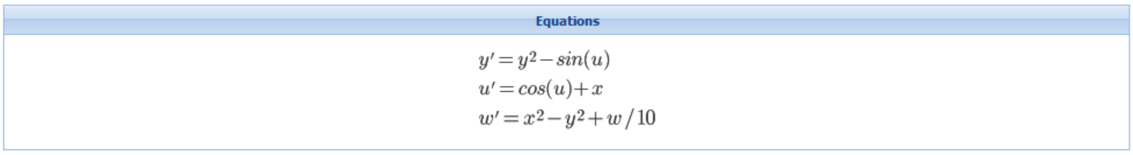
[✓ Runge-Kutta] Modified Midpoint Adams-Bashforth-Moulton

Figure 5-1 Input Panel

5.7.2 Equation Panel

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

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



$$y' = y^2 - sin(u)$$
$$u' = cos(u) + x$$
$$w' = x^2 - y^2 + w/10$$

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

5.7.3 Graph Panel

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

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

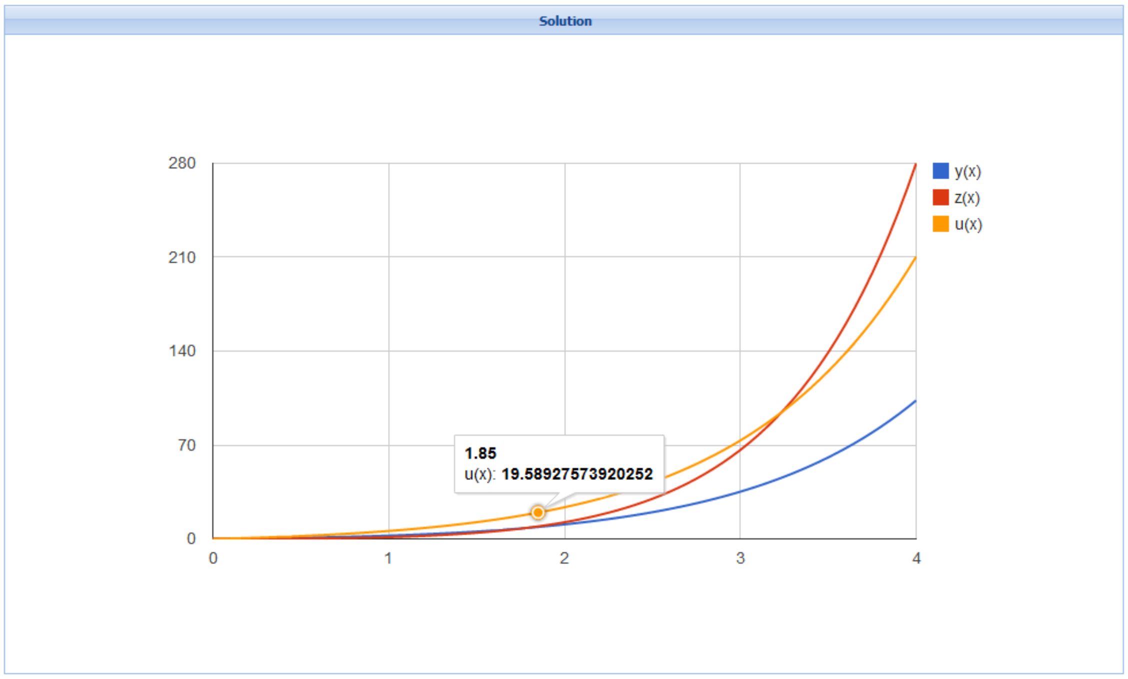
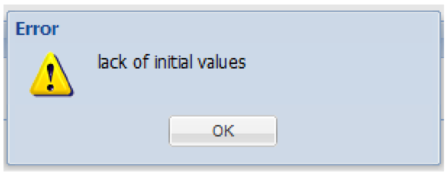


Figure 5-3 2D graph presenting solution of sample system of ODE's

5.7.4 Errors handling

An example of an error message presented on a dialog.

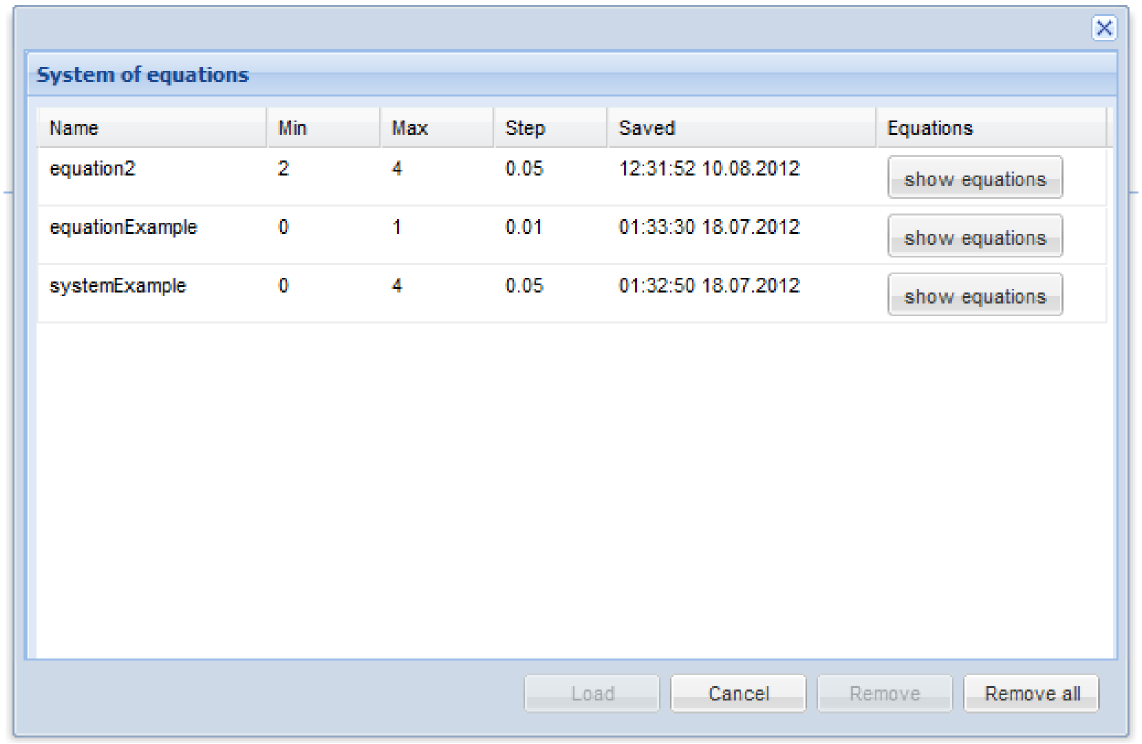

Error

lack of initial values

Figure 5-4 Example error dialog

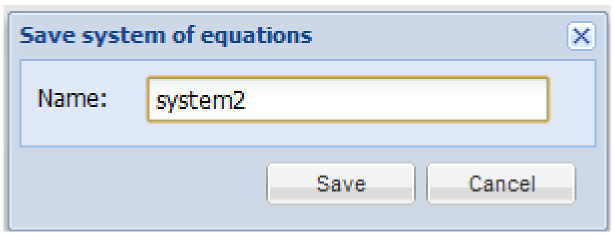
5.7.5 Equation Store GUI components

All operations connected with $Equation\ store$ and Datastore in general, can be performed by the user with following components of the application.



Name	Min	Max	Step	Saved	Equations
equation2	2	4	0.05	12:31:52 10.08.2012	show equations
equationExample	0	1	0.01	01:33:30 18.07.2012	show equations
systemExample	0	4	0.05	01:32:50 18.07.2012	show equations

Figure 5-5 Developed dialog for loading systems of equations



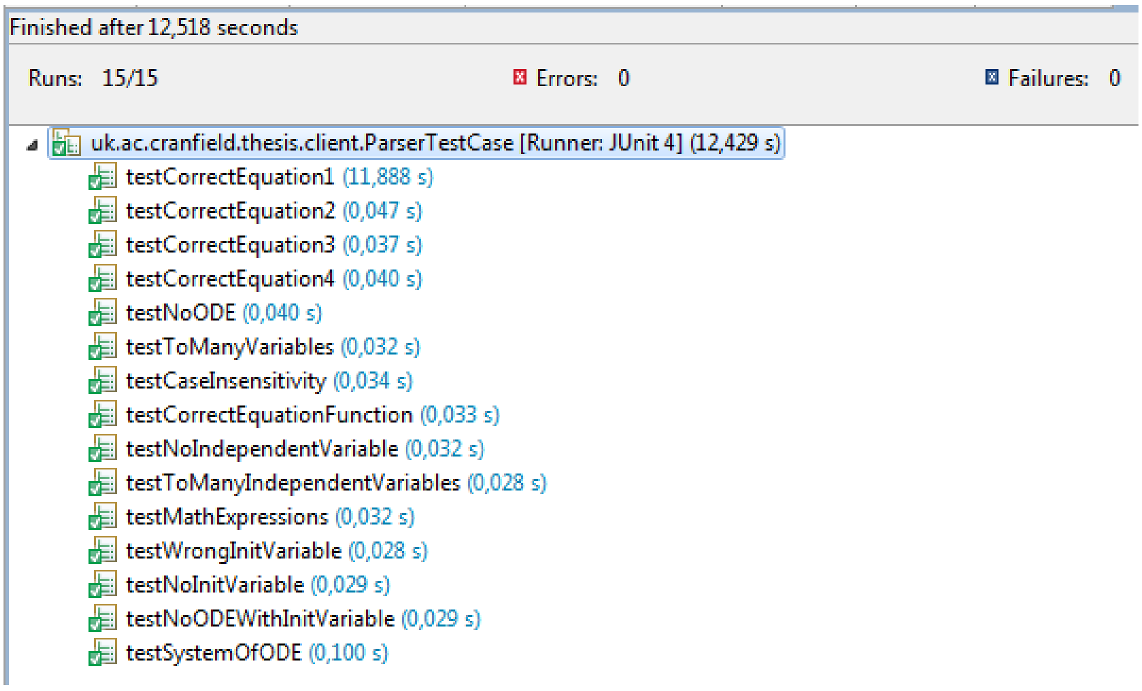
Name: system2

Figure 5-6 Developed dialog for saving systems of equations

6 Results

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

6.1 Parser Test Case results



```
Finished after 12,518 seconds
Runs: 15/15          Errors: 0          Failures: 0

uk.ac.cranfield.thesis.client.ParserTestCase [Runner: JUnit 4] (12,429 s)
    testCorrectEquation1 (11,888 s)
    testCorrectEquation2 (0,047 s)
    testCorrectEquation3 (0,037 s)
    testCorrectEquation4 (0,040 s)
    testNoODE (0,040 s)
    testToManyVariables (0,032 s)
    testCaseInsensitivity (0,034 s)
    testCorrectEquationFunction (0,033 s)
    testNoIndependentVariable (0,032 s)
    testToManyIndependentVariables (0,028 s)
    testMathExpressions (0,032 s)
    testWrongInitVariable (0,028 s)
    testNoInitVariable (0,029 s)
    testNoODEWithInitVariable (0,029 s)
    testSystemOfODE (0,100 s)
```

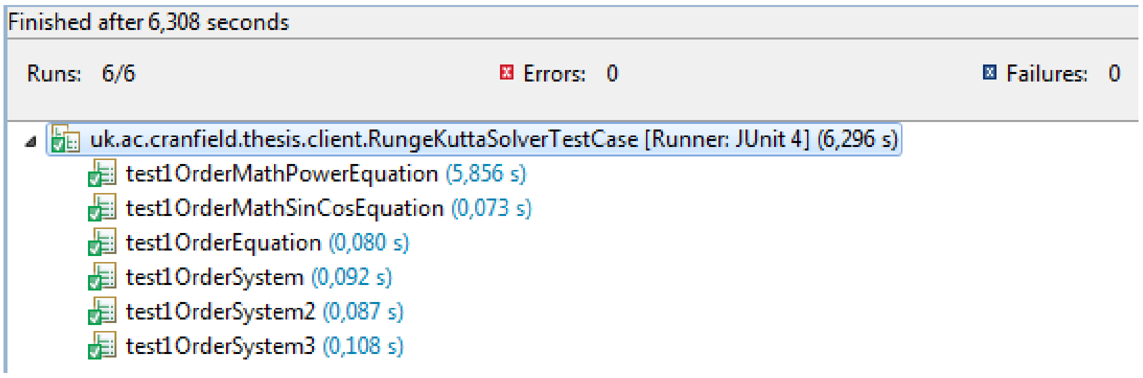
Figure 6-1 ParserTestCase results

All unit tests prepared for $Equation\ Parser$ module passed successfully after many corrections in the software. The vast majority of the tests were prepared according to the TDD approach, so before implementation. However, a few tests were written at the end of the process to cover scenarios where bugs occurred during black-box testing.

6.2 Solver Test Cases

The results of the tests performed to validate $ODE\ Solver$ module's functionality along with verification are presented in this chapter. Furthermore, this section presents the statement of all ODE Solvers implemented, along with accuracy levels and tests passed.

6.2.1 RungeKuttaSolverTestCase results



```
Finished after 6,308 seconds

Runs:   6/6          Errors:   0          Failures:   0

uk.ac.cranfield.thesis.client.RungeKuttaSolverTestCase [Runner: JUnit 4] (6,296 s)
    test1OrderMathPowerEquation (5,856 s)
    test1OrderMathSinCosEquation (0,073 s)
    test1OrderEquation (0,080 s)
    test1OrderSystem (0,092 s)
    test1OrderSystem2 (0,087 s)
    test1OrderSystem3 (0,108 s)
```

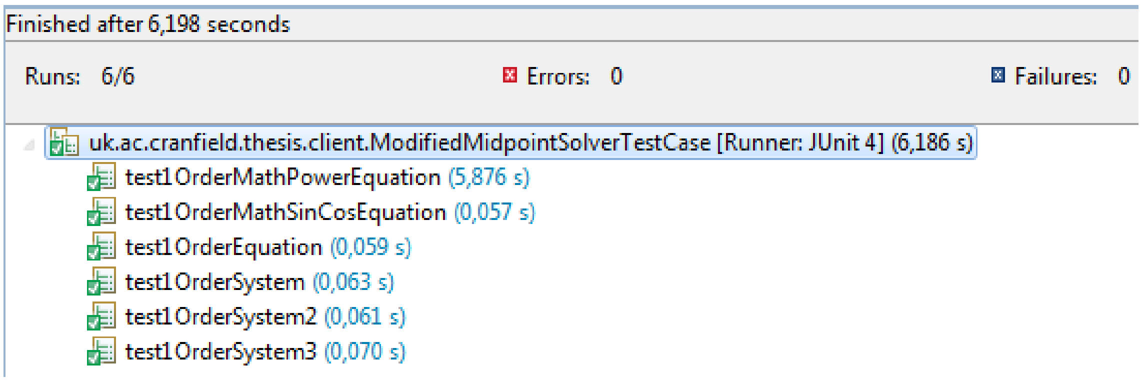
Figure 6-2 Runge Kutta TestCase results

The $RungeKuttaSolverTestCase$ extends $SolverTestCase$. The key point of this class is to perform tests for 4th order Runge-Kutta method. The accuracy is set to 10^{-4} . At this level all kinds of tests pass. Moreover, increasing the level of accuracy to 10^{-5} causes the failure of only one test ($test1OrderEquation$) which tests solving linear ODEs. After the accuracy is increased up to 10^{-6} all tests prepared for non-linear ODEs still pass, what is more some of them are still "green" with level of accuracy equals to 10^{-7}.

Table 6-1 RungeKutta tests results regarding accuracy

test \ accuracy	10^{-4}	10^{-5}	10^{-6}
test1OrderMathPowerEquation	√	√	√
test1OrderMathSinCosEquation	√	√	√
test1OrderEquation	√	X	X
test1OrderSystem	√	√	X
test1OrderSystem2	√	√	X
test1OrderSystem3	√	√	X

6.2.2 ModifiedMidpointSolverTestCase results



```
Finished after 6,198 seconds
Runs: 6/6          Errors: 0          Failures: 0

uk.ac.cranfield.thesis.client.ModifiedMidpointSolverTestCase [Runner: JUnit 4] (6,186 s)
    test1OrderMathPowerEquation (5,876 s)
    test1OrderMathSinCosEquation (0,057 s)
    test1OrderEquation (0,059 s)
    test1OrderSystem (0,063 s)
    test1OrderSystem2 (0,061 s)
    test1OrderSystem3 (0,070 s)
```

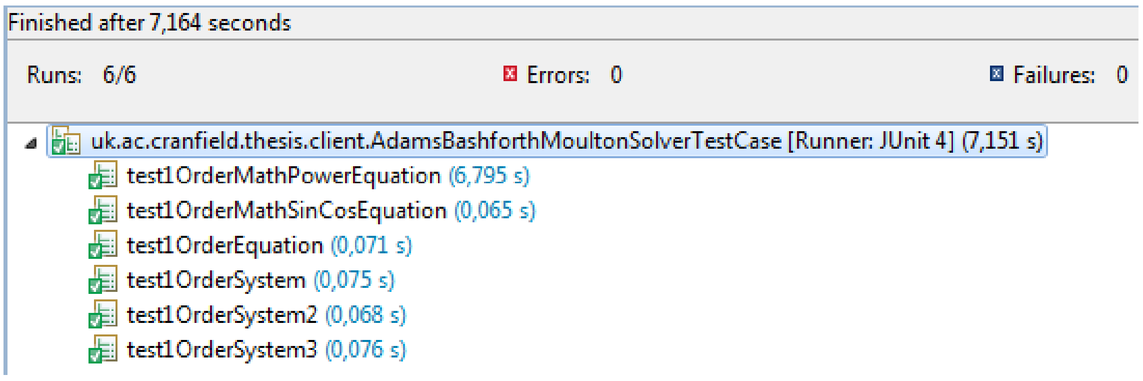
Figure 6-3 Modified Midpoint TestCase results

Another TestCase validates and verifies the Modified Midpoint method implemented. It is the 2nd order method as mentioned before, so the accuracy is only 10^{-2}, however the big advantage of this method is performance. The change of accuracy to 10^{-3} results in only two tests passed : $test1OrderMathPowerEquation$ and $test1OrderEquation$. These tests are prepared to verify linear and non-linear ODE's.

Table 6-2 Modified Midpoint tests results regarding accuracy

test \ accuracy	10^{-2}	10^{-3}	10^{-4}
test1OrderMathPowerEquation	√	√	X
test1OrderMathSinCosEquation	√	X	X
test1OrderEquation	√	√	√
test1OrderSystem	√	X	X
test1OrderSystem2	√	X	X
test1OrderSystem3	√	X	X

6.2.3 AdamsBashforthMoultonTestCase results



```
Finished after 7,164 seconds

Runs:   6/6            Errors:   0            Failures:   0

uk.ac.cranfield.thesis.client.AdamsBashforthMoultonSolverTestCase [Runner: JUnit 4] (7,151 s)
    test1OrderMathPowerEquation (6,795 s)
    test1OrderMathSinCosEquation (0,065 s)
    test1OrderEquation (0,071 s)
    test1OrderSystem (0,075 s)
    test1OrderSystem2 (0,068 s)
    test1OrderSystem3 (0,076 s)
```

Figure 6-4 Adams Bashforth Moulton TestCase results

The accuracy applied for this TestCase is 10^{-3}. At this level all tests passed . Moreover, by increasing the level to 10^{-4} only $test1OrderMathPowerEquation$ failed.

Table 6-3 Adams Bashforth Moulton tests results regarding accuracy

test \ accuracy	10^{-3}	10^{-4}	10^{-5}
test1OrderMathPowerEquation	√	X	X
test1OrderMathSinCosEquation	√	√	X
test1OrderEquation	√	√	X
test1OrderSystem	√	√	X
test1OrderSystem2	√	√	X
test1OrderSystem3	√	√	X

7 Discussion and conclusion

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

7.1 Analysis of numerical methods

Facing all reasonable practical numerical routines with requirements for this project, choices about methods to implement were made. The first choice is $4th\ order\ Runge-Kutta$ method. It is a very efficient and accurate method in most cases in terms of ODEs. Furthermore, it is step driven numerical routine, which fits project requirements.

Another implemented method is $Modified\ midpoint$. It is a very efficient 2nd order method with improved accuracy by intermediate approximations applied. Despite this improvement, this method has some troubles with non-linear equations. Overall accuracy is worse compared with $4th\ order\ Runge-Kutta$ method, however the advantage is better performance.

The last implemented method is $Adams-Bashforth-Moulton$, one of Predictor-Corrector routines. This numerical method is step dependent, which satisfies requirements. It is very efficient method, since only $predictor$ and $corrector$ steps take part in iteration of this routine. This method is still widely used in highly rated available software for solving ODEs. In terms of accuracy, results are especially good for smooth equations.

The main reason why $Bulirsch-Stoer$ method was not implemented is performance. This routine based on Richardson extrapolation is very efficient and accurate in terms of devising solution at some specific point. However, it is not possible to use the previous evaluation to perform another step. It means that in a step driven ODE Solver the only way to apply this method is to perform the whole routine for each step separately. Such a solution requires too many evaluations, especially for tasks with significant number of steps.

$Rosenbrock$ methods were found to be computationally complex offering only moderate accuracies. In general this type of method is less efficient than competitors because of plenty of matrix multiplications involved in ODE solving process.

7.2 Analysis of application's architecture

Suggested architecture for scalable GWT applications is Model View Presenter. This design pattern decouples application into following components :

- Model
- View
- Presenter
- AppController

The presented architecture is very similar to MVC. The Presenter contains all the application's logic and AppController handles logic that is not specific to any presenter and instead resides at the application layer. The key element in this architecture is $EventBus$ which is responsible for passing and registering events in the application.

The described architecture is a very good choice for large scale applications receiving plenty of events. However it is not necessary for GWT applications which uses a limited number of events to always follow MVP pattern. In exchange good separation of responsibilities can be provided using MVC pattern. When the number of possible event types is relatively small, there is no danger of it turning into a messy code having good separation of concerns. So usage of $EventBus$ is not always worth it in terms of GWT applications development.

The decision was made to develop this thesis project according to MVC since this specific application will not operate on big amounts of events. On the other hand change of architecture from MVC into MVP is not impossible for GWT applications. After extending application in terms of new features, what will probably increase number of events, MVP pattern is worth considering.

7.3 Results analysis

The level of correctness of the implemented ODE solvers can be estimated by the level of the accuracy set for Test Cases. Tables 6.1, 6.2 and 6.3 present results of ODE Solvers in terms of accuracy.

It is clearly visible that 4th Order Runge-Kutta method provides the best accuracy among all implemented methods. All tests passed with accuracy level set to 10^{-4}. Moreover, only one test failed after accuracy increased to 10^{-5}. The last column at the Table 6.1 shows that even at level of 10^{-6} this method provides correct results for non-linear ODEs.

The second implemented numerical routine is the Modified Midpoint method. The 2nd order of this method reflects lower level of accuracy. However this method is the most efficient among all implemented ones. All tests dedicated to this method passed with accuracy set to 10^{-2}. The level of 10^{-3} results positive only with linear and non-linear ODE tests. Finally , increasing accuracy to 10^{-4} ended up with only linear ODE test passed.

The Adams-Bashforth-Moulton method provides compromise of efficiency and accuracy among implemented methods. All tests run for this method passed with accuracy set to 10^{-3} . After increasing the level to 10^{-4} only one test for non-linear ODE failed, what shows very good accuracy results comparing to 4th order Runge-Kutta method. However the final increase of accuracy to 10^{-5} resulted in failure of all tests. It shows that this method is accurate to some limit.

7.4 Problems faced

Apart from minor problems related to design, implementation and testing process there was a situation which took more time to solve than other issues. The problem was with the Java version incompatibility between App Engine and the developed application. Initially the project was developed using Java 1.7 version. At the moment of writing this thesis, App Engine (SDK version 1.6.4) supports Java 1.6 runtime environment. Running the application using some Java 1.7 features on GAE caused a lot of bizarre errors and exceptions. After significant research of this topic, the issue was solved. The only possible way was to develop thesis project using JDK 1.6 . However it took some time to identify the source of the problem.

7.5 Conclusion

According to the aims and objectives included in the chapter 1.4, a summary of work carried out is stated below :

I. The Equation Parser module implements equations and systems parsing functionality.

II. The ODE Solver module implements three selected numerical routines solving series of 1st order simultaneous ordinary differential equations (linear or non-linear) for initial value problem. The user can provide domain's range and step through easy to use GUI.

III. Graph Viewer module implements visualisation of the solution as a 2D graph.

IV. Equation store module is responsible for storing and loading equations along with parameters provided by the user.

All aims and objectives are successfully fulfilled for this thesis project. The set of Test Cases was created to improve the quality of the software and verify if achieved the required level.

The author believes that developed application can be useful to achieve a solution for ODE or system of ODEs quickly from any device connected to the internet with web browser installed.

The developed application is currently running on Google's App Engine and available under following address : http://sodesolver.appspot.com/

7.6 Future work

The most important improvement for the future is probably adapting ODE Solvers to N-th order systems and equations solving. The application was developed in such a manner, that it is easy to extend it with new features. Moreover, the ODE Solver module was designed to operate on vectors instead of values with a view to N-th order implementation.

Another improvement for this application could be an authentication mechanism. The idea is to create separated accounts for users operating on the Equation store. Every user could store, load or remove ODEs or systems of ODEs which concern his account. To achieve this goal Google’s authentication mechanism could be applied, which is integrated with AppEngine.

REFERENCES

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

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

[3] Barnes, Stuart, (2012), “Advanced Software Engineering : Validation and Verification”, course notes, Cranfield University.

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

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

[6] Google Developers, (2012), “Command Pattern”, https://developers.google.com/java-dev-tools/codepro/doc/features/patterns/command_pattern (accessed 5 August 2012).

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

[8] Google Developers, (2012), “Datastore Overview”, https://developers.google.com/appengine/docs/java/datastore/overview (accessed 4 August 2012).

[9] Google Developers, (2012), “Entities, Properties, and Keys”, https://developers.google.com/appengine/docs/java/datastore/entities (accessed 4 August 2012).

[10] Google Developers, (2012), “Google Web Toolkit Overview”, https://developers.google.com/web-toolkit/overview (accessed 3 August 2012).

[11] Google Developers , (2012), “Large scale application development and MVP”, https://developers.google.com/web-toolkit/articles/mvp-architecture (accessed 13 August 2012).

[12] Google Developers, (2012), “Local Unit Testing for Java”, https://developers.google.com/appengine/docs/java/tools/localunittesting#Establishing_The_Execution_Environment (accessed 13 August 2012).

[13] Google Developers, (2012), The Java Servlet Environment : The Sandbox”, https://developers.google.com/appengine/docs/java/runtime#The_Sandbox (accessed 3 August 2012).

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

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

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

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

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

[19] 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.

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

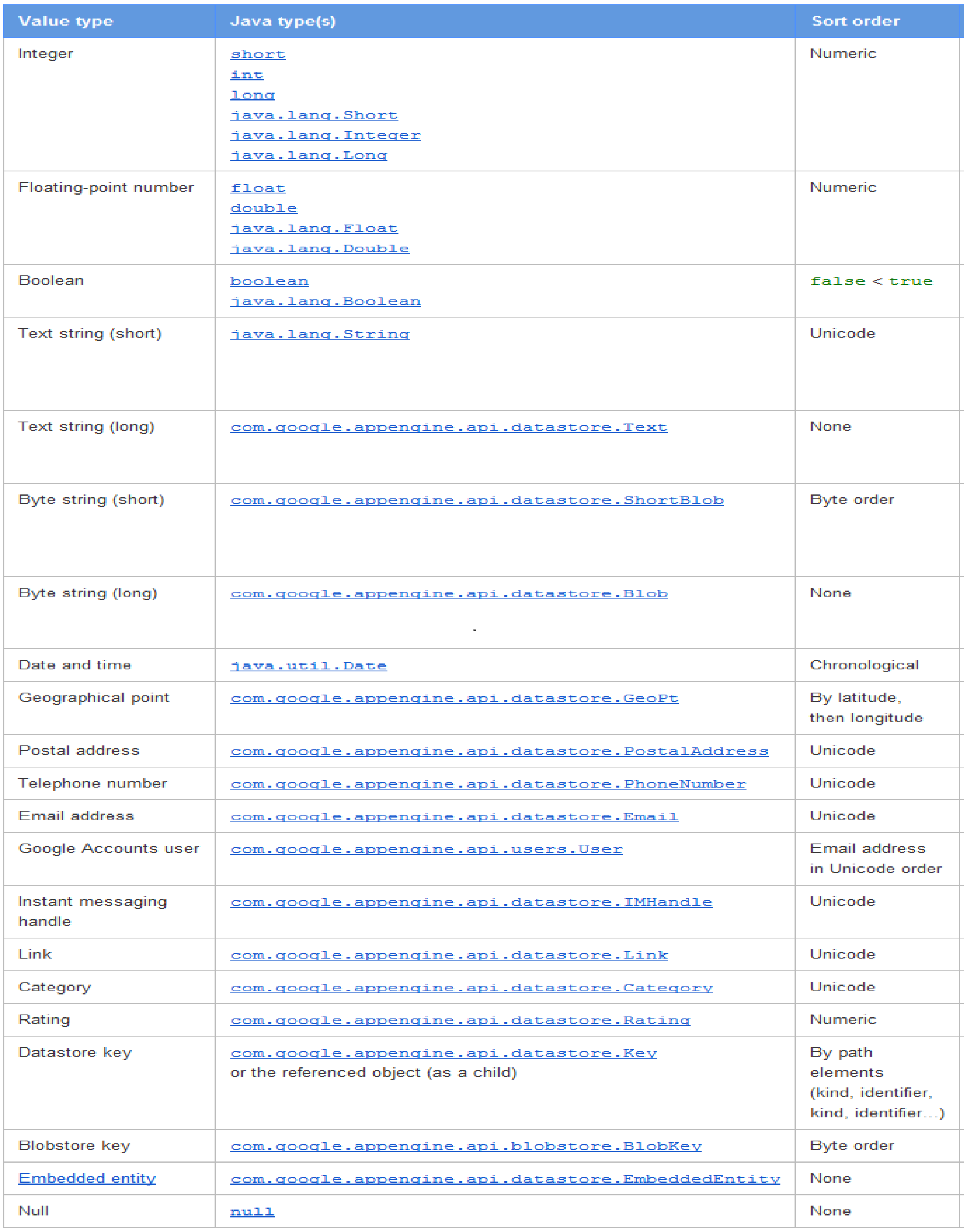
[21] Source Making : teaching IT professionals, “Command Design Pattern”, http://sourcemaking.com/design_patterns/command (accessed 6 August 2012)

[22] Source Making : teaching IT professionals, “Composite Design Pattern”, http://sourcemaking.com/design_patterns/composite (accessed 6 August 2012)

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

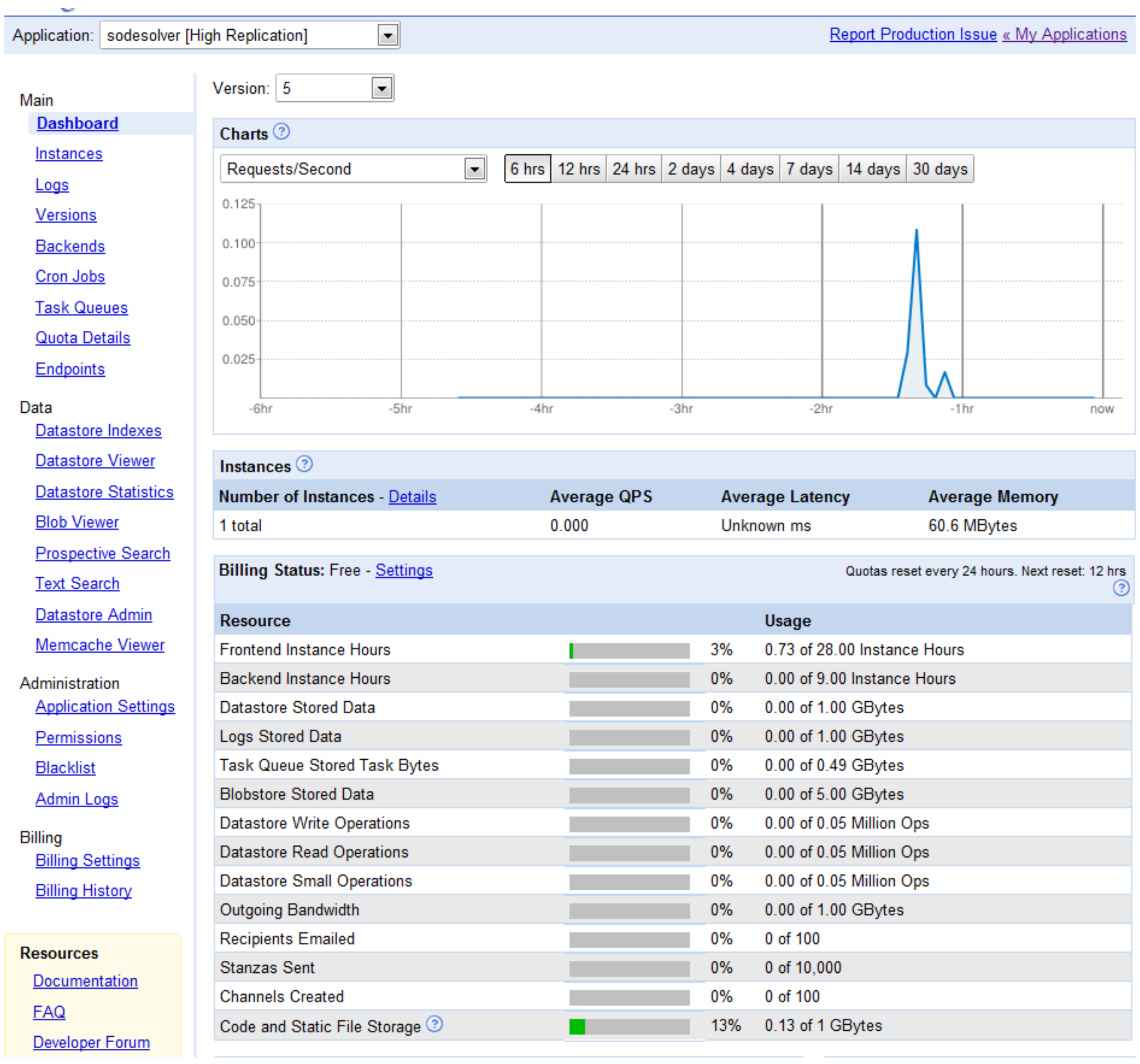
APPENDICES

Appendix A Datastore data types



Value type	Java type(s)	Sort order
Integer	short int long java.lang.Short java.lang.Integer java.lang.Long	Numeric
Floating-point number	float double java.lang.Float java.lang.Double	Numeric
Boolean	boolean java.lang.Boolean	false < true
Text string (short)	java.lang.String	Unicode
Text string (long)	com.google.appengine.api.datastore.Text	None
Byte string (short)	com.google.appengine.api.datastore.ShortBlob	Byte order
Byte string (long)	com.google.appengine.api.datastore.Blob	None
Date and time	java.util.Date	Chronological
Geographical point	com.google.appengine.api.datastore.GeoPt	By latitude, then longitude
Postal address	com.google.appengine.api.datastore.PostalAddress	Unicode
Telephone number	com.google.appengine.api.datastore.PhoneNumber	Unicode
Email address	com.google.appengine.api.datastore.Email	Unicode
Google Accounts user	com.google.appengine.api.users.User	Email address in Unicode order
Instant messaging handle	com.google.appengine.api.datastore.IMHandle	Unicode
Link	com.google.appengine.api.datastore.Link	Unicode
Category	com.google.appengine.api.datastore.Category	Unicode
Rating	com.google.appengine.api.datastore.Rating	Numeric
Datastore key	com.google.appengine.api.datastore.Key or the referenced object (as a child)	By path elements (kind, identifier, kind, identifier...)
Blobstore key	com.google.appengine.api.blobstore.BlobKey	Byte order
Embedded entity	com.google.appengine.api.datastore.EmbeddedEntity	None
Null	null	None

Appendix B Google App Engine's Administration console



Application: sodesolver [High Replication]

Report Production Issue « My Applications

Version: 5

Main
- Dashboard
- Instances
- Logs
- Versions
- Backends
- Cron Jobs
- Task Queues
- Quota Details
- Endpoints

Data
- Datastore Indexes
- Datastore Viewer
- Datastore Statistics
- Blob Viewer
- Prospective Search
- Text Search
- Datastore Admin
- Memcache Viewer

Administration
- Application Settings
- Permissions
- Blacklist
- Admin Logs

Billing
- Billing Settings
- Billing History

Resources
- Documentation
- FAQ
- Developer Forum

Charts

Instances

Number of Instances - Details	Average QPS	Average Latency	Average Memory
1 total	0.000	Unknown ms	60.6 MBytes

Billing Status: Free - Settings

Quotas reset every 24 hours. Next reset: 12 hrs

Resource		Usage
Frontend Instance Hours	3%	0.73 of 28.00 Instance Hours
Backend Instance Hours	0%	0.00 of 9.00 Instance Hours
Datastore Stored Data	0%	0.00 of 1.00 GBytes
Logs Stored Data	0%	0.00 of 1.00 GBytes
Task Queue Stored Task Bytes	0%	0.00 of 0.49 GBytes
Blobstore Stored Data	0%	0.00 of 5.00 GBytes
Datastore Write Operations	0%	0.00 of 0.05 Million Ops
Datastore Read Operations	0%	0.00 of 0.05 Million Ops
Datastore Small Operations	0%	0.00 of 0.05 Million Ops
Outgoing Bandwidth	0%	0.00 of 1.00 GBytes
Recipients Emailed	0%	0 of 100
Stanzas Sent	0%	0 of 10,000
Channels Created	0%	0 of 100
Code and Static File Storage	13%	0.13 of 1 GBytes