SOLVING ORDINARY DIFFERENTIAL EQUATIONS
APPLICATION AND PERFORMANCE

Mazen El-Masri

A Major Report

in the

Department of Computer Science

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

September 2002

0-612-77987-4

ABSTRACT

Solving Ordinary Differential Equations

Application and Performance

Mazen El-Masri

The purpose of this research project is to study performance of mathematical operations in software applications. Solving ordinary differential equations (ODE) is the main target to accomplish. Previous work in this field was done but there have not been intensive efforts to optimize performance.

The research aims on designing an application that can solve general differential equations using certain algorithms (ODE solvers) and developing the system in different languages in order to find the best solution that maximizes the application's performance. The design is an object-oriented design (OOD) due to the possibility of future enhancements to the system. The application is developed in two different ways. One version of the application uses Java as the programming language. This version is fully implemented and tested. The other version, which can be part of the future work, uses Java (front-end), Java native interface (JNI), and C++ (back-end).

Engineering Problem solver (EPS) is able to solve any differential equation using one of three algorithms that are build in to the system and plot the resulted solution as diagrams. During the development of the application performance is given great importance and is tested thoroughly. Certain changes are discussed to enhance this feature.

The outcome of the research is a framework for mathematical applications, a useful system that solves ordinary differential equations, and a performance study of the application.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

.NET is a Microsoft set of software technologies designed to connect the world of information, people, systems, and devices (10)

Algorithm is a set of rules that specify the order and kind of arithmetic operations that are used on specified set of data. These arithmetic operations could include such things as rounding rules, a logical decision or a specific formula (10)

Coefficient Is a constant or a variable function of time or a mix of both that can precede theta or any member of its family in the differential equation.

Example: $\tan(t^2)\,\frac{d^2\theta}{dt^2} + (\log(\frac{\pi e}{t})\theta = 0$

Δt (Delta T) Is equal to $t_{n+1} - t_n$ where t is time and n is the instance number.

F(t) (function of t) T is time. A function can vary according to time.

Example: $F(t) = \sin(\log(\frac{4t^2}{e}))$

Iterative Process is the act of repeating an operation with successive combinations of parameters, or the operation itself; automatic sequential operation (10)

JDK or Java Development Kit is a software development kit from JavaSoft, for producing Java programs (10)

J2SE or The Java 2 SDK is a development environment for building applications, applets, and components using the Java programming language. The Java 2 SDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform (15)

JNI or Java Native Interface is the bridge between a Java virtual machine (JVM) and natively compiled code. Native code can load a JVM and get access to Java objects through it, or Java classes may have native methods that are loaded by the JVM (8)

ODE or Ordinary Differential equation is Differential equations that involve only ONE independent variable are called ordinary differential equations (10)

OOD or Object Oriented Design is a software design method that models the characteristics of abstract or real objects using classes and objects (10)

Order Is the order of the equation or the number of the largest derivative of theta.

Example: $\sqrt{t}\,\frac{d^3\theta}{dt^3} + (\log(\frac{\pi e}{t})\theta + \sin(t) = 0$ is of order 3

θ˚ (Theta Prime) Is the first derivative of θ. Can also denoted by $\frac{d\theta}{dt}$

θ˚˚ (Theta Double Prime) Is the Second derivative of θ. Can also denoted by $\frac{d^2\theta}{dt^2}$

ACKNOWLEDGMENTS

To my mother who helped me throughout this research. Without your help and support, I would have never finished it. You never lost faith in me even when I lost faith in myself. Thank you Mom.

To my father who inspired me and trusted my ability. Your motivation that never stopped was the main reason I was able to complete this report. You made me believe that I can do it.

To my brother and sister for believing in me and assisting me when I needed it. I thank you a lot for that.

To professor Alejandro Allievi for his sincere and motivating supervision of my research project from the beginning. You have been a great asset and without your help my work would not have been as good as it is. You were professional in every way. You gave me the flexibility that I needed to make this work as pleasant as possible.

To professor Peter Grogono for acting as my formal supervisor.

To my girlfriend Genevieve who always understood me and stood by me. For your presence that was the help that I needed to keep on going; for your encouragement and support, for the coffee breaks and the chats that we shared in the middle of the nights throughout this research.

Finally to Sandi for her friendship and love. Your comments and inspiration affected my work and made it better.

Chapter 1

CHAPTER 1: INTRODUCTION

One of the main problems when building an engineering application is performance. Specific problems can be troublesome when trying to find an optimal solution especially when they deal with millions or even billions of calculations. These types of applications are very useful in vehicles, simulators, and various machineries. The major mission of the EPS system is to plot diagrams that establish results for general mathematical equations. The equation solvers (ODE Solvers) or algorithms take an equation as input and solve it using the built-in algorithms. There should be no limitations on how the equation is or how difficult it can be to plot its results.

This report is divided into seven chapters. In chapter two, we introduce the research and explain the workflow. Chapter three contains a detailed mathematical description of the problem. In this chapter we also identify the algorithms used to solve the problem. Chapter four describes the software process used for building the system. The chapter includes the analysis done on the requirements, the design, as well as implementation issues. Chapter five discusses all the design and coding modifications done for enhancing performance. Chapter six explains what resulted from the research. In chapter seven we introduce JNI or Java Native Interface and discuss future work to be done. Chapter eight is the conclusion of the research.

Chapter 2

CHAPTER 2: RESEARCH AND PROJECT DESCRIPTION

In this chapter we discuss the reasons behind this research, why is there a need for such an application, and what is the project goal. We also explain the research and development iterative process.

Initially, the aim of this project was to find a solution for ocean vessels to secure their safety against weather and ocean conditions. Small and medium size ships owners need a simple and cheap method to predict whether the ocean conditions are manageable and stable for them to control their vessels. These problems can be simulated using ordinary differential equation (ODE). The inputs that are needed for the ocean vessel situation are ship dimensions, wave dimensions, and additional ship motion information such as speed and time (14). The output is a set of diagrams showing the reaction of the ship towards these inputs. The main diagrams are the reactions of roll angle and roll velocity with time.

ODE solvers are not only used for ocean vessels. Since an algorithm can solve equations and an equation can describe the object's physical situation then we can target our application to solve any object simulation. By making a generic solution that accepts any ODE and can use various ODE solvers, we can produce a powerful application assisting various users in predicting future movements of their objects with time.

Therefore, our project's requirements has changed from helping small to medium ocean vehicles in answering their sailing problems to helping objects simulate their movement according to certain conditions with time.

The complexity of the project is not in finding a solution for the problem. It is to find the optimal solution that can show the needed results in an acceptable period of time. As mentioned before we might need to do a large number of calculations before we are able to supply the user with the proper answer. We need to keep in mind that the user might not have the luxury of time. The answer to our problem is not to develop a complex application with lots of unnecessary features but to develop a system that provides solutions with the minimum processing time and memory usage.

Some applications, such as real-time systems, have performance as its highest priority. In our situation, we want to establish an optimal solution that can be suitable to the user in all aspects.

Chapter 3

CHAPTER 3: MATHEMATICAL CONCEPTS

Most mathematicians and people involved in mathematics are familiar with equations, functions, and derivatives. For others who are not, there has to be some understanding of these aspects. In this section, I explain the project in detail. The reader, though, needs to know the basics of algebra, trigonometry, and calculus. The project is made up of:

1. Equations (to be solved)

2. Algorithms (equation solvers)

3. Initial inputs and parameters

Equations

An ordinary differential equation is the main input of the project. A general form of such an equation looks like the following:

- $$F^n(t)\frac{d^n\theta}{dt^n} + F^{n-1}(t)\,\frac{d^{n-1}\theta}{dt^{n-1}} + \ldots + F^2(t)\,\frac{d^2\theta}{dt^2} + F^1(t)\,\frac{d\theta}{dt} + F^0(t)\theta = F(t)$$

For a problem like a specific sailing ship, the equation is a fixed formula.

- $\frac{d^2\theta}{dt^2} + (\frac{W}{I_{xx}})(\mathrm{GM} - (\frac{\Delta GM}{2})\sin(2\omega_e t))\,\theta = 0$

Where:

- $F^2(t) = 1$ *[constant],*
- $F^1(t) = 0$ *[constant],*
- And $F^0(t) = (\frac{W}{I_{xx}})(\mathrm{GM} - (\frac{\Delta GM}{2})\sin(2\omega_e t))$ *[function of time]*

The above equation is a differential equation that describes the roll motion of a ship in waves. Using the EPS application, we are able to create any kind of differential equation allowing variable coefficients of function of time (t). Therefore, we can create an equation as the above one and solve the ship roll motion problem or any other (7).

A constant coefficient is a numeric value that is independent of time that precedes any member of the theta family (example: $2\theta'' - \theta = 0$), whereas a variable coefficient is a function that depends on the value of t. For example:

- $\tan(t^2)\ \theta'' + (\log(\frac{\pi e}{t})\theta = 0$

In the above example, the order of the equation is two. This is because of the existence of theta double prime (θ'') in the equation. The minimum order of any equation is two. The application can solve equations with the order of five at most. Example:

- $$F^5(t)\frac{d^5\theta}{dt^5} + F^4(t)\frac{d^4\theta}{dt^4} + F^3(t)\frac{d^3\theta}{dt^3} + F^2(t)\frac{d^2\theta}{dt^2} + F^1(t)\frac{d\theta}{dt} + F^0(t)\theta = F(t)$$

Any function attributed to any member of the theta family can include the following mathematical operations:

Trigonometric functions	Sine (sin), cosine (cos), tangent (tan), cotangent (cot)
Logarithmic functions	Log
Power	Pow
Other operations such as	+, -, *, /

Other constants can also be chosen. These constants include π (3.1416) and e. Brackets are included in the function builder because they are needed for writing proper functions. Functions can be simple. Example:

- sin(t)
- log(3t)

They can also be very complex like:

- $\log(\frac{\pi e^5}{\sin(4t)})$

- $\log(\sin(\cot(t)))$

The right hand side of the equation can also be a function of time. The equation can have a mixture of constant coefficients and variable coefficients. Coefficients can be zero except the coefficient of the highest order. The reason behind this is that if the coefficient of the highest order is zero then this highest order does not exist implying that the order of the equation that was initially set is incorrect.

Algorithms

Webster *New Collegiate Dictionary* defines an Algorithm as "a step-by-step procedure for solving a problem or accomplishing some end" (5). We use algorithms to integrate equations and solve them. In order to solve the equations explained in chapter three we use known algorithms. In the project, we have three algorithms implemented that are widely used to solve these types of problems and they are the following:

A- Explicit Euler scheme: $\theta_{n+1} = \theta_n + \Delta t\,\theta'_n$

Where:

- $\Delta t = t_{n+1} - t_n$. The time difference between instance n and instance n+1.

- θ (Theta) and all its corresponding derivatives like θ', θ''... (Theta prime and theta double prime) are the variables that are needed to be calculated with time. Since θ varies with time we need to calculate each instance of θ and its available derivatives with time change. The time change (Δt) is specified by the user. It means how often (or how quick) do we need to calculate θ and it's derivatives. To further explain, θ_n is the n^th instance of θ. The instance at time $n*\Delta t$. θ'_n is the n^th instance of θ' and so on and so forth.

- The initial values of θ and its derivatives are also constant inputs from the user

B- Crank-Nicolson scheme: $\theta_{n+1} = \theta_n + \Delta t\,(\frac{(3\theta'_n \;-\theta'_{n-1})}{2})$

C- Series of Predictor-Corrector schemes:

$$\theta^{(p)}{}_{n+1} = \theta_n + \Delta t\,[\sum_{k=1}^{k=n} \beta mk\; \theta'_{n-k+1}]$$

The user can choose one of these three Algorithms to solve an equation in order to help figure out all instances of the θ family with time. We need to provide the start time, the end time, the time step, and the initial values of the θ family. In order to plot a specific equation of theta in 2 minutes (120 seconds) with the time step of 10^{-6} seconds that means we have to calculate theta and family $\frac{120}{10^{-6}} = 120*10^{6} =$ 120 million instances. If the order of theta in the equation is 3, meaning the largest derivative is θ''', that means we have to calculate all instance of θ, θ', and θ''. The sum of all calculations would be 3 * 120 million = 360 million points. These 360 million points have to be plotted in order for the user to see how is his vessel or object reacting with time. More than 360 million points will be displayed because we will have to show how θ changes with time, how θ' changes with θ, how θ'' changes with θ, and so on.

The above example is just a small example that shows how important performance is for these types of applications. We will not go to the extreme since the machines that we will run the application on would be a regular PC and not a multiprocessor. During the coding-testing iteration process of the application we realized that plotting a large number of points slows the application considerably. Therefore, a study was done to determine the enough number of points that is needed per timeslot in order to plot a correct result. We tested the application with various timeslots and timeframes and we came up with a valid figure (20 points/timeslot).

Initial Conditions

In order to solve a specific equation some initial conditions need to be inputted. These inputs are:

1. The timeframe in which the points to be plotted as output needs to be calculated in,
2. The time-step for every single calculation,
3. The initial values of the theta family members,
4. And the algorithm selected for solving the inputted equation.

When inputting the equation we need to specify the order and the coefficients of every theta family member in the equation as well as the RHS (right hand side) of the equation (usually equals to zero).

Chapter 4

CHAPTER 4: SOFTWARE PROCESS

In this chapter we describe the software process that we followed in order to complete the project. It will describe the requirement analysis, the design, and the implementation.

Requirement Analysis Approach

The Analysis done on the initial requirements of the system is described in this section. The process we took in order to understand the requirements and the decisions we made in regards to design a generic application is also explained.

Complex mathematical applications are not common in the computer science field. That is why the best way to start analyzing the problem is with a pencil and a piece of paper. After getting familiar with this type of work we started with simple equations and the simplest solving algorithm (Euler). We took an iterative operation to implement the application:

1. Adding a few requirements,

2. Designing,

3. Coding,

4. Testing,

5. Optimizing the application be finding the performance bottlenecks whether in the application's design or in the code,

6. Redesigning and recoding (if necessary),

7. Recoding (if necessary),

8. And testing again (if necessary).

9. After that we go back to step one and add new requirements until all the requirements are fulfilled.

After reaching an optimal (or close to optimal) design and coding the only thing left would be the type of coding language we are using. The following two sections discuss this point in details.

Performance Optimization Approach

Addressing performance is done while analyzing the requirements. It is also studied after each development iteration (see chapter five for more details). The approach that is taken to improve performance was:

1- Measure the program's performance under realistic conditions. This should be done extensively since it is the major requirement.

   i. We time every method that is called when the main function is executed.

   ii. We verify that the time taken by the methods called is short.

If it meets the requirements then we are done. If not go to next step.

2- Find the most critical performance bottlenecks in the specific methods called and/or in the design and try to optimize them. For example if a specific loop in a method is taking too long to finish we try to optimize the code in it. This can be done by verifying that we are not using more resources and processing time than we are supposed to. To illustrate, using a variable of 'long' datatype when only a short datatype is needed might slow the process down considerably.

3- Find out what else could be changed to enhance performance. This can be a design issue or a coding issue. If a major design problem was found then redesigning the application is done.

4- Always time the code before and after making the changes to verify that your changes has improved the program performance.

Note that simple operation like incrementing a <long> variable when only an <int> variable is needed can greatly affect performance. Incrementing a long takes 3.5 times the time needed to increment an <int>. This is because a <long> datatype is 10 bytes long, whereas an <int> is only 2 bytes long. Let us take a quick example to explain how a small change like this can considerably enhance performance:

```
long lTimeStep;

For (long lCount = = 0; lCount < ( (lMaxT – lMinT) / iDeltaT ); lCount++)

{

        lTimeStep++;

        If (lTimeStep > 10)

                lTimeStep = lTimeStep - 10;

        depVar[0][lCount]=depVar[0][lCount]+depVar[0][ lCount +1]* lTimeStep;

}
```

The above for loop takes approximately 30 μs to complete one iteration on a Pentium II 266 MHz with 128 megabytes of RAM. The three lines of code in the loop take 7, 10, and 13 μs respectively. In this case, if there are one million iterations in this loop it will need 30 seconds to complete. If we look carefully at the code we can notice that lTimeStep can never be greater than 10. There is no need to allocate a long for a variable that cannot grow greater than 10. If we define lTimeStep to be a short instead of a long we will allocate less memory and less CPU time to work with that variable. The time needed for this loop drops considerably. When testing a method that is very similar to the above piece of code we found out that optimizing variable allocations can sometimes cut the processing time in half and sometimes less. In this specific loop the new time needed to complete one million iteration drops to less than 11 seconds. That is more than 60% improvement. This example is one of the many studies done to enhance performance.

Object oriented technology has proven to be the leading technology in software development. Recent studies by Microsoft have strengthened this idea by using the object-oriented methodology especially with their latest .NET framework where the operating system platform framework classes can be used in development. Microsoft also changed VB (which is now known as VB.NET) to use object orientation as the infrastructure of the programming application. Since our application is a starting point for more complex engineering application we wanted it to be generic enough to be a base framework for future enhancements. Object oriented design and programming is the best solution for us. This is due to various benefits it provides (12):

1. Use-case driven design: This approach is essential for capturing the requirements of the system. Use cases and their scenarios speak the language of the end-users thus providing a way for them to state their expectations about the desired behavior of the system to its developers.

2. Reusability: When using Object oriented programming approach we reduce the coding involved by using inheritance and polymorphism in our design.

3. Easier to understand: When we collect the requirements and design our application we get a better understanding of the application. With diagrams like the use-case diagrams, the class diagrams, and the sequence diagrams we can view the application easier, especially when we are developing complex applications or incremental systems.

4. Separation of concerns: separating objects into class categories and encapsulating layers of abstraction provide an ideal basis for dividing up the staff into project teams.

As for picking up the perfect suitable language we decided that either VC++ or Java would be the best approach. This is because these two languages are highly supported and used by big organizations. Since we are building a framework of such a mathematical application we wanted it to be used by other researchers. Furthermore, these two programming languages supply the developers with large libraries and functionalities that make the development process easier and faster.

We mentioned before that reaching a good design and programming EPS (Engineering Problem solver) efficiently is possible but finding which is the best programming language for programming the engine for the application can be harder to reach (VC++ or Java). Nanoseconds per point calculation can make a difference so choosing which is best can be impossible unless we have two versions of the application and we can compare the performance. A study to determine the best appropriate programming language is not in this report but can be part of the future work to enhance the research.

That being said we will have two versions of EPS:

1. One is developed as a pure Java application (GUI and engine). This version is fully implemented and is used to test the design and the performance of the application.

2. And the other would be a mix of the two. The engine can done in C++, the front-end is in Java (same graphical user interface) with a Java-C++ interface in between. For such interface we can use a new technology known as JNI or Java Native Interface. A more detailed discussion about this version is in chapter seven.

Implementation Approach

This section describes the first phase of development. After fully understanding how the application run and what is required from it we start the iterative approach of designing, coding, testing, and evaluating performance using only Java as the programming language.

There are a few benefits which lead to Java being chosen over C++. Both languages are very similar; they both follow object orientation and have great syntactic similarity and a similar semantics. The main benefit in using Java as the designers of Java said is that they took C++ and removed the features that caused many common bugs, such as memory leaks. C++ is a very powerful language; yet has dangerous tools such as pointers and manual memory management one does not have to use in every case. Java provides garbage collection if we want or we can use references instead of pointers and we can use vectors instead of manually allocating memory for dynamic arrays (11). The point is that C++ gives you the choice to program at the appropriate level of abstraction.

Java forces us to always be safe at the real cost of performance (11). Another benefit is that managing code in Java is much easier than C++. Add to the fact that Java has one of the most robust type checking and error-handling systems that has ever been seen in a programming language. This is because the developers of Java implemented detailed error-handling libraries and functionalities.

Java is feasible for web development so if we were going to redesign the application to run as an applet or a web-enabled application Java would be a good choice. Java applets can be imbedded in HTML or ASP pages easily and affectively.

Object handling is automatic in Java; initialization of primitive class data members is guaranteed and defaulted to zero or equivalent. Destructors are not needed; this is another benefit in Java. The object's lifetime is determined by the garbage collector and not by the programmer. The built-in multithreading support is another benefit that is useful in coding. Java contains standard libraries for solving specific tasks like networking, multithreading, distributing objects, compression, and database connection. Thus, the Java language emphasizes accuracy and reliable application behavior at the expense of performance.

There are lots of benefits in Java that can be very helpful in the process of development. Slowness is the only drawback of this programming language. The question is: will these benefits help make the Java-version application run faster than the C++-version or will the great performance that C++ provide outcome these benefits. This will be answered after comparing the performance of the two applications.

Application Design

The context diagram below illustrates the main system and it's relationship when the external entity; the user. The user is any person who needs the application for simulation purposes. The communication between this entity and the system are described as inputs and outputs.

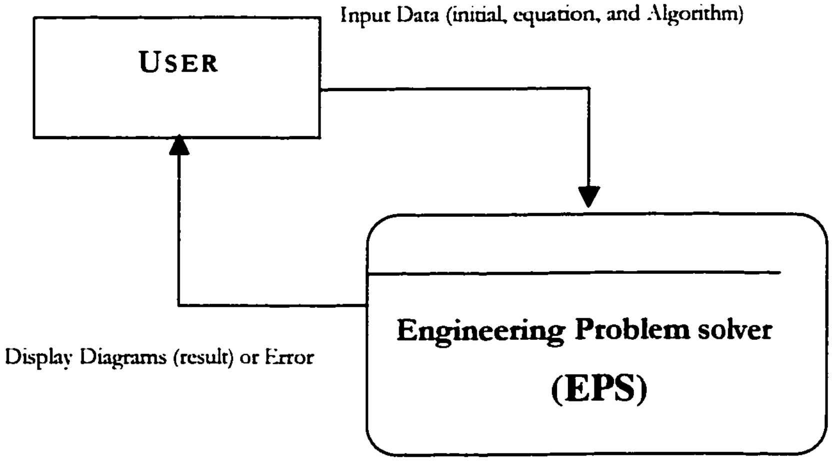

Input Data (initial, equation, and Algorithm)
USER
Display Diagrams (result) or Error
Engineering Problem solver
(EPS)

Figure 1: Context Diagram

The main functionality of the application is to calculate plotting points for the theta and family, from the equation, and to draw the corresponding diagrams. For this reason we only have one use case for the whole application.

The main use case has one method to use which is <CALCULATE>.

The scenario of the main use case is as follows:

Actor: User.

Stakeholders:

— User: Wants accurate and fast result.

— EPS system: Gives clear information to the user with the best possible performance.

Preconditions:

—User knows the input that application needs to calculate and plot the graph.

Post conditions:

—N/A.

Main Scenario:

1- User Enters the order of the equation (1 → 5)

2- User Enters the coefficients of every member of the theta family in the equation by carefully building the functions from the dropdown box or just specifying constants

3- User enters the right hand side of the equation (RHS)

4- User specifies the initial values of theta and it's family

5- User specifies the initial time, the stop time, and the time-step in which the diagrams are to be plotted

6- User requests the calculation to take place

7- System calculates the needed points and plots the diagrams

Below is the main use case diagram (M-UCD)

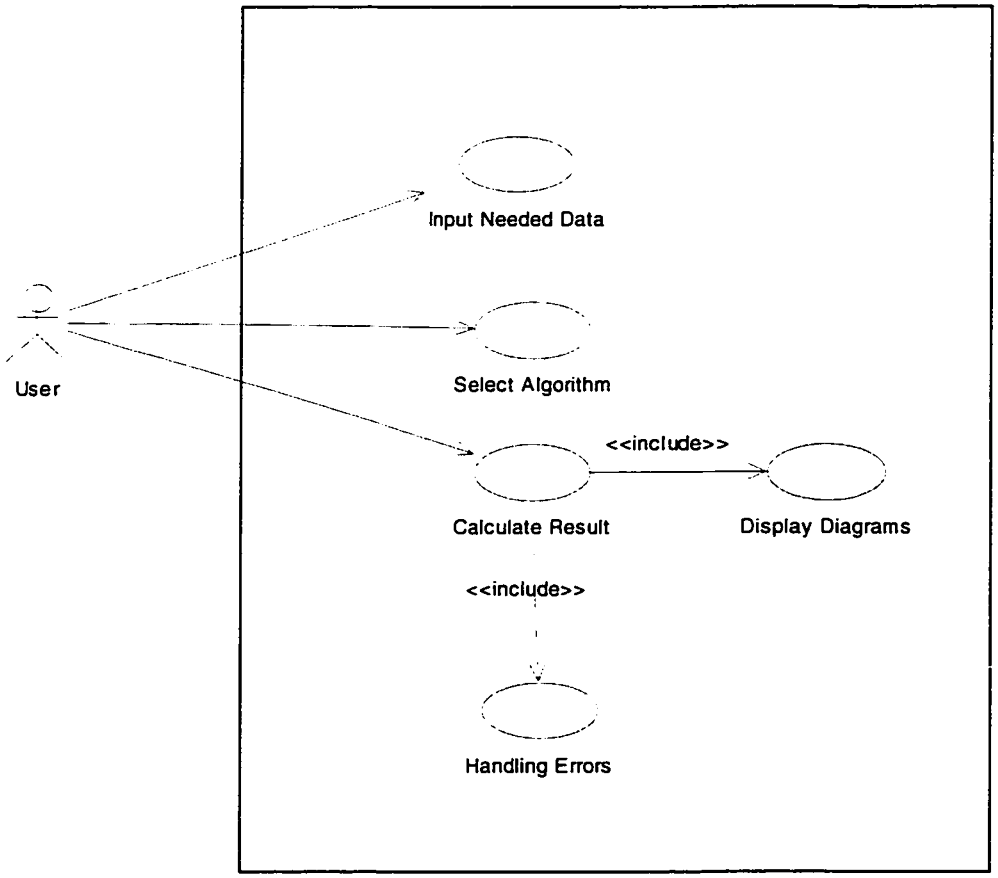


Figure 2: Main Use Case Diagram

We will now describe the use case diagram in details and will further divide the main use case to sub use-cases. There are four operations described in the main diagram above. Therefore we will have four use cases:

1. Input Data
2. Select Algorithm
3. Calculate Result
4. Display Diagrams

The 'Input Data' Use Case Diagram (UC-1):

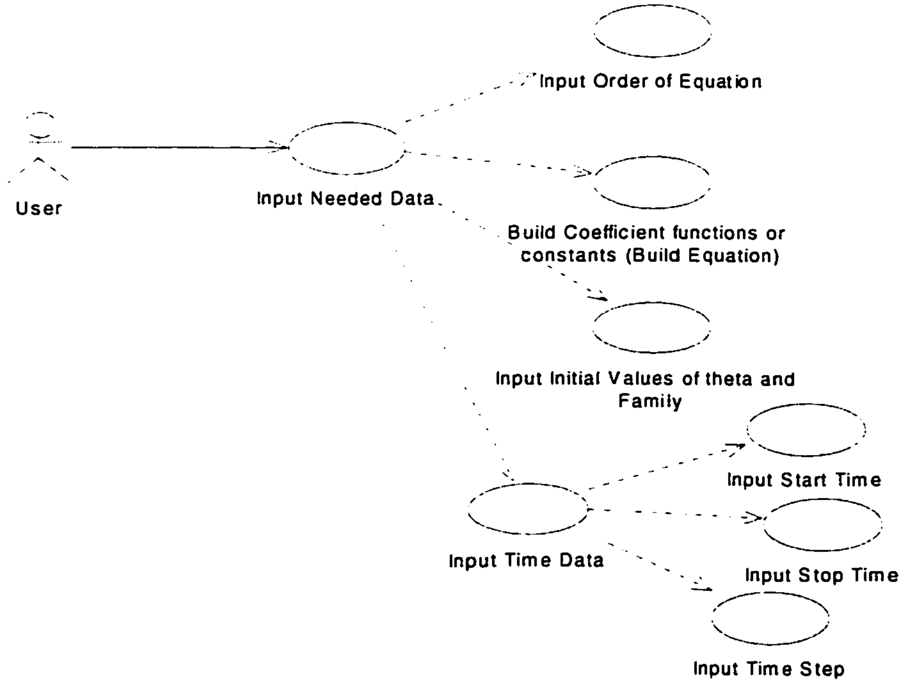
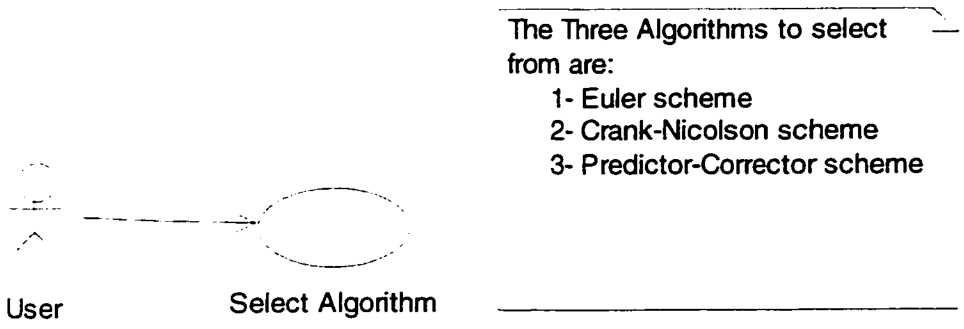


Figure 3: Input Data Use Case Diagram

The 'Select Algorithm' Use Case Diagram (UC-2):



The Three Algorithms to select from are:
1- Euler scheme
2- Crank-Nicolson scheme
3- Predictor-Corrector scheme

Figure 4: Select Algorithm Use Case Diagram

The 'Calculate Result' Use Case Diagram (UC-3):

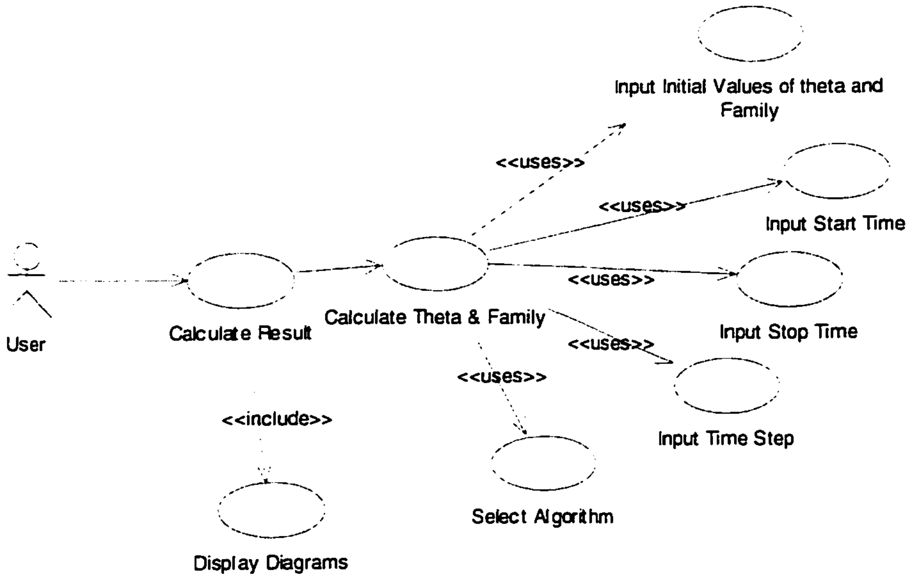


Figure 5: Calculate Result Use Case Diagram

The 'Display Diagrams' Use Case Diagram (UC-4):

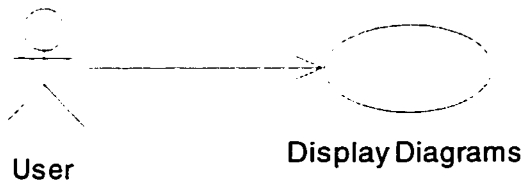


Figure 6: Display Diagrams Use Case Diagram

As mentioned before, the approach we are taking in order to research and develop such project is an iterative one. Every phase in this approach will be visited a few times until all the requirements are satisfied. We intentionally missed a few requirements the first time we started the design and implementation phases. This is because we wanted to simplify the initial work. The first requirement phase did not have the following requirements:

- Functions as coefficients in the equation. We only dealt with constant coefficients for simplicity purposes.
- We only had one algorithm (Euler)
- There was only one diagram displayed

- ➢ Error handling was kept to the minimum.
- ➢ Speed and performance was the main concern

What we did is we built the specification for the minimum amount of requirements we could handle and we designed the application. We will not go into details of that version of the design since it is not necessary at this point. We followed a software engineering methodology with the name: "Evolutionary development". It is similar to the waterfall model except that the requirement are added one step at a time keeping every step as simple as possible. When the whole development phase is finished we go back to adding new requirements → redesigning → implementing and testing.

The objective of evolutionary development is to evolve a final system from an initial outline specification. Start with well-understood requirements and start building on them. The problems with this model are:

1- Lack of process visibility

2- Systems are often poorly structured

These problems can be managed and resolved in small to medium-size interactive systems and with proper documentation. This is the case of our system; we took great caution in following this model. The benefits are much more than the drawbacks since one tend to understand the system along the way (if it's complex) as well as focusing on certain things like performance.

Here is the general evolutionary development model we are following in developing our system (6):

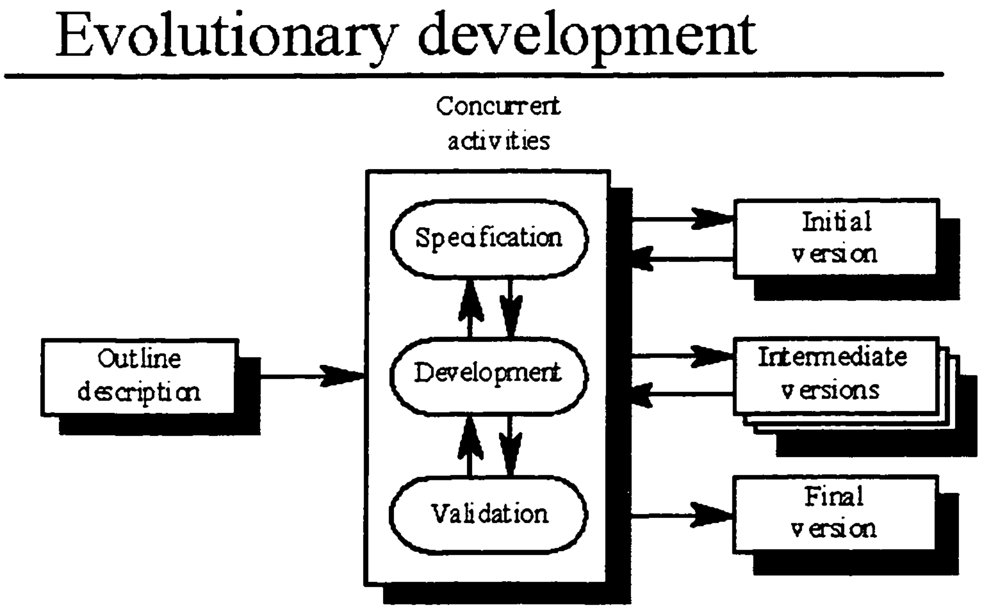


Figure 7: Evolutionary development model

We discussed what was missing in the first system build. After finishing and fully testing the design and code of the system we went back and added the functions as coefficients requirement. This, of course, forced us to modify the specification and the initial design of the system but made us understand why we are redesigning it. Emphasizing on performance, we were able to pinpoint the bottlenecks and fix them on the spot. The reason behind that is that the size of application is small and growing.

After Finishing build II we started build three by adding the other two algorithms. Build IV concerned error handling. In build V we focused on major bottlenecks in performance and we made few coding changes as well as changing the third party software we were using to display diagrams to a faster one. Build VI was another pass-thru the design and code to assure optimal performance. All the changes we made will be described in details in a later section since the design has not been discussed yet. We will not discuss or show the previous versions of design and code in this paper but we will explain the changes we've done in detail and the reasons behind them.

The revised use case descriptions in the EPS system are used as an input document. The Use-Case descriptions were reviewed for nouns and noun phrases to identify potential classes.

From the potential classes, we develop an ideal object model for each of the use cases. We begin by creating an interface class to handle the interaction between the use case and the actor (user) outside the system. Then, we create one or more control classes to manage the use case. Last, we create entity classes for each of the major things that the transaction involves.

Here is the general class descriptions evolving from the use-case diagrams:

Actions	Preliminary Classes	Class type	Original use-case
User selects equation order	ODE (Ordinary Differential equation)	Entity	UC-1 (Input Data)
User inputs The variable coefficients for the equation (building the equation)	Vcoefficient	Entity	UC-1 (Input Data)
User inputs the time information	VarCoefODESolver	Entity	UC-1 (Input Data)
User inputs the initial values of the theta family.	VarCoefODESolver	Entity	UC-1 (Input Data)
User selects the algorithm required for the calculation	VarCoefODESolver VarCoefODESolver1 VarCoefODESolver2 VarCoefODESolver3	Entity Entity Entity Entity	UC-2(Select Algorithm)
User requests the output (depending on the algorithm requested)	VarCoefODESolver1 VarCoefODESolver2 VarCoefODESolver3 Vcoefficient ODE	Controller Controller Controller Entity Entity	UC-3 (Input Data)

Table 1. Preliminary classes table

Other classes where added after the preliminary ones to insure proper error handling and for the graphical user interface. These classes are:

- MainGui
- Welcome
- DialogError

Reasons	**Preliminary Classes**	**Class type**	**Original Use-Case**
In case of any error occurring in the calculation phase because of invalid user input	DialogError	Controller	M-UCD
Graphical User Interface	MainGui	Interface	UC-3 (Display result) UC-1 (Input Data)
Displaying a welcome screen on startup	Welcome	Interface	N/A

Table 2. Additional classes table

From the Classes we defined in the last section we can now structure a class object model that shows a simulation of how the classes interact (1). Below is a diagram that defines the ideal object model.

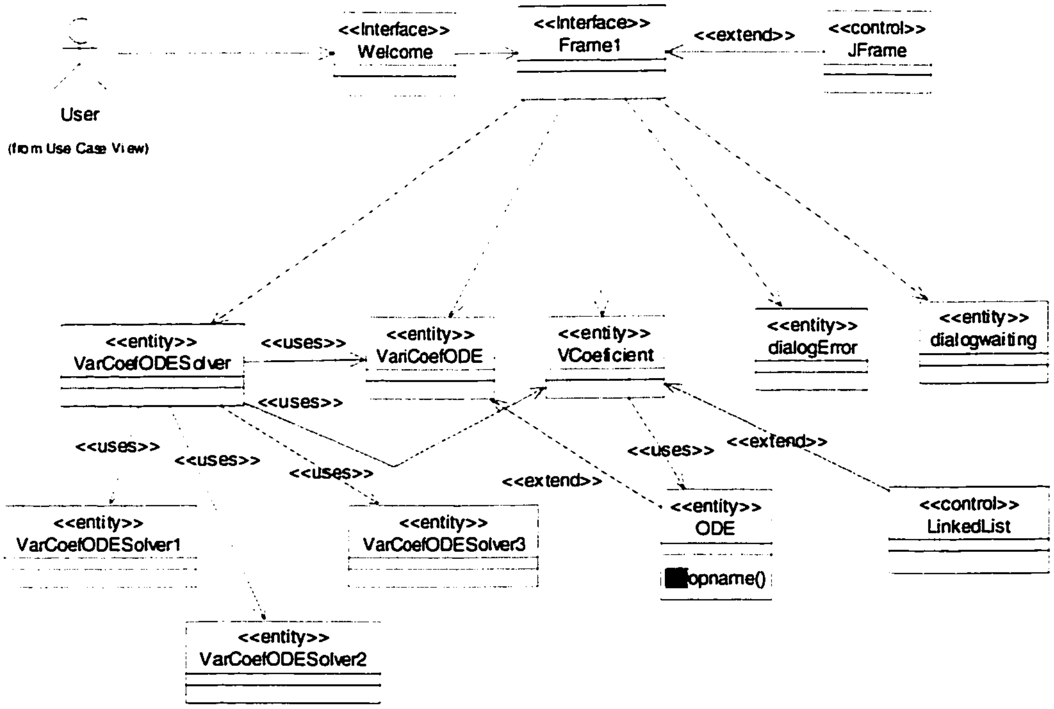


Figure 8: Ideal Object Model for main use case and sub use cases

All the above analysis was derived from the use cases previously designed. The potential objects and their behaviors were thoroughly studied and reviewed to come up with the following decisions (1):

- Create three classes for solving equations depending on the three specific algorithms. Class names were called (VarCoefODESolver1, VarCoefODESolver2, VarCoefODESolver3)

- Build a separate class for error handling (DialogError)

- Add a welcome interface class.

- Enforce weak coupling between classes (entities) for reusability purposes and future enhancements. Entities are passes as parameters from once class to another during execution.

We collected the entity classes together and found the required functionalities that they need. Some entity classes were redesigned; others were restructured for better use of inheritance in order to have a framework that can be reused (4). The User interface classes were added as well. We used the actor (EPS-User) in order to come up with the proper interface classes. Finally we were able to structure a high-level system analysis class diagram showing these classes and the relationships between them.

Now we will discuss the details of the architectural classes. Some self-explained classes are not described in detail. Only the important classes are listed. Some of the less important classes that are not listed in this section will be described in the future versions. The main classes involved in the application are:

1. MainGui: Which is the main graphical user interface class. It's mainly derived from JFrame from the Java libraries.

2. Welcome: It is another Interface class that represents the welcome screen. It's the first screen that the user sees when starting the application.

3. Vcoeficient: It is an entity class used for calculating the variable coefficients in the equation.

4. VariCoefODE: It is the used to retrieve and store the values of the coefficients in the equation as well as the right hand side value.

5. VarCoefODESolver: It is used to solve the equation created by the user. It is an abstract class for the three algorithm subclasses we included in the project.

6. VarCoefODESolver1: It is used to solve the equation according to the Euler algorithm.

7. VarCoefODESolver2: It is used to solve the equation according to the Crank-Nicolson algorithm.

8. VarCoefODESolver3: It is used to solve the equation according to the Predictor-Corrector algorithm.

9. dialogError: It is used to handle any error that happens during execution time.

10. ODE: It is used to maintain the order of the equation that is to be solved.

The classes are explained in more details in appendix C. The attributes, methods, subclasses, super-classes, and a more detailed description are all part of the detailed description of the classes in the appendix.

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the use case scenario and the sequence of messages exchanged between the objects needed to carryout the functionality of the scenario.

In UML an object in a sequence diagram is drawn as a rectangle containing the name of the object underlined. An object can be named in one of three ways: the object name, the object name and its class, or just the class name (anonymous object). In this document we will use the second way (1).

Each object also has its timeline represented by a dashed line below the object. Messages are represented by arrows, which point from the client (sender of the message) to the supplier (receiver of the message).

Objects interact by calling methods in other objects. Objects use this mechanism to send information, request a service, or request information. While messages are invoked from the calling objects, the implementations are in the receiving objects.

In Object-Oriented design methodology we prepare use-case descriptions and scenarios first. Then we move on to either class diagrams or sequence diagrams (5). There are two approaches: Problem-centric approach and Data-centric approach. The former moving from a sequence diagram to a class diagram and the latter moving from a class diagram to a sequence diagram. In this document, we take the Data-centric approach. Use Case Diagrams and Class Diagrams from previous sections are used as inputs (3).

To create a sequence diagram, we begin by identifying a set of classes that will be involved in a scenario. Once we have identified the objects involved in the scenario, we list the objects in a line along the top of a page in the order they are used in the scenario. The leftmost object should represent an actor who starts the scenario.

The next three pages will show the most important sequence diagrams in the system.

The Input Data Sequence Diagram (first part: excluding functions):

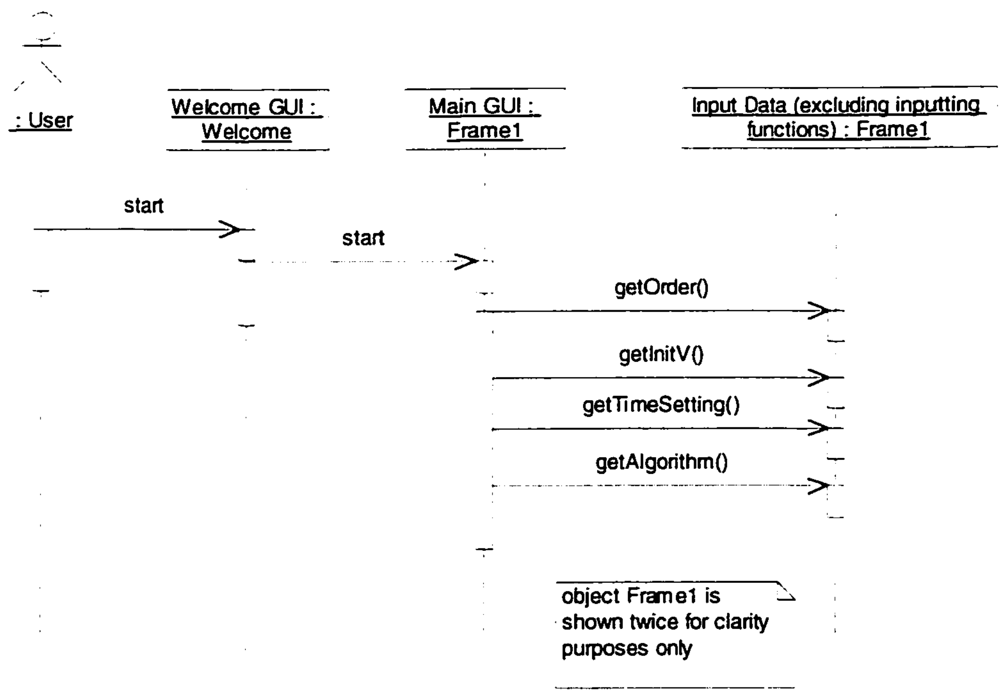

: User
Welcome GUI : Welcome
Main GUI : Frame1
Input Data (excluding inputting functions) : Frame1
start
start
getOrder()
getInitV()
getTimeSetting()
getAlgorithm()
object Frame1 is shown twice for clarity purposes only

Figure 9: Input Data (part 1) Sequence Diagram

The Input Data Sequence Diagram (second part: only functions):

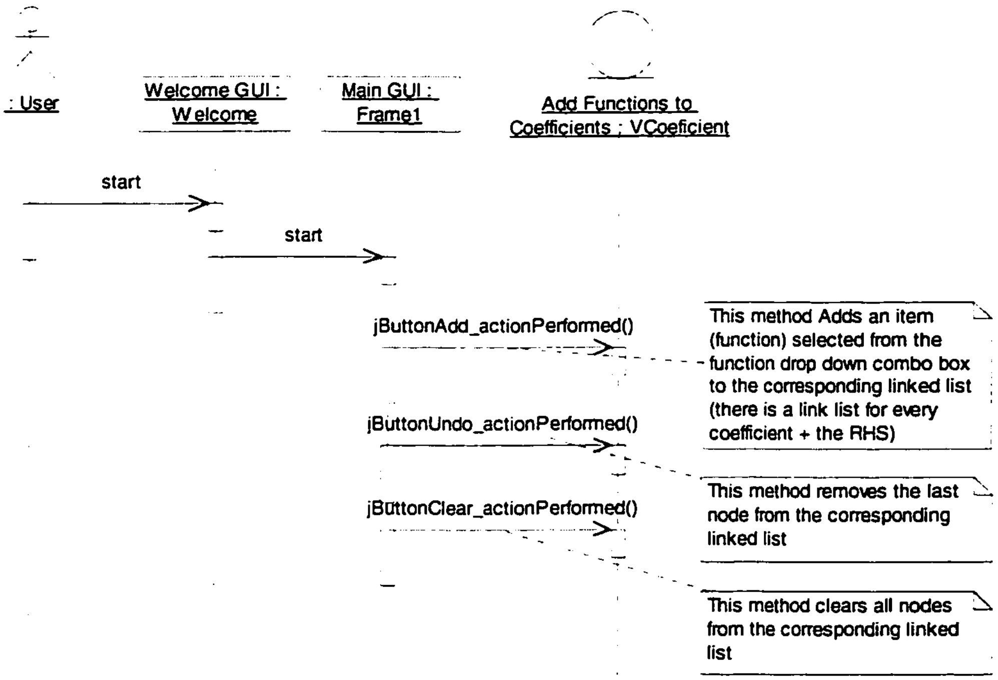

: User
Welcome GUI : Welcome
Main GUI : Frame1
Add Functions to Coefficients : VCoeficient
start
start
jButtonAdd_actionPerformed()
jButtonUndo_actionPerformed()
jButtonClear_actionPerformed()
This method Adds an item (function) selected from the function drop down combo box to the corresponding linked list (there is a link list for every coefficient + the RHS)
This method removes the last node from the corresponding linked list
This method clears all nodes from the corresponding linked list

Figure 10: Input Data (part 2) Sequence Diagram

The Start Calculation Sequence Diagram:

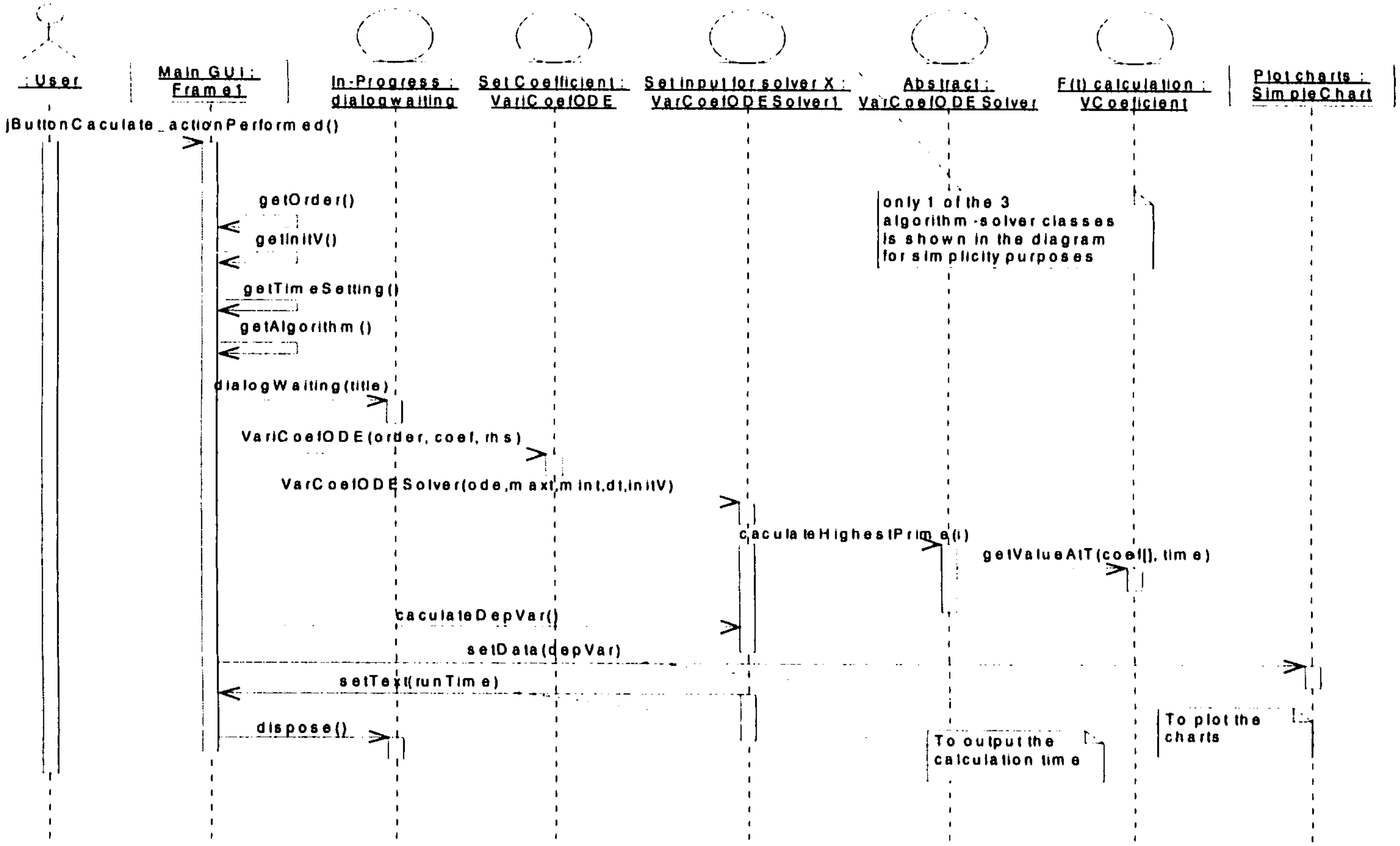

: User
Main GUI : Frame1
In-Progress : dialogwaiting
SetCoefficient : VariCoefODE
SetInput for solver X : VarCoefODESolver1
Abstract : VarCoefODESolver
F(t) calculation : VCoeficient
Plotcharts : SimpleChart
jButtonCaculate_actionPerformed()
getOrder()
getInitV()
getTimeSetting()
getAlgorithm ()
dialogWaiting(title)
VariCoefODE(order, coef, rhs)
VarCoefODESolver(ode,maxt,mint,dt,initV)
caculateHighestPrime(i)
getValueAtT(coef[], time)
caculateDepVar()
setData(depVar)
setText(runTime)
dispose()
only 1 of the 3 algorithm-solver classes is shown in the diagram for simplicity purposes
To output the calculation time
To plot the charts

Figure 11: Start Calculation Sequence diagram

A collaboration diagram is an alternate way to show a scenario. This type of diagram shows object interactions organized around the objects and their links to each other. A collaboration diagram contains (4):

- Objects drawn as rectangles,
- Links between objects shown as lines connecting the linked objects

Messages shown as text and an arrow that points from the client to the supplier.

The Input Data Collaboration Diagram (first part: excluding functions):

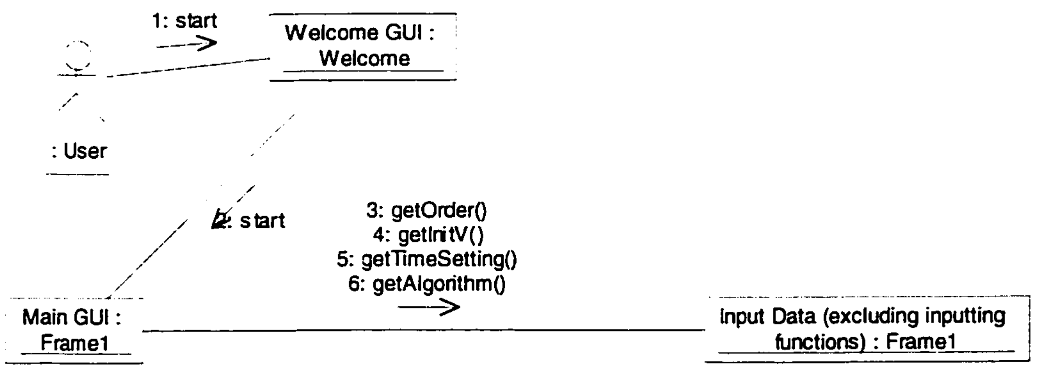


Figure 12: Input Data (part 1) Collaboration Diagram

The Input Data Collaboration Diagram (second part: only functions):

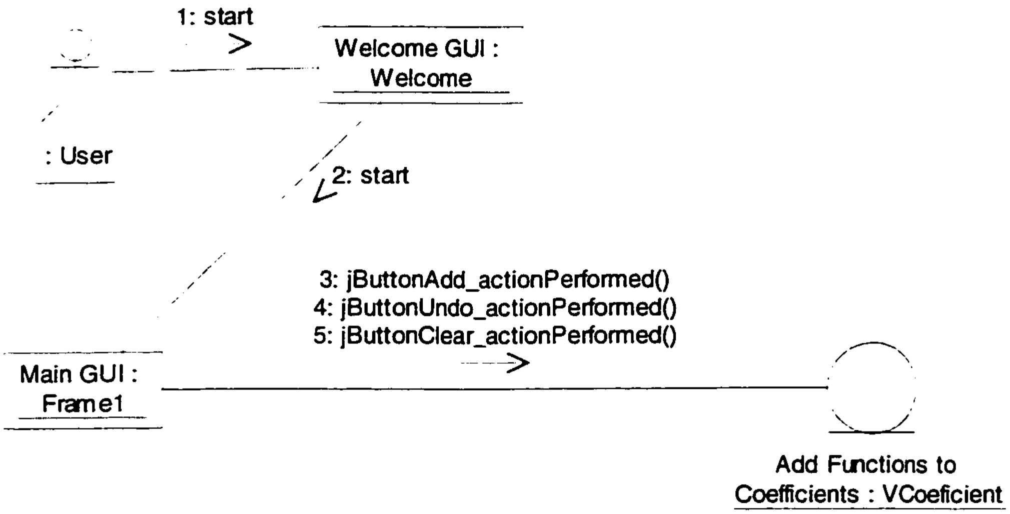

1: start
Welcome GUI :
Welcome
: User
2: start
3: jButtonAdd_actionPerformed()
4: jButtonUndo_actionPerformed()
5: jButtonClear_actionPerformed()
Main GUI :
Frame1
Add Functions to
Coefficients : VCoeficient

Figure 13: Input Data (part 2) Collaboration Diagram

The Start Calculation Collaboration Diagram:

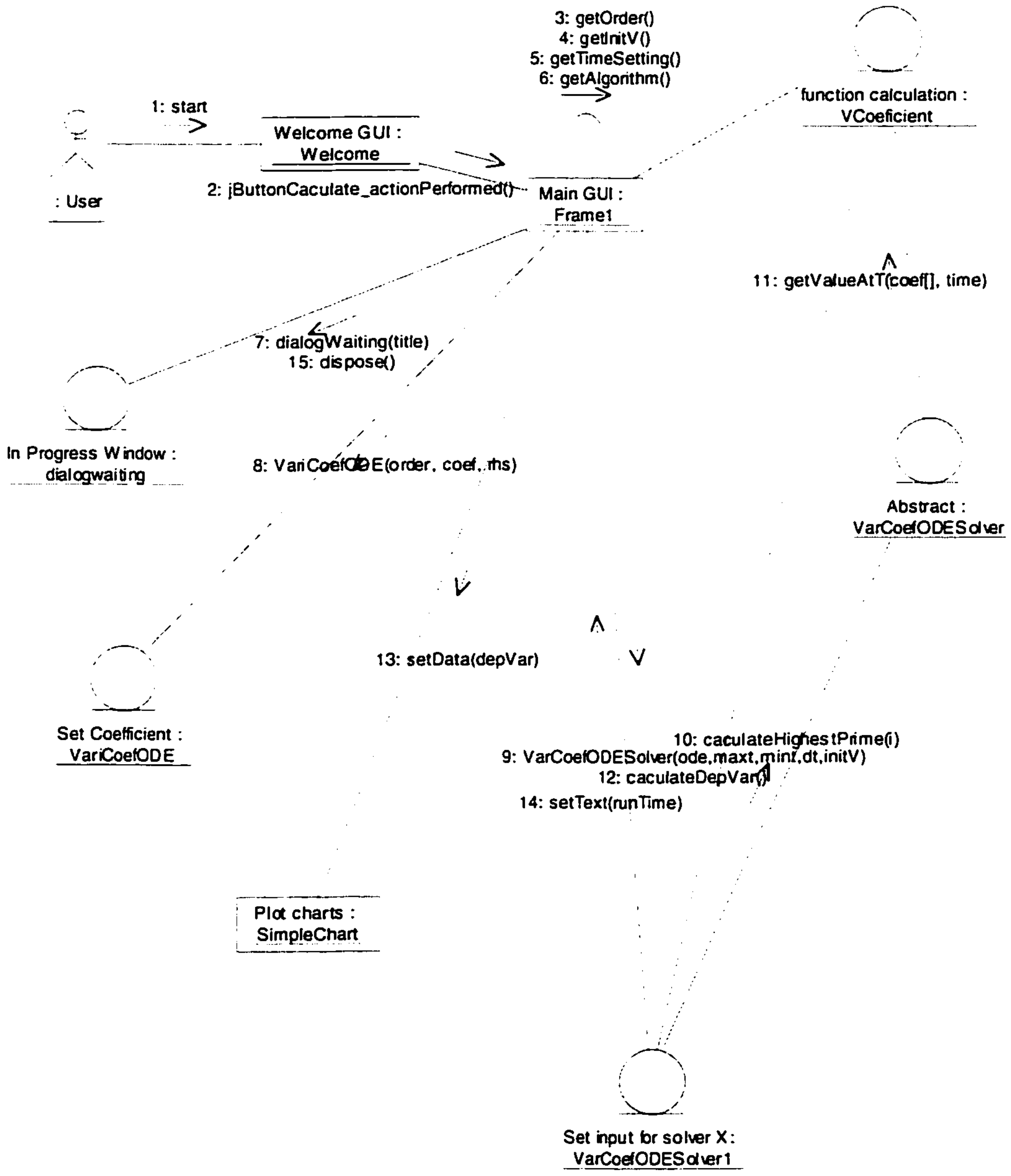


Figure 14: Start Calculation Collaboration Diagram

For a more detailed description in the three main sequence and collaboration diagrams refer to appendix D.

State and activity diagrams are very important when designing a system since they show the object life cycle (1). We will take every object by itself and study its changes when different actions are called on it. The state diagram will show the object in its different states, what kind of action was done on it, and conditions needed in order for the state to change. The activity diagrams will show what kind of actions can be done on an object as well as the conditions needed for the activity to happen. Since both type of diagrams show the same thing in different ways we decided to only include the state diagrams of certain objects. The three diagrams are shown on the next three pages:

VarCoefODESolver state diagram:

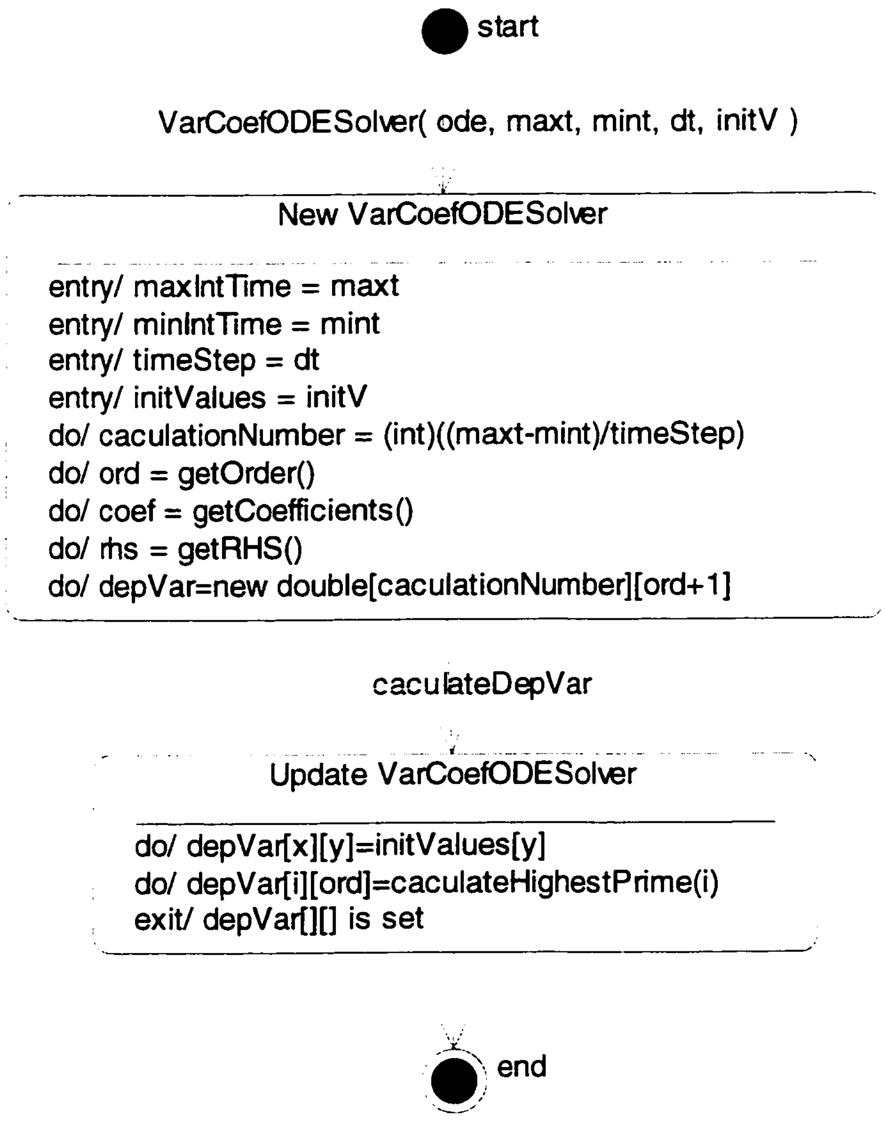

start
VarCoefODESolver(ode, maxt, mint, dt, initV)
New VarCoefODESolver
entry/ maxIntTime = maxt
entry/ minIntTime = mint
entry/ timeStep = dt
entry/ initValues = initV
do/ caculationNumber = (int)((maxt-mint)/timeStep)
do/ ord = getOrder()
do/ coef = getCoefficients()
do/ rhs = getRHS()
do/ depVar=new double[caculationNumber][ord+1]
caculateDepVar
Update VarCoefODESolver
do/ depVar[x][y]=initValues[y]
do/ depVar[i][ord]=caculateHighestPrime(i)
exit/ depVar[][] is set
end

Figure 15: VarCoefODESolver State Diagram

VariCoefODE state diagram:

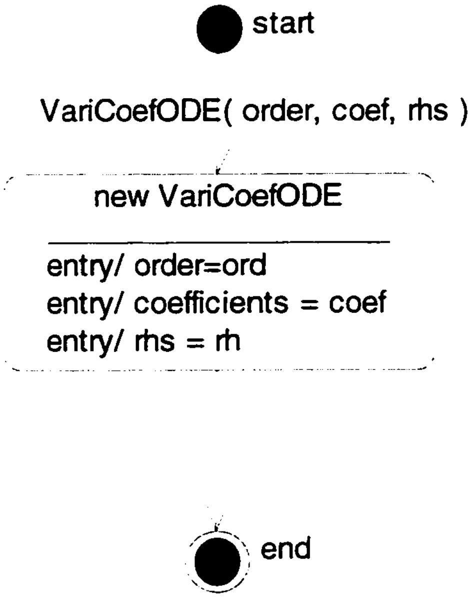


Figure 16: VariCoefODE State Diagram

VCoefficient state diagram:

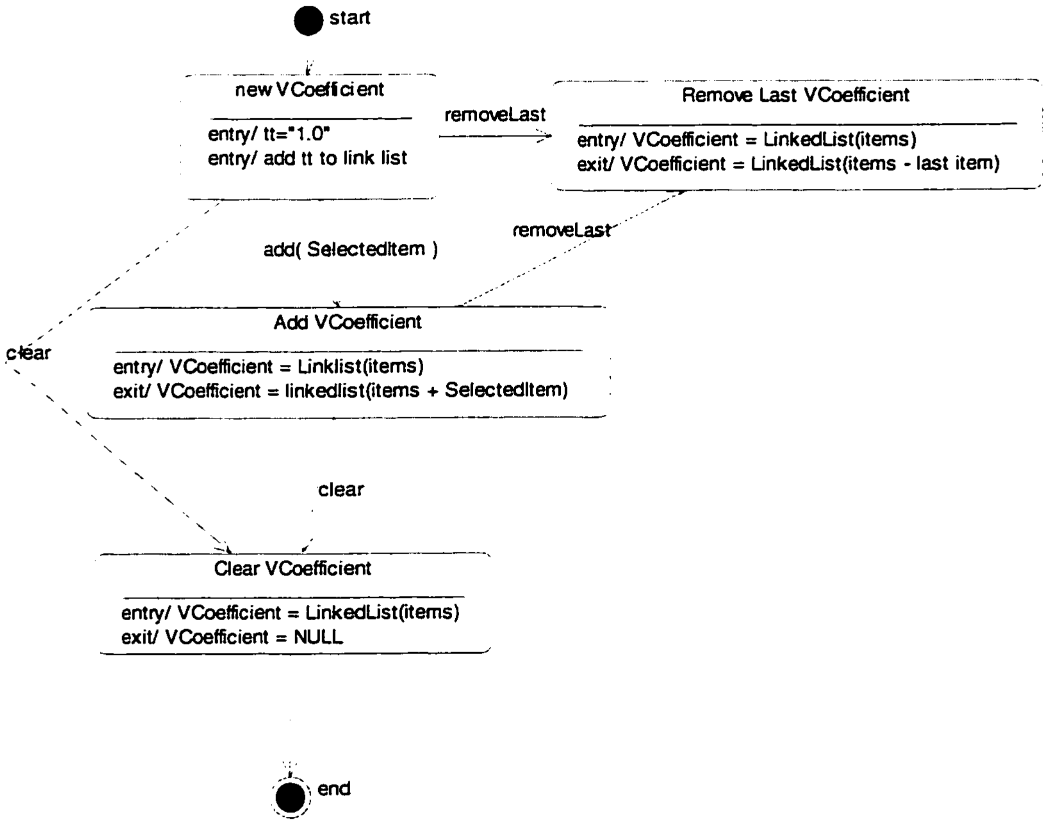

start
new VCoefficient
entry/ tt="1.0"
entry/ add tt to link list
removeLast
Remove Last VCoefficient
entry/ VCoefficient = LinkedList(items)
exit/ VCoefficient = LinkedList(items - last item)
removeLast
add(SelectedItem)
Add VCoefficient
entry/ VCoefficient = Linklist(items)
exit/ VCoefficient = linkedlist(items + SelectedItem)
clear
clear
Clear VCoefficient
entry/ VCoefficient = LinkedList(items)
exit/ VCoefficient = NULL
end

Figure 17: VCoefficient Statc Diagram

A Component Diagram shows the organization and dependencies among a set of components that are the physical implementation of the system. It is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. A component typically represents the physical packaging of otherwise logical elements, such as classes, interfaces, and collaborations (2).

A component diagram shows the organization and dependencies among a set of components. Component diagrams are essentially class diagrams that focus on a system's components.

The diagram on the next page is the EPS component diagram:

The EPS system component diagram:

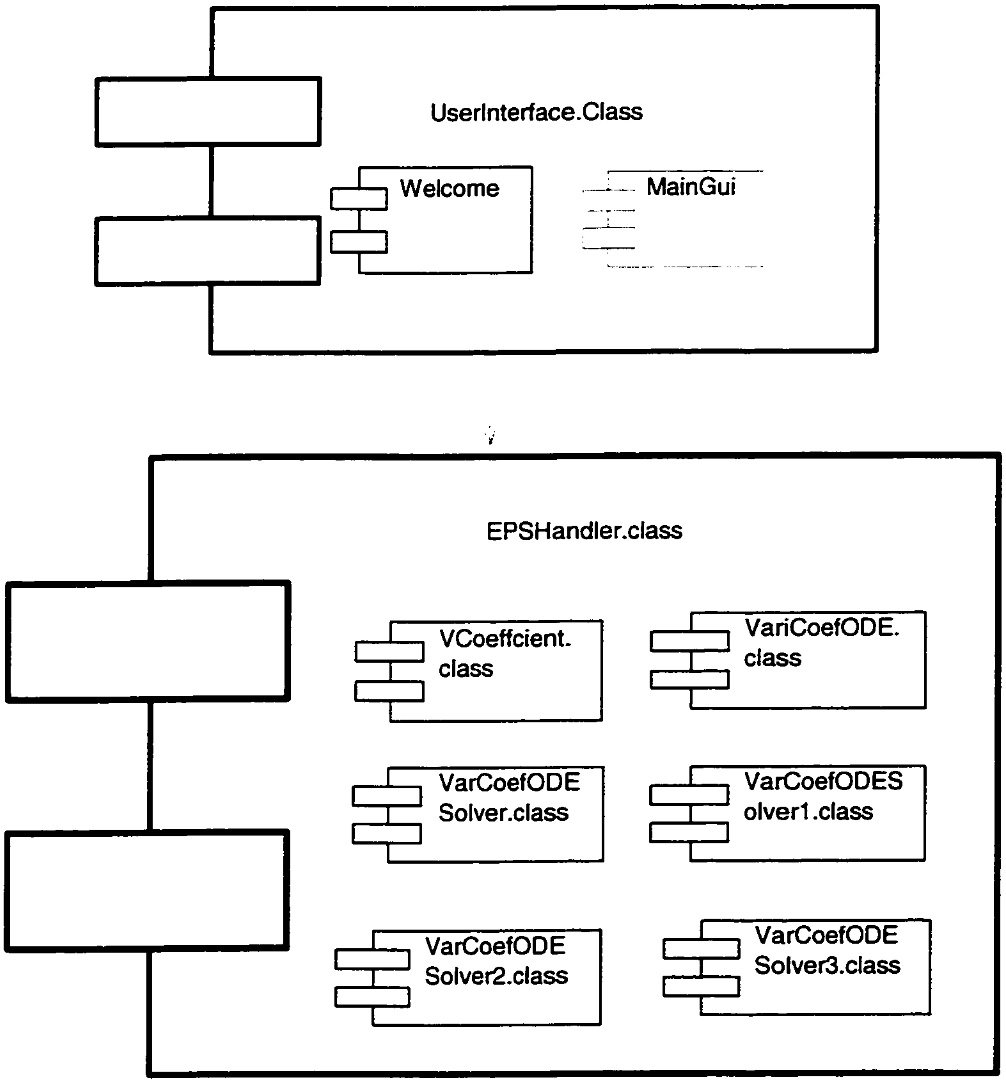


Figure 18: The EPS Component Diagram

Chapter 5

CHAPTER 5: DESIGN AND CODE CHANGES

This chapter focuses on the design and implementation changes that are done in order to enhance performance. Some of the issues are discussed in previous chapters but not in details. When starting a new project it is very common that the first version has some problems. There are lots of reasons why some applications do not have good performance. Some of the reasons might be technical; others include time and resources.

Due to the competitive environment we live in, some companies rush in developing application and put them in to the market even though they do not perform well or the contain certain defects. This is, obviously, because they want to control the market. This is the nature of most software development companies. Another reason performance is not seriously taken in to consideration is because there are no resources available to allocate the time and human power for it.

Fortunately EPS does not have these problems. Enough resources and time are allocated for the project because the primary requirement is performance. Certain issues related to the design and coding of the application are encountered.

Design Changes

In the original design there were separate classes for constant coefficient and for variable coefficient. We had two classes: ConstCoefODE and VarCoefODE. So if the coefficient was a constant we use ConstCoefODE; otherwise we use VarCoefODE. This made the application run slower because of the existence and creation of two separate classes. The application must use these two classes if the equation created had constant and variable coefficients at the same time. We were able to join these two classes in one class: VariCoefODE. The methods needed for variable coefficients and constant coefficients were also combined and we had one generic class with generic attributes and methods to work with any time of coefficients.

Another design bottleneck was the equation solver. Since we had three different algorithms or solvers we initially created three separate classes: EulerSolver, CrankNicSolver, and PredCorSolver. These three algorithms had some common methods and attributes. It was taking more resources to load three similar classes to the memory and more processing time. The design was changed to have an abstract solver class that has the common methods and attributes and the three different solvers derived from that class. This led us to create an abstract class VarCoefODESolver and three different classes (VarCoefODESolver1, VarCoefODESolver2, VarCoefODESolver3).

Error handling was another design modification that was done to enhance performance. It was also needed to me more generic for future enhancement. Initially we were handling errors from within class methods. We then created a new class, ErrorHandler, which deals with all types of error thrown out from the application. A message and an error type are passed to the ErrorHandler class and the error is established only in that class. This made the error handling code for all the application generic and centralized in one class. It is also more efficient for future additions to the application.

Coding Changes

There were a lot of changes done to the code; some were important and made a noticeable enhancement in performance. Other changes were small but definitely sped up the calculation time too. We already discussed why it is important to be careful while programming such applications because a small mistake can make a big difference. In chapter four we gave an example about how a variable that was defined as a long instead of a short can make performance drop dramatically.

After coding every method used in the EPS system we went through the code again and made sure that:

1. All unused variables were taken out

2. Variables used have the right size

3. The code in loops is kept to the minimum

4. Calculations done are all necessary

5. Calculations done once and no duplication of processing

We also changed some pieces of code to increase performance. When adding the coefficients as variable functions we initially used an array. It was taking more resources and processing time because we did not know how big should it be. It was also needed through out the processing time so it has to be available from the start till the end of the calculation. This array was often accessed and manipulated so it was one of the main bottlenecks of the application. We then found out a better way of working with coefficients; linked lists. Linked lists can grow dynamically. If a function is as small as sin(t), a linked list can be 2 nodes big whereas an array would be of fixed length (initially 100 elements). After calculating how much sin(t) was, the two nodes (sin, t) would be deleted from the linked list (popped) and the actual value is added. This made function calculation much faster, easier, and more generic.

Another major modification was in the plotting of the diagrams. Initially we were plotting every single point with a third party software/library. Plotting the diagrams was taking about 70% of the total time needed for the whole calculation to finish. It was not acceptable so we investigated other ways of plotting them. A thorough research was done on different libraries that are available in the market. Initially we used a charting library offered by Sitraka (Jclass ServerChart). This library was then replaced by another since it did not come with the source code, therefore, we don't have the flexibility of modifying it to meet our requirements. We looked for open-source libraries that come with the GNU Lesser General Public License (LGPL) which guarantees the freedom to share and change free software--to make sure the software is free for all its users. The most important libraries that were researched include:

1. JetChart offered by Jinsight Informatica Ltda,
2. Chart2D offered by sourceforge (17),
3. JopenChart also offered by sourceforge,
4. And JfreeChart offered by Object-Refinery (18).

The library that suited our project the best was JfreeChart. It was fast and contained all the features that we needed. Small modifications to the library were made and it was added to the program as part of the same package. This was done for two reasons:

- To make the application easier to use without the need of plugging an extra library.
- To have the source code as part of the application code for future modifications.

It is important that the libraries used are not corporate dependent. The initial release of the application contained Borland libraries since the graphical user interface was build using Borland's JBuilder. The layout class included in com.Borland.jbcl library of JBuilder version six was needed for displaying charts and other GUI elements. A research was made to find other ways to design the GUI without being dependant on Borland's libraries. The requirement was to find free libraries that are available in the market that can display the graphs quickly and efficiently. The best libraries that were freely available in the market were the Java swing and awt libraries. We made the modifications and tested the application thoroughly again for performance and we found out that with Java libraries the application ran almost as fast as the previous version.

The current version of the EPS application uses libraries that are free and available in the market. The only libraries needed to compile the application are included in the Java^{TM} 2 SDK Standard Edition or simply J2SE by Sun Microsystems.

Chapter 6

CHAPTER 6: THE RESULT

Before we discuss the result we will just explain how to run the EPS program. The main thing that needs to be done is to input the required equation which include setting the order of the equation, the coefficients, the initial values, the time settings, and the algorithms to be used. Most of the inputs are straightforward except setting up the coefficients. In order to set a specific coefficient of an equation the user needs to select the corresponding radio button and then build the function from the dropdown menu. If numerical values are needed the user can simply type the value needed in the dropdown box and click on the 'add' button. If a mistake is done the 'undo' button can be use. The user can clear the whole coefficient by clicking on the 'clear' button. A snapshot of the main graphical user interface can be viewed in Appendix A-2.2. A ReadMe.txt file is available with the program to explain how to install and run the EPS application.

Designing a high performance mathematical application will always be a challenging task. After reaching an optimal and generic design it is time for us to test the performance. An additional functionality is added to the program to compute the time needed to calculate the plotting points and draw the diagrams. This time depends on the hardware that we are running the application on as well as the complexity of the equation and the solver (the algorithm). In the current version of the program we limited the number of points we calculate to three million so we do not run out of memory. Reading the memory available on the machine used and setting the limitation accordingly can solve this problem. This is recommended for the next application release.

As mentioned in previous chapters the hardware needed to run the application on does not need to be powerful. It can range between a Pentium I, 33Mhz to a Multiprocessor machine. We decided to test the program on a Pentium II, 266 MHz with 128 megabytes of RAM. We also wanted to test its performance with a more powerful machine, Pentium IV 1.2 GHz with 512 megabytes, to see how much faster the application can perform.

Moreover, the elements that affect the resulting calculation times are:

1. The order of the equation,

2. The complexity of the coefficients and the right hand side of the equation,

3. The time-step and the time range,

4. And the algorithm selected to solve the equation.

On a Pentium II, 266 MHz with 128 megabytes of RAM the performance was high. The table below lists a few test criterion that were performed and the approximate calculation time each test needed. Each test was performed 4 times and the average was taken and rounded to the nearest ten milliseconds for more precision.

Order	Coefficient complexity	Algorithm	Time Range	Time Step	Result (ms)
2	Constant	Euler	0-20	0.1	≈ 220
2	Constant	Crank-Nicolson	0-20	0.1	≈ 220
2	Constant	Predictor-	0-20	0.1	≈ 220
2	Constant	Euler	0-20	0.0001	≈ 4050
2	Constant	Euler	0-500	0.01	≈ 10170
5	Constant	Euler	0-500	0.01	≈ 19960
5	Constant	Predictor-	0-500	0.01	≈ 21730
2	Variable-Simple	Euler	2-20	0.1	≈ 270
2	Variable-Simple	Crank-Nicolson	2-20	0.1	≈ 270
2	Variable-Simple	Predictor-	2-20	0.1	≈ 380
2	Variable-Simple	Euler	2-5	0.00001	≈
2	Variable-Simple	Predictor-	2-5	0.00001	≈
2	Variable-Simple	Euler	2-30	0.001	≈ 19610
2	Variable-Simple	Predictor-	2-30	0.001	≈ 38220
3	Variable-	Crank-Nicolson	2-50	0.1	≈ 1440
3	Variable-	Predictor-	2-50	0.1	≈ 2520
3	Variable-	Crank-Nicolson	2-50	0.005	≈ 21770
3	Variable-	Predictor-	2-50	0.1	≈ 43910

Table 3. EPS test results table

As discussed in chapter three the coefficients of an equation can be constants or variables. A variable coefficient can be a simple one like sin(t), or a more complex one like cot(log(t)) * t^e.

The application was also tested on a much faster machine, Pentium IV, and the results were astonishing. The time needed for each test was cut eight to fifteen times. Error handling was also tested thoroughly. The memory needed to solve each equation depends on the amount of data that must be temporarily stored for plotting the diagrams. When there is not enough memory available the application outputs an error message to notify the user with the problem. The application also notifies the user if there is any problem with the equation. For example, if one of the coefficients was log(t) and t was set to zero the user gets a message that the coefficient are invalid. The application never failed to work. We did not encounter any problem while doing the last phase of testing.

The application has shown very high performance on a Pentium II machine. For example, the following equation takes less than 20 seconds:

- $\sin(t)\frac{d^2\theta}{dt^2} + \log(t*e)\frac{d\theta}{dt} + 3t\theta = \cos(t)$

We are able to calculate 150,000 points and plot them in less than 20 seconds. This number drops to 2.4 seconds on a Pentium IV 1.2 GHz computer. The plotting time of this equation using Sitraka, a third party software, takes almost 70% of the total time. Almost three of the four seconds are needed to plot the above equation. This drops considerable when working with more complex equations since the plotting time does not vary as much as the calculation time does.

Chapter 7

CHAPTER 7: FUTURE WORK

This section presents some issues raised by this research, and suggests related problems we believe need to be addressed.

One of the main issues that need to be discussed is the programming language. As mentioned before, we are not sure if the application performs best with Java as the programming language. Earlier we proposed another version that has the same interface (Java) but a C++ engine. There must be an interface layer between the two; Java Native Interface.

Java Native Interface Description

Sun micro-systems describe Java Native Interface (JNI) as: "the native programming interface for Java that is part of the JDK. By writing programs using the JNI, you ensure that your code is completely portable across all platforms" [15]. This interface allows programs written in Java to communicate with other programs written in other languages like C, C++, and assembly. Programmers usually use this interface in situations where applications cannot be entirely programmed in Java. In our situation, we want to use this interface because we are predicting that our application run faster using C++ for computation.

JNI is part of the Java Developer Kit (JDK). It is easy to see that the JNI serves as the glue between Java and native applications. The following diagram shows how the JNI ties the C side of an application to the Java side [9].

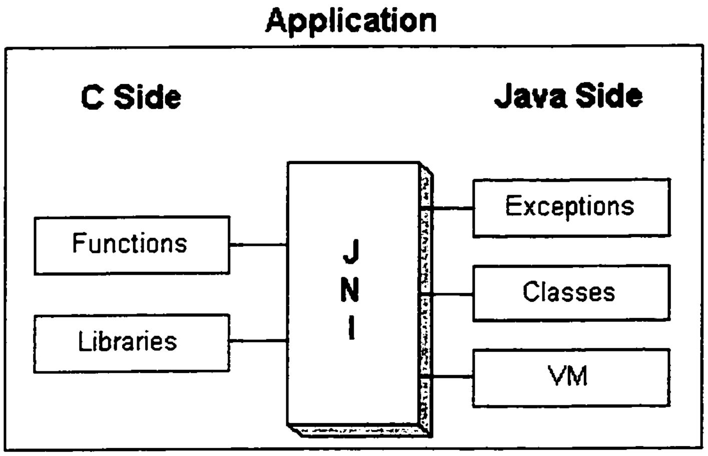


Figure 19: JNI in action

JNI adds to the complexity of the application [9]. Certain considerations have to be taken and this is why programmers only use this functionality when it is a necessity.

The Overall JNI Process

In order to embed C++ code and use JNI to call the C++ methods certain process has to be taken [9]:

1. Write a Java class that has a native method for example this JNIODEExample.java:

2. Compile JNIODEExample.java.

3. Generate the JNIODEExample.h header file containing the prototypes of the C++ methods you must write with.

   - javah.exe -jni -o JNIODEExample.h cmp.JNIExper. JNIODEExample

4. Write a C or C++ class for example: JNIODEExample.c

5. Link that C++ code it calls into a DLL. The dll may contain methods from many different classes.

6. Pre-install the DLL on the browser's classpath, e.g. \JDK1.3\jre\bin\ext (for the JDK 1.4 plug-in), WINNT\java\trustlib.

7. Use the Java native methods as ordinary Java methods.

Performance Assumption

Java Native Interface adds a new layer to the application. This layer slows the performance of the application down considerably. Using C++ for computation will speed up performance but it might not be enough to overshadow the slowness that the extra layer adds. This study can only be performed after having two running versions of the application. Even at that time it might be difficult to decide which version performs better since it depends on the complexity of the computation. To illustrate, lets take a look at the following two examples:

Version	Equation	Computation	Interfacing	Total
All-Java	Simple Eq. (X)	150 ms	N/A	150 ms
Java-JNI-C++	Simple Eq. (X)	50 ms	200 ms	250 ms
All-Java	Complex Eq. (Y)	5,000 ms	N/A	5,000 ms
Java-JNI-C++	Complex Eq. (Y)	800 ms	200 ms	1,000 ms

Table 4. Performance comparison between All-Java versus Java-JNI-C++

The All-Java version performs better with the simple Equation X, whereas the Java-JNI-C++ version performs better with the complex equation Y. This assumption is valid but the figures given in the example are just for illustration.

Additional Algorithms

The algorithms that are provided in the application are not the only algorithms that can solve such mathematical equations though they are some of the most popular. The application was designed in a way to easily plug in additional algorithms. Whenever an additional algorithm is to be created certain changes has to be made. They are the following:

- Add the solver as an extended class from the abstract class VarCoefODESolver to the program,
- Add the solver class to the interface in the algorithm combo-box in the MainGui class,
- Add the solver class to the switch case in the jButtonCalculate method.

Chapter 8

CHAPTER 8: CONCLUSION

In the course of this major report we have discussed the importance of performance in certain mathematical applications. We defined and analyzed the problem as well as established a solution for solving all sorts of equations by constructing an optimal framework that can be a starting point for future enhancement. Along with the research we built an application to test our methodology. The iterative process that we followed helped us in understanding the problem correctly, finding the best way to interpret this problem, as well as testing our findings thoroughly. We also researched what resulted from the application and discussed what can be done to improve it.

There are two major themes in this work. The first has been elaborated on extensively throughout this work which concerns being able to build an application to simulate the movement of object according to external conditions. This application must perform rapidly with minimum resources. The conditions can be described in an equation that is inputted to the application. The output is a set of diagrammatic interpretations of the result and performance statistics.

The second major theme is the proof that these applications can be done and is accessible to all who need it. It concerns the development of numerical methods to solve the equations using certain algorithms. Three popular algorithms were implemented and the probability of adding new algorithms to the program is high due to the generality of the design. Algorithms vary in their complexity and the accuracy of their outputs. The proof is a running application done using object-oriented design for enhancement purposes. Java was the programming language that was used.

The study described the design in details and emphasized on the performance issues that were encountered and the modifications that were done in order to solve them whether they were coding or design changes. The work also includes a general overview of a different coding approach. This approach uses different programming language like Java, Java Native Interface, and C++. The reason behind the inclusion of this alternative approach is to direct the future enhancers of this work to different methods for possible performance enhancement.

APPENDIX A: IMPORTANT APPLICATION FUNCTIONALITIES

A-1: "welcome.java" description

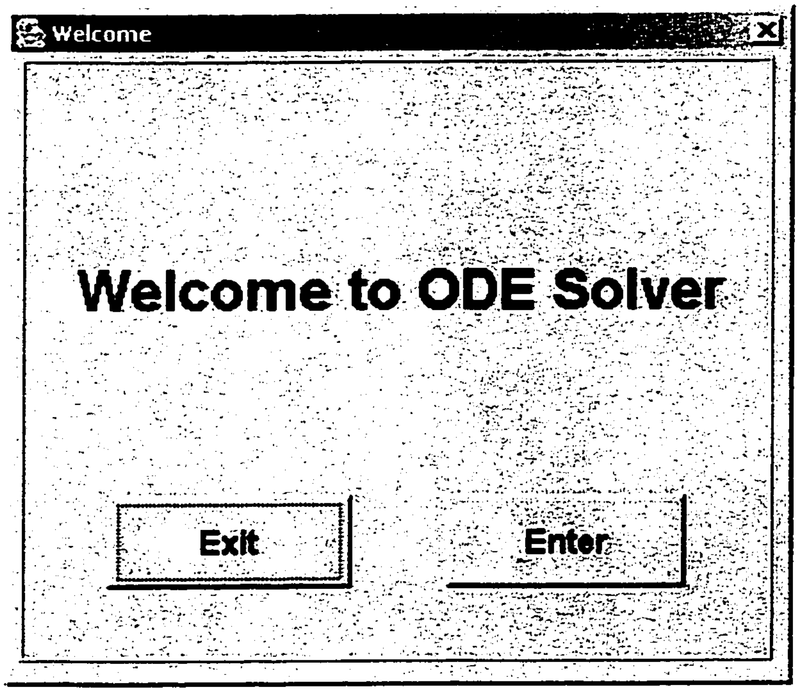
This welcome class is a graphical interface screen that the user gets when starting the application. It displays the name of the application as well as two buttons to either exit the application or enter the main menu.

A-1.1: "welcome.java" source code

```
// this is the first GUI class (window) we get when we start the application
// it just contains 2 buttons to start or exit the application
public class Welcome extends JDialog {
 JPanel panel1 = new JPanel();
 XYLayout xYLayout1 = new XYLayout();
 Border border1;
 XYLayout xYLayout2 = new XYLayout();
 JButton jButton1 = new JButton();
 JButton jButton2 = new JButton();
 JLabel jLabel1 = new JLabel();

 // constructor. just to create the frame/window
 public Welcome(Frame frame, String title, boolean modal) {
  super(frame, title, modal);
  try {
   jbInit();
   pack();
  }
  catch(Exception ex) {
   ex.printStackTrace();
  }
 }

 public Welcome() {
  this(null, "", false);
 }
 // used to build the first window and set the colors, text, size....
 void jbInit() throws Exception {
  border1 = new EtchedBorder(EtchedBorder.RAISED, Color.white, new
Color(134, 134, 134));
```

```
    panel1.setLayout(xYLayout2);
    this.getContentPane().setLayout(xYLayout1);
    panel1.setBorder(border1);
    jButton1.setFont(new java.awt.Font("Dialog", 1, 16));
    jButton1.setText("Enter");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButton1_actionPerformed(e);
      }
    });
    jButton2.setFont(new java.awt.Font("Dialog", 1, 16));
    jButton2.setText("Exit");
    jButton2.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButton2_actionPerformed(e);
      }
    });
    jLabel1.setFont(new java.awt.Font("Dialog", 1, 28));
    jLabel1.setText("Welcome to ODE Solver");
    xYLayout1.setWidth(386);
    xYLayout1.setHeight(308);

this.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
    panel1.add(jButton1,  new XYConstraints(210, 216, 120, 46));
    panel1.add(jButton2, new XYConstraints(43, 216, 120, 46));
    panel1.add(jLabel1,  new XYConstraints(27, 84, 346, 56));
    this.getContentPane().add(panel1,   new XYConstraints(5, 3, 375, 301));
  }

  // if the Enter button is clicked
  void jButton1_actionPerformed(ActionEvent e) {
    this.dispose();
  }
  // if the Exit button is clicked
  void jButton2_actionPerformed(ActionEvent e) {
    System.exit(0);
  }
}
```

A-1.2: "welcome.java" Screenshot



A-2: "MainGui.java" description

The "MainGui" class is the main interface with the user. It is divided in to seven containers for user friendliness. The containers are either input or output containers and they are the following:

1. The "Set Order" container where the user chooses an order for the equation. The order can be 2, 3, 4, or 5. A combo-box was used to reduce ambiguity, so the user can only choose an appropriate input.
2. The "Set Coefficient" container is used to set the coefficients in the equation. Depending on the order set in the "Set Order" container, the coefficients available to set gets activated. For example, if the order was set to 3 then only coefficients that get enabled are (0, 1, 2, and 3). The RHS (right hand side) is also activated. Coefficients can be constants, functions or a mixture of both. When setting up a specific coefficient in the equation the user builds the function from a combo-box of available operations (For available operations check chapter three). If the function the user built is invalid the application throws an error message upon execution. The user has the capability to:
    a. "add" operations to the function
    b. "undo" the last add
    c. And "clear" the whole function or coefficient.

3. The "Initial Value" container is where the user inputs the initial values of theta and its family. The textboxes used in this container to input the values are also enabled or disabled according to the order of the equation.
4. The "Time Setting" container contains three text boxes to input the starting time, the ending time, and the time step.
5. The "Algorithm Setting" container has a combo-box to select which algorithm is to be used to calculate the output.
6. And the last container that consumes most of the screen is the output container that has up to six diagrams, also depending on the order of the equation, which shows the result of the calculation. The diagrams are:
    a. Theta vs. theta prime
    b. Theta vs. time
    c. Theta prime vs. time
    d. Theta double prime vs. time
    e. Theta triple prime vs. time
    f. And theta quadruple prime vs. time

The "start calculation" button is at the bottom of screen and does not belong to any specific container since it uses all the inputs of the input containers to calculate the points needed and plots them in the output container. It also calculates the time that was needed to find the result and displays it in a "Calculation Time" text box in milliseconds.

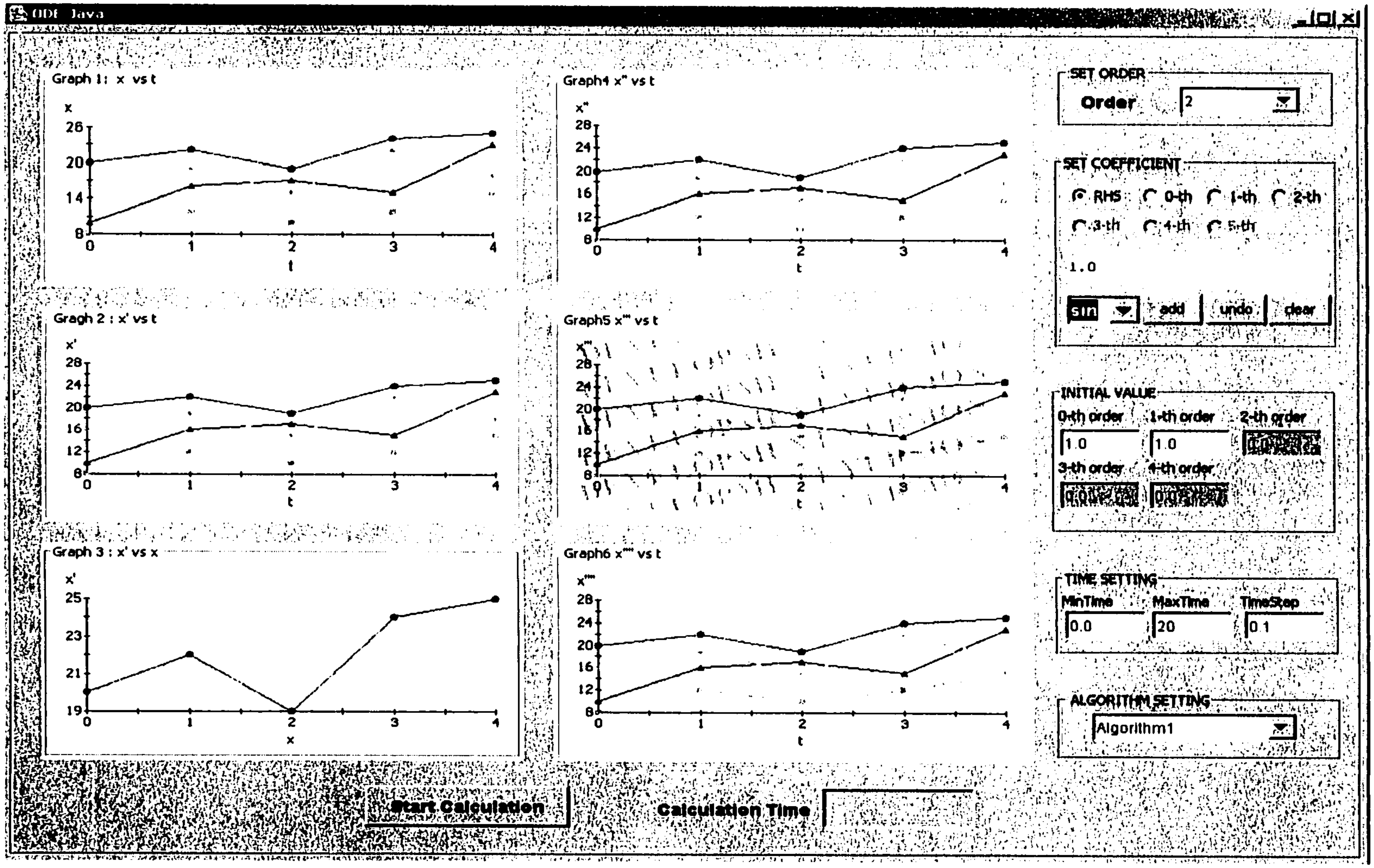
A-2.1: Important functions in “MainGui.Java”

```
// the main function. This starts the process of calculation.

void jButtonCalculate_actionPerformed(ActionEvent e) {
        // get Start Time
        Calendar startCalendar = Calendar.getInstance ();
        Date date1 = startCalendar.getTime();
        long startTime = date1.getTime();

        // display waiting dialog
        dialogWaiting dialogwaiting = new dialogWaiting(this, "In Excecution,
        Please Waiting...", false);
        dialogwaiting.setLocation(300, 250);
        dialogwaiting.setVisible(true);

        try     {
                getOrder();
                getInitV();
                getTimeSetting();
                getAlgorithm();
                if((order + 1) * (maxt - mint) / dt > 3000000) throw new
                Exception("out of memory");
                if (coef[order].size()==1 && Double.valueOf (coef[order].getFirst().
                toString().trim()).doubleValue() == 0)
                        throw new Exception("the highest coefficient is 0");
                VariCoefODE ode = new VariCoefODE(order, coef, rhs);
                // we instantiate the odesolver and we decide on which algorithm to
                use
                // (meaning which class to create VarCoefODESolver1, 2 or 3
                depending on
                // the user selection
                VarCoefODESolver odesolver = null;
                switch(algo)
                {
                        case 1:
                        odesolver = new VarCoefODESolver1(ode, maxt, mint, dt,
                        initV);
                        break;
                        case 2:
                        odesolver = new VarCoefODESolver2(ode, maxt, mint, dt,
                        initV);
        break;
        case 3:
        odesolver = new VarCoefODESolver3(ode, maxt, mint, dt,
        initV);
        break;
    }
    odesolver.calculateDepVar();
    double[][] depVar = odesolver.getDepVar();
    // the first three charts are a must
    simpleChart1.setData(getArray(depVar, 1));
    simpleChart2.setData(getArray(depVar, 2));
    simpleChart3.setData(getArray(depVar, 3));
    // the rest of the charts depends on the order. max is 6 charts
    if(order < 3) {
    simpleChart4.setData("Array 1 0");
    }
    else {
    simpleChart4.setData(getArray(depVar, 4));
    }
    if(order < 4) {
    simpleChart5.setData("Array 1 0");
    }
    else {
    simpleChart5.setData(getArray(depVar, 5));
    }
    if(order < 5) {
    simpleChart6.setData("Array 1 0");
    }
    else {
    simpleChart6.setData(getArray(depVar,6));
    }
    dialogwaiting.dispose();
    //get End Time
    Calendar endCalendar = Calendar.getInstance();
    Date date2 = endCalendar.getTime();
    long endTime = date2.getTime();
    long runTime = endTime - startTime;
    calculationTime.setText(Long.toString(runTime) + "ms");
}
// catches exceptions due to overflow or memory shortage or coefficient
problems
catch (Exception e1) {
    System.out.println(e1.getMessage());
    dialogwaiting.dispose();
```

```
        dialogError dialogerror;
        if(e1.getMessage() == "out of memory")
            dialogerror=new dialogError(this,"Error
            Message",true,"System doesn't have enough
            memory","Please change time setting and try again!");
        else {
            if(e1.getMessage() == "overflow")
                dialogerror=new dialogError(this,"Error
                Message",true, "Calculation result
                overflows","Please check the coefficient and try
                again!");
            else {
                if(e1.getMessage() == "the highest coefficient is 0")
                    dialogerror=new dialogError(this,"Error
                    Message", true, "The coefficient of highest
                    order can't be 0","Please check it and try
                    again!");
                else
                    dialogerror=new dialogError(this,"Error
                    Message", true,"Something is wrong with
                    coefficient","Please check it and try again!");
            }
        }
            dialogerror.setLocation(300,250);
            dialogerror.setVisible(true);
    }
}
```

```
// If the Add button is clicked the last selection is added to the end of the
// linked list
void jButtonAdd_actionPerformed(ActionEvent e) {
        String selectedItem = jComboBox2.getSelectedItem().toString();
        selectedItem = jComboBox2.getSelectedItem().toString();
        if (selectedItem != null) {
                if (RadioIndex == -1) {
                        rhs.add(selectedItem);
                        CoefDisplayArea.setText(rhs.getString());
                }
                else {
                        coef[RadioIndex].add(selectedItem);
                        CoefDisplayArea.setText(coef[RadioIndex].getString());
                        System.out.println("size" + coef[RadioIndex].size());
                }
        }
}
```

A-2.2: "MainGui.java" Screenshot



SET ORDER
Order 2

SET COEFFICIENT
● RHS ○ 0-th ○ 1-th ○ 2-th
○ 3-th ○ 4-th ○ 5-th
1.0
sin

INITIAL VALUE
0-th order 1.0
1-th order 1.0
2-th order
3-th order
4-th order

TIME SETTING
MinTime 0.0
MaxTime 20
TimeStep 0.1

ALGORITHM SETTING
Algorithm1

A-3: "Vcoefficient.java"

The "Vcoefficient.java" class is needed to manage and calculate the variable functions or constants that are added as the coefficients of the equation. There are two important functions in this class: getTempList and getValue. The former goes through the linked-list and overrides the constants added to the list with their values. The later function also goes through the list, evaluates the coefficient, and returns the value of the function or an error if the function is invalid.

A-3.1: Important functions in "VCoefficient.Java"

```
// this method is to substitute the values of constants in the linked list
// if the variable added to the list by the user is t or e or PI then
// we need to substitute them with the actual value. the actual value of
// t varies according to time so it's passed into the method. the value of
// E and and PI are fixed and we get them from the math library

private LinkedList getTempList(double t) {
        LinkedList templist=new LinkedList();
        for(int i = 0; i < size(); i++) {
                if(get(i) == "t") {
                        templist.add(Double.toString(t));
                        continue;
                }
                if(get(i) == "e") {
                        templist.add(Double.toString(Math.E));
                        continue;
                }
                if(get(i) == "PI") {
                        templist.add(Double.toString(Math.PI));
                        continue;
                }
                templist.add(get(i));
        }
        return templist;
```

```
    }


private double getValue(LinkedList templist) throws Exception {
    int i;
    int oldSize;
    double des;
    double opr1;
    double opr2;
    oldSize = templist.size();

    while(templist.size() > 1) {
        i=templist.indexOf("pow");
        // If pow is found in the list
        if (i != -1) {
            //if there's no bracket after pow then throw an exception
            if (templist.get(i+1)!="(") {
            throw new Exception("coefficient wrong");
        }
        else {
            int flag = 1;
            int start = i+1; //starting position of pow( , )
            int commaPosition = start; // the position of "," in  pow( , )
            int endingPosition = start; // the ending position  of pow( , )
            i++;
            while(flag!=0) {
                i++;
                if(templist.get(i)=="(") {
                    flag++;
                }
                if(templist.get(i)==")") {
                flag--;
                }
                if(templist.get(i)=="," && flag==1) {
                    commaPosition = i;
                }
            }
            endingPosition = i;
            LinkedList templist1=new LinkedList();
            LinkedList templist2=new LinkedList();
            for (int j = start+1; j < commaPosition; j++) {
                templist1.add(templist.get(j));
        }
        for (int j = commaPosition + 1; j < endingPosition; j++) {
            templist2.add(templist.get(j));
        }
        for (int j = 0; j < endingPosition - start + 2; j++) {
            templist.remove(start - 1);
        }
        templist.add(start                    -                    1,
        Double.toString(Math.pow(getValue(templist1),
        getValue(templist2))));
        continue;
    }
 }
// If "(" is found in the list then we add 1 to the flag variable, if ")" is found
then we
// subtract 1 from the flag variable
i=templist.indexOf("(");
if (i!=-1) {
    int  flag=1;
    int  start=i;
    int number=1;
    while(flag!=0) {
    i++;
    number++;
    if(templist.get(i) == "(") {
        flag++;
    }
    if(templist.get(i) == ")") {
        flag--;
    }
}
LinkedList templist1=new LinkedList();
for(int j = start + 1; j < i; j++) {
    templist1.add(templist.get(j));
}
for (int j = 0; j < number; j++) {
    templist.remove(start);
}
templist.add(start, Double.toString(getValue(templist1)));
continue;
}
// if sin is found then we calculate the value of sin(X),  pop both and
```

```
        // add the value to the linked list
        i=templist.indexOf("sin");
        if (i != -1) {
                opr1=Double.valueOf(templist.get(i                              +
1).toString().trim()).doubleValue();
                des=Math.sin(opr1);
                for(int j = 0; j < 2; j++) {
                        templist.remove(i);
                }
                templist.add(i,Double.toString(des));
                continue;
        }
        // if cos is found then we calculate the value of cos(X),  pop both and
        // add the value to the linked list
        i = templist.indexOf("cos");
        if (i != -1) {
            opr1            =              Double.valueOf(templist.get(i          +
1).toString().trim()).doubleValue();
                des = Math.cos(opr1);
                for(int j = 0; j < 2; j++) {
                        templist.remove(i);
                }
                templist.add(i,Double.toString(des));
                continue;
        }
        // if tan is found then we calculate the value of tan(X),  pop both and
        // add the value to the linked list
        i = templist.indexOf("tan");
        if (i != -1) {
                opr1            =              Double.valueOf(templist.get(i      +
        1).toString().trim()).doubleValue();
                des = Math.tan(opr1);
                for(int j = 0; j < 2; j++) {
                        templist.remove(i);
                }
                templist.add(i, Double.toString(des));
                continue;
        }
        // if cot is found then we calculate the value of cot(X),  pop both and
        // add the value to the linked list
        i = templist.indexOf("cot");
        if (i != -1) {
                opr1 = Double.valueOf(templist.get(i +
1).toString().trim()).doubleValue();
                des = Math.tan(opr1);
                if(des == 0) throw new Exception("overflow");
                des = 1.0/des;
                for(int j = 0; j < 2; j++) {
                templist.remove(i);
                 }
                 templist.add(i, Double.toString(des));
                 continue;
        }
        // if log is found then we calculate the value of sin(X), pop both and
        // add the value to the linked list
        i = templist.indexOf("log");
        if (i != -1) {
                opr1 = Double.valueOf(templist.get(i +
                1).toString().trim()).doubleValue();
                // the log(0) is - infinity so we can't have that so we throw an
                exception
                if(opr1 <= 0) throw new Exception("overflow");
                des = Math.log(opr1);
                for(int j = 0; j < 2; j++) {
                        templist.remove(i);
                }
                 templist.add(i, Double.toString(des));
                 continue;
        }
        // if "*" is found then we calculate the value of X * Y, pop both and
        // also pop the "*" sign and then add the value to the linked list
        i = templist.indexOf("*");
        if (i != -1) {
                 opr1=Double.valueOf(templist.get(i -
1).toString().trim()).doubleValue();
                 opr2=Double.valueOf(templist.get(i +
1).toString().trim()).doubleValue();
                 des=opr1 * opr2;
                 templist.remove(i - 1);
                 templist.remove(i - 1);
                 templist.remove(i - 1);
                 templist.add(i - 1, Double.toString(des));
                 continue;
        }
        // if "/" is found then we calculate the value of X / Y,  pop both and
        // also pop the "/" sign and then add the value to the linked list
        i = templist.indexOf("/");
        if (i != -1) {
                opr1              =              Double.valueOf(templist.get(i              -
1).toString().trim()).doubleValue();
                opr2              =              Double.valueOf(templist.get(i              +
1).toString().trim()).doubleValue();
                if(opr2 == 0) throw new Exception("overflow");
                des = opr1/opr2;
                templist.remove(i - 1);
                templist.remove(i - 1);
                templist.remove(i - 1);
                templist.add(i - 1, Double.toString(des));
                continue;
        }
        // if "+" is found then we calculate the value of X + Y,  pop both and
        // also pop the "+" sign and then add the value to the linked list
        i = templist.indexOf("+");
        if (i!=-1) {
                opr1=Double.valueOf(templist.get(i                                          -
1).toString().trim()).doubleValue();
                opr2=Double.valueOf(templist.get(i                                          +
1).toString().trim()).doubleValue();
                des=opr1 + opr2;
                templist.remove(i - 1);
                templist.remove(i - 1);
                templist.remove(i - 1);
                templist.add(i - 1,Double.toString(des));
                continue;
        }
        // if "-" is found then we calculate the value of X - Y,  pop both and
        // also pop the "-" sign and then add the value to the linked list
        i = templist.indexOf("-");
        if(i != -1) {
                if (i == 0) {
                        opr1=Double.valueOf(templist.get(i  +  1).toString().trim()).
                        doubleValue();
                        des= 0 - opr1;
                        templist.remove(i);
                        templist.remove(i);
                        templist.add(i, Double.toString(des));
        }
        else {
            opr1 = Double.valueOf(templist.get(i - 1).toString().trim()).
            doubleValue();
            opr2 = Double.valueOf(templist.get(i + 1).toString().trim()).
            doubleValue();
            des = opr1 - opr2;
            templist.remove(i - 1);
            templist.remove(i - 1);
            templist.remove(i - 1);
            templist.add(i - 1, Double.toString(des));
        }
        continue;
    }
    if(oldSize > templist.size()) {
        oldSize = templist.size();
    }
    else { // wrong coefficient since the size should decrease and not stay the
        //same or increase
        throw new Exception("coefficient wrong");
    }
}
return Double.valueOf(templist.getFirst().toString().trim()).doubleValue();
}
```

A-4: “varcoefodesolver.java”

This is an abstract class used to determine which of the three solvers is to be used. It has one common function (calculateHighestPrime) that is used by the three algorithms.

A-4.1: Important functions in “Varcoefodesolver.Java”

```
// An important method. common for all algorithms already built into the project.
// used to calculate the member of the theta family with the highest prime
protected double calculateHighestPrime(int row) throws Exception {
        int i = row;
        double highestPrime;
        if(i == 0) {
                highestPrime = 0;
        }
        else {
                highestPrime = depVar[i - 1][ord];
        }
        // if the coefficient @ time t of the highest order is not zero then the highest
        //prime would be equal to the (RHS - All other Coefficients @ time t)/Coef
        of highest order @ t
        if (coef[ord].getValueAtT(minIntTime + i * timeStep) != 0) {
                highestPrime=rhs.getValueAtT(minIntTime + i * timeStep);
                for (int j = 0; j < ord; j++) {
                        highestPrime = highestPrime -
                        coef[j].getValueAtT(minIntTime + i * timeStep) *
                        depVar[i][j];
                }
                highestPrime = highestPrime/coef[ord].getValueAtT(minIntTime + i
                * timeStep);
        }
        return highestPrime;
}
```

A-5: “varcoefodesolver1.java”

The “varcoefodesolver1” class implements the Euler algorithms. It is derived from varcoefodesolver described in Appendix A-4. The main function of this class is calculateHighestPrime which is used to calculate the values of the highest member of the theta family.

A-5.1: Important functions in “Varcoefodesolver1.Java”

```
//This method is to create the DepVar array - the logic of Euler algorithm is used
here.
//The calculateHighestPrime method used is a generic method derived from the
super class
//VarCoefODESolver and is used to get the value of the highest prime of theta. the
rest of
// theta family are calculated and saved in the array in this method:
calculateDepVar()
public void calculateDepVar() throws Exception {
        for(int i = 0; i < ord; i++) {
                depVar[0][i] = initValues[i];
        }
        depVar[0][ord] = calculateHighestPrime(0);
        for(int i = 1; i < calculationNumber; i++) {
                for( int j = 0; j < ord; j++) {
                        depVar[i][j] = depVar[i-1][j]+depVar[i-1][j+1]*timeStep;
                }
                depVar[i][ord] = calculateHighestPrime(i);
        }
}
```

A-6: “varcoefodesolver2.java”

The “varcoefodesolver2” class implements the Crank-Nicolson algorithms. It is derived from varcoefodesolver described in Appendix A-4. The main function of this class is also calculateHighestPrime that is used to calculate the values of the highest member of the theta family.

A-6.1: Important functions in “Varcoefodesolver2.Java”

```
public class VarCoefODESolver2 extends VarCoefODESolver {

//This method is to create the DepVar array - the logic of crank-nicolson algorithm
is used
//here. The calculateHighestPrime used is a generic method derived from the super
class
// VarCoefODESolver and is used to get the value of the highest prime of theta. the
rest of
// theta family are calculated and saved in the array in this method:
calculateDepVar()
public void calculateDepVar() throws Exception {
        for(int j = 0;j < ord; j++) {
                depVar[0][j] = initValues[j];
        }
        depVar[0][ord] = calculateHighestPrime(0);
        for(int j = 0; j < ord; j++){
                depVar[1][j] = depVar[0][j]+depVar[0][j+1]*timeStep;
        }
        depVar[1][ord] = calculateHighestPrime(1);
        for(int i = 2; i < calculationNumber; i++){
                for( int j = 0; j < ord; j++){
                        depVar[i][j]    =    depVar[i-1][j]+(1.5*depVar[i-1][j+1]-
                        0.5*depVar[i-2][j+1])*timeStep;
                }
        depVar[i][ord] = calculateHighestPrime(i);
        }
}
```

A-7: "varcoefodesolver3.java"

The "varcoefodesolver3" class implements the Predictor corrector algorithms. It is derived from varcoefodesolver described in Appendix A-4. The main function of this class is calculateHighestPrime that is used to calculate the values of the highest member of the theta family.

A-7.1: Important functions in "Varcoefodesolver3.Java"

```
//This method is to create the DepVar array - the logic of Predictor corrector
algorithm is used here.
//The calculateHighestPrime method used is a generic method derived from the
super class
//VarCoefODESolver and is used to get the value of the highest prime of theta. the
rest of //theta family are calculated and saved in the array depVar. What we do is
we get the predicted
//values of theta and then we uses these values to calculate the correct values and
then //resave them in the array
public void calculateDepVar() throws Exception {
        double prePara1 = 1.0;
        double[] corPara1 = {0.5, 0.5};
        double[] prePara2 = {1.5, -0.5};
        double[] corPara2 = {(5.0/12.0), (8.0/12.0), -(1.0/12.0)};
        double[] prePara3 = {(23.0/12.0), -(16.0/12.0), (5.0/12.0)};
        double[] corPara3 = {(9.0/24.0), (19.0/24.0),- (5.0/24.0), (1.0/24.0)};
        double[] prePara4 = {(55.0/24.0), -(59.0/24.0), (37.0/24.0), -(9.0/24.0)};
        double[]  corPara4  =  {(251.0/720.0),  (646.0/720.0),  -(264.0/720.0),
        (106.0/720.0), -(19.0/720.0)};
        //set initial value: Q0,Q'0,Q"0...
        for(int i = 0; i < ord; i++) {
                depVar[0][i] = initValues[i];
        }
        depVar[0][ord] = calculateHighestPrime(0);
        //predict Q1, Q'1, Q"1...
        for(int j = 0; j < ord; j++) {
                depVar[1][j] = depVar[0][j]+prePara1*depVar[0][j+1]*timeStep;
        }
        depVar[1][ord] = calculateHighestPrime(1);
```

```
//correct Q1,Q'1,Q"1...
for(int j = 0; j < ord; j++) {
        depVar[1][j]                                                    =
        depVar[0][j]+(corPara1[0]*depVar[1][j+1]+corPara1[1]
        *depVar[0][j+1])*timeStep;
}
depVar[1][ord] = calculateHighestPrime(1);
//predict Q2,Q'2,Q"2...
for( int j = 0; j < ord; j++) {
        depVar[2][j]                                                    =
        depVar[1][j]+(prePara2[0]*depVar[1][j+1]+prePara2[1]*
        depVar[0][j+1])*timeStep;
}
depVar[2][ord] = calculateHighestPrime(2);
//correct Q2,Q'2,Q"2...
for( int j = 0; j < ord; j++) {
        depVar[2][j]                                                    =
        depVar[1][j]+(corPara2[0]*depVar[2][j+1]+corPara2[1]*
        depVar[1][j+1]+corPara2[2]*depVar[0][j+1])*timeStep;
}
depVar[2][ord] = calculateHighestPrime(2);
//predict Q3,Q'3,Q"3...
for( int j=0;j<ord;j++) {
        depVar[3][j]                                                    =
        depVar[2][j]+(prePara3[0]*depVar[2][j+1]+prePara3[1]*
        depVar[1][j+1]+prePara3[2]*depVar[0][j+1])*timeStep;
}
depVar[3][ord] = calculateHighestPrime(3);
//correct Q3,Q'3,Q"3...
for( int j = 0;j < ord; j++) {
        depVar[3][j]                                                    =
        depVar[2][j]+(corPara3[0]*depVar[3][j+1]+corPara3[1]*
        depVar[2][j+1]+corPara3[2]*depVar[1][j+1]+corPara3[3]*depVar[
        0][j+1])*timeStep;
}
depVar[3][ord] = calculateHighestPrime(3);
for(int i = 4;i < calculationNumber; i++) {
//predict
for( int j = 0; j < ord; j++) {
        depVar[i][j]        =        depVar[i-1][j]+(prePara4[0]*depVar[i-
        1][j+1]+prePara4[1]        *depVar[i-2][j+1]+prePara4[2]*depVar[i-
        3][j+1]+prePara4[3]*depVar[i-4][j+1])*timeStep;
        }
        depVar[i][ord] = calculateHighestPrime(i);
        //correct
        for( int j = 0; j < ord; j++) {
                depVar[i][j]                    =                    depVar[i-
                1][j]+(corPara4[0]*depVar[i][j+1]+corPara4[1]          *depVar[i-
                1][j+1]+corPara4[2]*depVar[i-2][j+1]+corPara4[3]*depVar[i-
                3][j+1]+corPara4[4]*depVar[i-4][j+1])*timeStep;
        }
        depVar[i][ord] = calculateHighestPrime(i);
        }
}
```

APPENDIX B – DETAILED DESCRIPTION OF EPS CLASSES

Class Name	MainGui
Super Class	Jframe
Sub Class	N/A
Attributes	algo, order, rhs, algorithm, coef, dt, initV, mathExpression, maxt, mint, maxTime, minTime, ordersetting, RadioIndex, and other interface-related attributes (combo-boxes, text-fields…)
Methods	MainGui(), jbInit(), getAlgorithm(), processWindowEvent(), getArray2(), getInitV(), getOrder(), getTimeSetting(), jButtonAdd_actionPerformed(), jButtonCalculate_actionPerformed(), jButtonClear_actionPerformed(), jButtonUndo_actionPerformed(), jComboBox1_itemStateChanged(), jRadioButton0_focusGained(), jRadioButton1_focusGained(), jRadioButton2_focusGained(), jRadioButton3_focusGained(), jRadioButton4_focusGained(), jRadioButton5_focusGained(), jRadioButton6_focusGained(), jRadioButtonRhs_focusGained(),
Description	This is an interface class. It's main purpose is the graphical user interface, getting the input values from the user, calling the action methods according to the user request, and displaying the result graphs and statistics to the user.

Class Name	**Welcome**
Super Class	Jdialog
Sub Class	N/A
Attributes	None (Only interface-related attributes)
Methods	Welcome(), jbInit(), jButton1_actionPerformed(), jButton2_actionPerformed()
Description	This is also an interface class. Its main purpose is the graphical user interface welcome screen. No input is done on this screen.

Class Name	**Vcoeficient**
Super Class	LinkedList
Sub Class	N/A
Attributes	None
Methods	Vcoeficient() getTempList(), getString(), getValueAtT(), getValue()
Description	This class is used to calculate the variable function of t (time) of every coefficient in the equation. A linked list is used to contain the coefficient functions. Methods are performed to go through the list and calculate the final value at a specific time (t)

Class Name	VariCoefODE
Super Class	ODE
Sub Class	N/A
Attributes	VCoeficient, rhs
Methods	VariCoefODE(), getCoefficients(), getRHS(), setCoefficients(), setRHS()
Description	This class is mainly used to get and set the RHS (right hand side) of the equation as well as coefficient array

Class Name	VarCoefODESolver
Super Class	N/A
Sub Class	VarCoefODESolver1, VarCoefODESolver2, VarCoefODESolver3
Attributes	calculationNumber, maxIntTime, minIntTime, timeStep, initValues, depVar, ord, ccode, rhs, coef
Methods	VarCoefODESolver(), calculateDepVar(), getDepVar(), getInitValues(), getMaxIntTime(), getMinIntTime(), setInitValues(), setInitValues(), setMaxIntTime(), setMinIntTime(), calculateHighestPrime()
Description	This abstract class is used to solve the equation according to the

	algorithm selected. The abstract method calculateDepVar() is the one used to get the values of every instance of theta and family according to time and time-step. Since each algorithm has its own way of calculating the two dimensional array of theta & family values the actual code is in separate subclasses (VarCoefODESolver1, VarCoefODESolver2, VarCoefODESolver3)

Class Name	**VarCoefODESolver1**
Super Class	VARCOEFODESOLVER
Sub Class	N/A
Attributes	None
Methods	VarCoefODESolver1 (), calculateDepVar()
Description	This class is extended from the abstract class VarCoefODESolver.It is used to solve the equation according to the first algorithm (Euler). The only method that overrides the one in the base class is calculateDepVar() which is the one used to get the values of every instance of theta and family according to time and time-step.

Class Name	**VarCoefODESolver2**
Super Class	VARCOEFODESOLVER
Sub Class	N/A
Attributes	None
Methods	VarCoefODESolver2(), calculateDepVar()
Description	This class is extended from the abstract class VarCoefODESolver.It is used to solve the equation according to the first algorithm (Crank-Nicolson). The only method that overrides the one in the base class is calculateDepVar() which is the one used to get the values of every instance of theta and family according to time and time-step.

Class Name	**VarCoefODESolver3**
Super Class	VARCOEFODESOLVER
Sub Class	N/A
Attributes	None
Methods	VarCoefODESolver3(), calculateDepVar()
Description	This class is extended from the abstract class VarCoefODESolver.It is used to solve the equation according to the first algorithm (Predictor-Corrector). The only method that overrides the one in the base class is calculateDepVar() which is the one used to get the values of every instance of theta and family according to time and time-step.

Class Name	**dialogError**
Super Class	MAINGUI
Sub Class	Jdialog
Attributes	interface-related attributes
Methods	DialogError(), jbInit(), jButton1_actionPerformed()
Description	This class handles any error that occurs while execution

Class Name	**ODE**
Super Class	N/A
Sub Class	VariCoefODE
Attributes	Order
Methods	ODE(), getOrder(), setOrder()
Description	This abstract class is used to maintain the order of the equation that is to be solved

APPENDIX C – DETAILED DESCRIPTION OF SEQUENCE DIAGRAMS

The Input Data sequence diagram details (first part: excluding functions):

Sequence diagram name:	**Corresponding Use Case:**
Input Data (1)	**UC-1: Input Data**
Involved Objects:	**Description:**
: Welcome	Interface class (welcome screen). No interaction with the user.
: MainGui	Interface class which interacts between the user and the EPS system
Actions	**Description**
1: start	User start the application
2: start (2)	User goes to the main and only GUI used
3: getOrder()	EPS gets the order of the equation from the user
4: getInitV()	EPS gets the initial values of theta and family from the user
5: getTimeSetting()	EPS gets the start time, end time and the time step from the user
6: getAlgorithm()	EPS gets the user choice

The Input Data diagram details (second part: functions only):

Sequence diagram name:	**Corresponding Use Case:**
Input Data (2)	**UC-1: Input Data**
Involved Objects:	**Description:**
: Welcome	Interface class (welcome screen). No interaction with the user.
: MainGui	Interface class which interacts between the user and the EPS system
: VCoefficient	Entity class for the variable coefficients
Actions	**Description**
1: start	User start the application
2: start (2)	User goes to the main and only GUI used
3: jButtonAdd_actionPerformed()	User chooses to add a specific mathematical function or value to the function of a specific coefficient selected
4: jButtonUndo_actionPerformed()	User deletes the last mathematical function or value that was added to the function of a specific coefficient selected
5: jButtonClear_actionPerformed()	User deletes the whole function of a specific coefficient selected

The Start Calculation diagram details:

Sequence diagram name:	**Corresponding Use Case:**
Start Calculation	UC-3: Calculate Result
Involved Objects:	**Description:**
: Welcome	Interface class (welcome screen). No interaction with the user.
: MainGui	Interface class which interacts between the user and the EPS system
: dialogwaiting	Control class for the application error handling
:VCoefficient	Control class for calculating the array of linked lists (function) that is set for the equation
: VariCoefODE	Entity class that is used to set the coefficients (functions) as well as the right hand side of the equation
: VarCoefODESolver1, 2, or 3	Control class for calculating the theta and family two-dimensional array of instances by time. This array will be used to plot the

	diagrams. (derived from VarCoefODESolver)
: VarCoefODESolver	Abstract class that calculates the highest order of the theta family by time
: SimpleChart	A third party class (library) used for plotting simple diagrams
Actions	**Description**
1: start	User start the application
2: jButtonCalculate_actionPerformed()	After inputting all required data the user calls the calculation routine to get the result
3: getOrder()	Gets the order of the equation
4: getInitV()	Gets the initial values of theta and family
5: getTimeSetting()	Gets the time setting (min-time, max-time and time-step)
6: getAlgorithm()	Gets the algorithms selected by user
7: dialogWaiting(title)	Displays a in-progress execution window to alert the user that the process is in progress

8: VariCoefODE(order, coef, rhs)	Initializes the coefficients and sets the RHS and order for the solver to use
9: VarCoefODESolverX(ode,maxt,mint,dt,initV)	Note: X here is 1, 2, or 3. (depending on the algorithm selected) Initializes the solver to be used to solve the equation
10: calculateHighestPrime(i)	Calculates the highest member of the theta family
11: getValueAtT(coef[], time)	Gets the value of a specific function at a specific time
12: calculateDepVar()	Calculates the values of every instance of theta and it's family
13: setData(depVar)	Sets the resulting two dimensional array to plot the graph
14: setText(runTime)	Displays the time that was needed to execute the calculation
15: dispose()	Takes off the waiting window applied in action 7

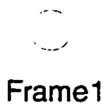
APPENDIX D – UML SYMBOLS USED IN THE RESEARCH DIAGRAMS

——◇ **Aggregation:** A special form of association that specifies a whole-part relationship between the aggregate (the whole) and a component (the part).

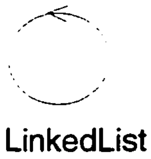
——— **Association:** A structural relationship that describes a set of links, in which a link is a connection among objects; the semantic relationship between two or more classifiers that involves the connections among their instances.

------> **Dependency:** A semantic relationship between two things in which a change to one thing (the independent thing) may affect the semantics of the other thing (the dependent thing).

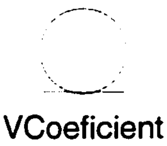
—▷ **Generalization:** A specialization/generalization relationship, in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent).



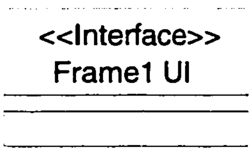
Icon form of Interface class



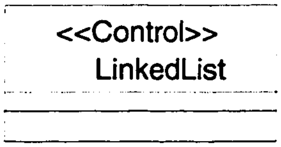
Icon form of Control class



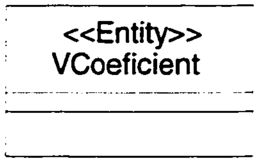
Icon form of Entity class



Label form of Interface class.

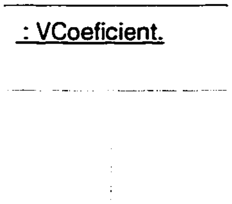


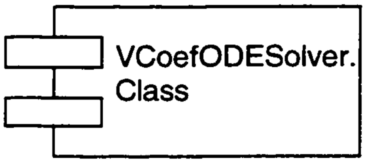
Label form of Control class



Label form of Entity class

UML Notation for Object instance in an Interaction Diagram The dashed line is the lifeline of the object, which represents the period during which the object exists.

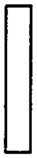




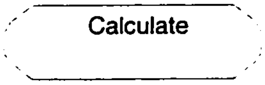
UML Notation for an Actor in an Interaction Diagram

UML Notation for message in an Interaction Diagram

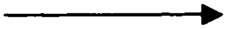
In a sequence diagram, the arrow goes from the sender to The receiver. The arrow is labeled with the message that is being sent.



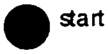
UML Notation for Focus of Control in an Interaction Diagram



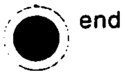
UML Notification for an activity in an activity diagram



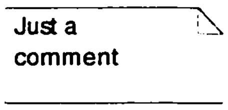
UML Notation for a transition between one activity and another in an activity diagram



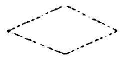
UML Notification for the start of an activity diagram



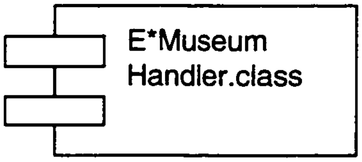
UML Notification for the end of an activity diagram



UML Notification for a comment in an activity diagram



UML Notification for a decision in an activity diagram. According to some results from previous activities a decision is to be made to decide on which activity should be triggered next.



UML Notation for a Component

BIBLIOGRAPHY

[1] Grady Booch, James Rumbaugh and Ivar Jacobson Doe. *The Unified Modeling Language User Guide.* Addison-Wesley, 1999.

[2] Terry Quatrani. *Visual Modeling with Rational Rose 2000 and UML.* Addison Wesley Longman, Inc, 2000.

[3] John Paul Fullerton. *Benefits of Object-Oriented Design.* 2002. Website: http://www.rtis.com/ nat/user/jfullerton/benefit.htm

[4] Wendy Boggs and Michael Boggs. *Mastering UML with Rational Rose.* SYBEX, 2000.

[5] CT Arrington. *Enterprise Java™ with UML.* John Wiley & Sons, Inc. 2001.

[6] Tom Rodden. *Introduction to software engineering.* Lancaster University, 1995. Website: http://www.comp.lancs.ac.uk/computing/users/tam/CS231/slides-1/sld001.htm

[7] Rota, Gian-Carlo;Birkhoff, Garrett. *Ordinary Differential Equations*, 1989.

[8] Patrick Hartling. Tweek, *The Programmer's Guide, 2002.* Website: http://www.vrjuggle.org/tweek/1.0/programmer.guide/programmer.guide/go01.html

[9] Roedy Green. *JNI, The Java Native Interface, 2002.* Website: http://mindprod.com /jni.html

[10] ComputerUser Inc, *High-Tech Dictionary, 2002.* Website: http://www.computeruser.com /resources/dictionary/

[11] Jason Carucci. *Java vs. C++.* Website: http://www.jasoncarucci.com/soapbox /javavsc. html

[12] Martin, Robert C. *Designing Object Oriented C++ Applications Using the Booth Method.* Prentice Hall, Inc. 1995.

[13] Sitraka Inc. *Sitraka JClass ServerChart.* Website: http://www.sitraka.com/software /jclass/jclassserverchart.html

[14] De La Cruz, J.M. De Lucas, P. Aranda, J. Giron-Sierra, J.M. Velasco, F. Maron. *A research on motion smoothing of fast ferries.* Website: http://www.dacya.ucm.es/giron /PDF/Gwoverw10.pdf

[15] Sun Microsystems. *Java Native Interface , 2002.* Website: http://java.sun.com/docs /books/tutorial/native1.1/

[16] Sitraka. Inc. 2002. Website: http://www.sitraka.com/

[17] Chart2D 2002. Website: http://www.sourceforge.net/

[18] JfreeChart 2002. Website: http://www.object-refinery.com/jfreechart/