
Software Requirements Specification

for

GALVIS

Version 1.0 approved

Prepared by Anzor Balkar

Cpt S 322

5/2/06

Table of Contents

Table of Contents	ii
Revision History	ii
1. Introduction	1
1.1 Purpose.....	1
1.2 Document Conventions and Definitions.....	1
1.3 Intended Audience and Reading Suggestions.....	2
1.4 Project Scope.....	2
1.5 References	3
2. Overall Description	3
2.1 Product Perspective	3
2.2 Product Features	3
2.3 User Classes and Characteristics.....	4
2.4 Operating Environment	4
2.5 Design and Implementation Constraints.....	5
2.6 User Documentation.....	5
2.7 Assumptions and Dependencies	5
3. System Features	6
3.1 XML Graph Generation.....	6
3.2 3D Graphics	10
3.3 Separation of Application and Algorithm	13
4. External Interface Requirements	15
4.1 User Interfaces.....	15
4.2 Hardware Interfaces.....	15
4.3 Software Interfaces.....	15
4.4 Communications Interfaces.....	17
5. Other Nonfunctional Requirements	17
5.1 Performance Requirements	17
5.2 Safety Requirements	17
5.3 Security Requirements	17
5.4 Software Quality Attributes	17
6. Other Requirements	18
Appendix A: Analysis Models	18

Revision History

Name	Date	Reason For Changes	Version
Anzor Balkar	4/30/06	Initial creation of document	0.1
Anzor Balkar	5/2/06	Completed first version	1.0

1. Introduction

1.1 Purpose

The purpose of this document is to provide feedback to the customer. This is the customer's assurance that the developers of this software understand the issues or problems to be solved and the software behavior necessary to address those problems. The document decomposes the problem into component parts and serves as the parent document to subsequent documents, such as the software design specification and statement of work.

This document states in precise and explicit language those functions and capabilities the GALVIS software must provide, as well as states any required constraints by which the system must abide. In essence, it functions as a blueprint for completing this project with as little cost growth as possible.

1.2 Document Conventions and Definitions

1.2.1 GALVIS

This is the final name of the release product. GALVIS stands for "Graph Algorithm Visualization Software."

1.2.2 ProjectDamascus

This is the code/working name of this project.

1.2.3 Algorithm

In mathematics and computing, an algorithm is a procedure (a finite set of well-defined instructions) for accomplishing some task which, given an initial state, will terminate in a defined end-state. The computational complexity and efficient implementation of the algorithm are important in computing, and this depends on suitable data structures.

1.2.4 Graph

A graph G is defined as follows: $G = (N, E)$, where N is a finite, non-empty set of nodes and E is a set of edges (links between pairs of vertices). When the edges in a

graph have no direction, the graph is called undirected, otherwise called directed. In practice, some information is associated with each node and edge.

1.2.5 Node

In graph theory, a graph describes a set of connections between objects. Each object is called a node or vertex.

1.2.6 Edge

In graph theory, a graph describes a set of connections between objects. Each object is called a node or vertex. The connections themselves are called edges or arcs.

1.2.7 Graph Piece

This will be referencing either a graph node or graph edge.

1.3 Intended Audience and Reading Suggestions

1.3.1 Developers

Developers may use this document to reference the necessary features and requirements of the GALVIS software that must be constructed and tested before release.

1.3.2 Project Managers/Customers

Project Managers/Customers may use this document to verify that all the necessary features they are seeking for this software are present and adequately described. These features include functional requirements, behavior between objects, and software behavior with the user, etc.

1.4 Project Scope

The purpose of GALVIS is to provide a mechanism for a prospective user to learn, improve, or reinforce their knowledge of various Graph Theory and Computer Science related graph algorithms. The software accomplishes this by enabling the user to visually iterate through their chosen algorithm while receiving real-time feedback from the system - ensuring them a successful traversal throughout the algorithm.

1.5 References

Microsoft .Net Framework:

<http://msdn.microsoft.com/netframework/gettingstarted/default.aspx>

All information required to learn about what the .NET framework is and what it provides.

OGRE: <http://ogre3d.org/>

The official website of the open-source 3D graphics library that GALVIS uses.

WIKIPEDIA: <http://wikipedia.org/>

Querying wikipedia for various graph terminology, Computer Science or mathematical terminology used in this document provides adequate results.

GraphML: <http://graphml.graphdrawing.org/>

The official site of the GraphML graph file format.

2. Overall Description

2.1 Product Perspective

This software exists specifically to aid undergraduate Computer Science students in their senior level Algorithms class or other related course work. Algorithms taught in such classes are usually complex and heavy in Computer Science or Mathematics jargon. GALVIS seeks to abstract all such discipline dependent terminology and visualize the algorithm to the user in the most easily understandable terms and graphics.

2.2 Product Features

GALVIS will allow a user to visually iterate through an algorithm. When the program turns on, the user will be prompted to select the algorithm of which they want to visualize, as well as the type of graph of which they want to visualize the algorithm with. After choosing these two options the user will then interactively iterate through the algorithm. After visualizing an algorithm for an undefined/variable number of times, the end effect is that the user will have a strong conceptual understanding of the algorithm and be able to apply the algorithm for any graph without having to reference the software.

2.3 User Classes and Characteristics

2.3.1 Undergraduate Computer Science Students (High Priority)

The main audience GALVIS is set to attract is undergraduate Computer Science students. Although the algorithm visualization abstracts the “heavy” discipline dependant terminology, there is still some level of terminology and understanding that the user must possess. Furthermore, the student is required to be interested in learning the algorithm. Because of the fact that the user input at each state of the algorithm is checked for correctness, a user may input incorrect values and not hinder the algorithms flow or final outcome. Such a feature may cause the student to become overly dependant on the system correcting their mistakes which may hinder their learning experience. However, only students who have a limited desire to learn the algorithm will experience this.

2.3.2 Intelligent Human Beings Curious in Graph Algorithms (Low Priority)

The software promotes algorithm visualization with immediate system feedback to the user. For this reason, any intelligent human being that has some intermediate level math and is curious in graph algorithms should at the minimum be able to navigate the user interface until reaching an end state for the algorithm that they chose. Having the user understand and retain the knowledge of the chosen algorithm is not required for this class of users as it was for the last class. They must merely be able to satisfy each step of the algorithm.

2.4 Operating Environment

2.4.1 Software Requirements

- Operating System: Microsoft Windows XP SP1 or newer
- Video: DirectX 9.0 or later
- Libraries: Microsoft .NET Framework 2.0

2.4.2 Hardware Requirements

- CPU: 1.0 GHz
- RAM: 256 MB
- Hard Drive: 20MB
- Video Card: ATI Radeon series 7000 or better, GeForce series

2.5 Design and Implementation Constraints

2.5.1 Software Constraints

- Must be completely written in C#
- Must use Microsoft .NET Framework 2.0
- Must use a third party UI control library for UI layout
- Must use a third party 3D graphics library for graphics
- Must use XML for all file operations
- All major user interface sections must exist in standalone class libraries

2.5.2 Hardware Constraints

- Rendering graphs with many nodes is video card intensive and may cause an inadequate frames per second slow down

2.6 User Documentation

2.6.1 README

GALVIS must be shipped with a README file that describes the basic functionality of the software, requisite skills needed to be an effective user, and graph creation.

2.6.2 Website/Online Help

Upon code complete of this software, a website must be created and provide a FAQ for usability. The questions this FAQ consists of may be derived from preliminary data gathering techniques such as questionnaire, and low/high fidelity prototype walkthroughs with test users.

2.7 Assumptions and Dependencies

2.8.1 Graphics library

The graphics library to be used is assumed to be seamlessly usable and callable from our native C# code environment.

2.8.2 Adequate Video Card Performance

It is assumed that the video card will be able to render the graphs that are passed to it from the XML files. Building huge graphs seems beyond the point of the graph algorithm visualizing software, therefore it is assumed that the reasonably sized graph

the video card will have to render will be done without slowing down the overall system.

3. System Features

The following is a list of features that are required in this version of GALVIS. Any relevant Likert scales between 1 and 9 to follow deem 1 as low and 9 as high.

3.1 XML Graph Generation

3.1.1 Description and Priority

Graphs should be generated dynamically by parsing and reading information from XML files. Maintaining a certain level of “dynamic” feeling when users visualize algorithms is imperative. One way to accomplish this is by having many different graphs. This will make it apparent to the user that the algorithm they are visualizing is general, and may be applied to any qualifying graph. Often, students will learn an algorithm from some Java Applet visualization in which there is only one graph. When the student tries to apply the algorithm to any other graph, they realize that they merely memorized a series of ways to solve one particular graph - not learn the algorithm conceptually.

XML Graph Generation

Priority: 9

Benefit: 9

Penalty: 7

Cost: 7

Risk: 5

3.1.2 Stimulus/Response Sequences

Upon turning on the program and selecting an algorithm from the list of available algorithms, the system will populate the list of available graph categories, with each algorithm selection having the potential to populate its own list of available graph categories.

3.1.3 Functional Requirements

REQ-1: CUSTOM GRAPH XML FORMAT

Description:

Conforming to, and fully implementing support for existing graph xml schemas such as GraphML is not required in this version of the software. However, this may be necessary in succeeding versions. As such we will create our own graph xml file format.

Inputs:

The information in the XML file may be inputted 'by hand' by any person that has a basic understanding of 3D coordinates.

Outputs:

The XML file must be saved with an ".xml" file extension with the text preceding that being completely arbitrary. The file must be saved in the "media/graphs" folder.

Functionality:

Our custom XML file format will possess the following pattern.

```
<graph category="Small" for_algorithm="Dijkstra's Algorithm">
  <node x="10" y="0" z="120">A</node>
  <node x="66" y="0" z="100">B</node>
  <node x="-123" y="0" z="70">C</node>

  <edge weight="21" directed="true">AB</edge>
  <edge weight="94" directed="true">BC</edge>
  <edge weight="123" directed="true">CA</edge>
</graph>
```

Nodes are defined with the X, Y, Z positions, as well as a one character letter that represents the name of the node. Edges may define their weight, and also a true or false on whether they are directed. Also, their names represent the nodes in with which they are incident, with the first name being the source node and the latter being the destination node. The number of nodes and the number of edges is arbitrary and may be added as desired.

Performance:

Performance is not an issue in this requirement, as graph XML files will never be unreasonably large, and parsing XML files is quick.

Error Handling:

Any graph XML file which does not follow this pattern strictly will be ignored. Upon reading an erroneous file, the system will disregard its existence and the user will never see it.

Design Constraints:

Some algorithms do not require the use of defining whether or not an edge is directed, etc. However, because we are seeking to make a format that is most usable, we will implement such functionality. After completing this file format and comparing it to the wholly designed GraphML schema, they are very similar. The major difference is that GraphML supports things such as graphs within graphs – features that far beyond the scope and purpose of GALVIS.

REQ-2: GRAPH CATEGORIZATION

Description:

Graphs may be categorized based on the category name that they define, whether or not they define that they are only meant for a certain algorithm, and whether or not a selected algorithm defines that it will only use graphs specific graphs.

Inputs:

The “category” and “for_algorithm” attributes used in the “graph” tag in the XML file will determine the way in which graphs are categorized.

Outputs:

When the user selects an algorithm, the correct and corresponding graph categories will be populated in the available graphs listbox.

Functionality:

Every graph will belong to a category and the name of this category will be specified inside the xml file in the designated “category” attribute in the “graph” tag. Leaving this attribute empty (“”) will cause this graph to be ignored. Furthermore, a graph can specify to

which algorithm it belongs. Leaving this attribute empty signifies that this is a “general” graph and that any algorithm that accepts “general” graphs may use it. This exists because it may be the case that certain algorithms can only work on certain types of graphs. For this reason, an algorithm may define a flag that states whether to only look for graphs whose string in the for_algorithm attribute is equal to the name of the algorithm. Hence, when an algorithm is clicked on in the available algorithms listbox, the system will first look for graphs that define themselves for that algorithm, and then add the name of their category to the list of available graphs, if it doesn’t already exist there. If the algorithm allows all graphs, the system will also add graphs whose for_algorithm attribute are “”. Remember that multiple graphs can be defined with the same category. This is so that when the user selects the graph category, a random graph which defines that category will be selected and used. Thus the effect of dynamic graph sense for the user is realized.

Performance:

Performance is not an issue in this requirement.

Error Handling:

Any graph XML file which does not follow our specified format will be ignored before it even gets to this point.

Design Constraints:

Our goal was to give the user the ability to choose a graph on which to visualize their chosen algorithm. Although it is also true that not every algorithm can run on every graph. For example, an algorithm like Prim’s Algorithm to calculate a minimum spanning tree of a graph can be used on any graph, however an algorithm like Dijkstra’s algorithm only works on graphs with non-negative edge weights. Meanwhile it is not necessarily the responsibility of the graph creators to manage this predicament. For this reason we had to create some type of mechanism to ensure that the algorithm visualization only happened on graphs which do not contradict the algorithms assumptions about the graph it is traversing.

3.2 3D Graphics

3.2.1 Description and Priority

Marshall McLuhan once wrote: "the medium is the message." And when learning complex algorithms, the simple aspects of the medium begin to hinder the learning process more and more. A way to deal with this is to heighten the sense of interactivity between the user and teacher - in this case GALVIS. Thus, a major objective of this software is to create an interactive environment for the user. One of the ways to do this is to render the graph in three dimensions. This will raise the users' entertainment level and thus increase their interest level. It will also have the user feel like a graph is a tangible object, not just an abstract data structure.

3D Graphics

Priority: 9

Benefit: 9

Penalty: 6

Cost: 8

Risk: 8

3.2.2 Stimulus/Response Sequences

Upon selecting an algorithm and graph of which to visualize, the system will show a sequence of the graph creation, complete with cinematic type camera controls and special effects.

3.2.3 Functional Requirements

REQ-3: 3D GRAPHICS LIBRARY INTEGRATION

Description:

A 3D graphics library must be chosen and integrated into the software.

Inputs:

The DLL's or References of the graphics library that will be used.

Outputs:

An application that is now rendering a 3D world and scene to a user interface control such as a .NET Panel.

Functionality:

The graphics library will successfully initialize upon initialization of the software and successfully destroy itself upon exit of the software. After choosing the algorithm and graph of which to visualize, the program will enter a visualization state. It is here that the 3D scene will be updated and rendered upon every cycle of the application. The entities within the scene such as nodes and edges may be manipulated at runtime at specific times that the software designer defines (such as graph creation). The camera controls within the world may be manipulated at runtime via a user interface that a user experience engineer and software engineer design.

Performance:

None for integrating the engine however view the next requirement for more in depth issues.

Error Handling:

If the graphics library fails to initialize correctly, the application should show an error dialog to the user and graciously exit.

Design Constraints:

Finding a 3D graphics library with a native C# interface will be difficult. Remember that the graphics library must out-of-the-box support meshes, mesh animation, camera animation, particle effects, octree geometry occlusion, and DirectX 9.0 or better fixed function lighting. Also, the size of the 3D window should be big enough for the graph to be seen clearly yet small enough for other relevant user interface controls to be present on the screen.

REQ-4: 3D GRAPH REPRESENTATION**Description:**

We have established that the information concerning the graph is read and parsed from an XML file. But what are we to do after these data structures have been populated in memory? We must now represent each one of these as physical objects in the 3D world.

Inputs:

The list of graph nodes within the selected graph – each node complete with all relevant information that was included in the XML file. The list of graph edges within the selected graph – each edge complete with all relevant information that was included in the XML file.

Outputs:

There will be physical representation of each graph piece in the 3D world.

Functionality:

After populating the list of all the graph pieces in the selected graph, the system will proceed to create meshes for each of these graph pieces. For the nodes, the meshes should somehow represent the concept of a node. As for the edges, cylinders with length should be placed exactly in 3D space between the two nodes in which it connects. It should then be dynamically rotated such that its local x-axis is parallel to the vector difference between the positions of the graph nodes it connects. After the rotation, the mesh should be mathematically scaled on its x-axis at a ratio that will physically connect the two nodes together with the edge.

Performance:

The number of nodes and edges (thereby geometry in the 3D world) that can be added to the scene is dependant upon the information in the graph XML file. The number of nodes and edges that an XML file can have in it is nearly infinite, or at least unreasonably large. Obviously the amount of video card resources is not infinite, and it's inevitably possible to define a graph in the XML that will bring the video card to its knees at runtime. However, the graphics library should at a minimum be able to render at least 50 nodes and edges and ~75 polygons a piece and maintain a frames per second rate greater than 30.

Error Handling:

Data structures for the graph pieces are already guaranteed to have proper information before reaching this point.

Design Constraints:

The meshes for the graph nodes should not exceed 75 polygons per mesh. The meshes for the edges must be constructed such that when scaled, there is negative effect on the meshes representation and should not exceed 30 polygons per mesh.

3.3 Separation of Application and Algorithm

3.3.1 Description and Priority

Algorithms should be able to be developed, compiled, and built outside of the application. The role of the application is to provide an interface for the graph, as well as some user interface space for the algorithm to use. The inner workings and control flow of the algorithm should be completely hidden from the application, and will be. This is required because adding new algorithms to the software or changing existing ones has absolutely nothing to do with the application itself, and therefore not require a recompilation of the application project.

Separation of Application and Algorithm

Priority: 9

Benefit: 9

Penalty: 9

Cost: 9

Risk: 5

3.3.2 Stimulus/Response Sequences

Upon turning the program, the application will query the directory for algorithm class libraries and show the names of them in the available algorithms listbox. Upon iteration through the algorithm, every time the "Next" button has been clicked, indicating to iterate to the next step of the algorithm, such a message will be sent to the algorithm instructing it to proceed to the next step that it has defined.

3.3.3 Functional Requirements

REQ-5: AN ABSTRACT ALGORITHM BASE CLASS

Description:

An abstract base class for the algorithm must be constructed. This will “pure virtually” declare all the necessary interfaces that need to be present for application and algorithm interaction.

Inputs:

This will input the DLL’s or References to the “Graph” and “Graphics” libraries.

Outputs:

This will output a DLL for which child algorithms can inherit and define their own functionality independently.

Functionality:

The abstract algorithm base class defines the necessary interfaces between the application and the algorithm. These interfaces include, among other things, telling the algorithm to move to its next state and populating it, and also the application copying over these controls from the algorithm itself. The application must also have a way to query the algorithm and see if it is done with its traversal.

Performance:

There are no performance issues with this requirement.

Error Handling:

When loading individual algorithms from their respective DLL’s, the application must ensure a class exists within that assembly which is of the derived from the abstract algorithm base class. Also, any methods that the application will call on the algorithm must be pure virtually declared in the abstract algorithm base class, ensuring that it is defined in all the children algorithms.

Design Constraints:

Providing too many interfaces for this abstract algorithm base class will provide more confusion to the algorithm developers. The interface must be minimal as possible – including the most important aspects such as sending the “go to next state” message and user interface control copying between the two.

4. External Interface Requirements

4.1 User Interfaces

4.1.1 Choosing Algorithm State

In this state, there must be two listboxes. One listbox will be to hold the available algorithms that the user may select to visualize. The second listbox will contain the available graph categories that the user will be able to select to visualize their chosen algorithm with. Finally, an "Exit" button and a "Launch" must be present for the user to proceed to the next state of their choosing.

4.1.2 Visualization State

The user interface of this state will be divided into four sections. The major section in the top left corner will be the 3D rendering of the chosen graph. In the top right corner will be a small section for camera control for the 3D world. Running along the bottom will be the "Navigation" window. This is where each state of the algorithm will be copied to. Additionally, running down the right side, and under the camera control section, will be a "Workspace." Each algorithm has a separate control called a "Workspace" that they may use to aid the user in remembering or storing information. At startup, this workspace will be copied from the algorithm DLL into the applications EXE into the "Workspace" window that we have allotted for it.

4.2 Hardware Interfaces

The major hardware interface between the software and any hardware device is the video card. However this interface is controlled by the graphics engine that we're using. We will not have to write code specifying this interface in the software, therefore it is irrelevant to write about it here.

4.3 Software Interfaces

4.3.1 Microsoft .NET Framework 2.0

All the user interface controls that GALVIS uses are derived from this framework. The XML and dynamic DLL loading this framework supports are also taken advantage of in this software..

The .NET Framework is a development and execution environment that allows different programming languages & libraries to work together seamlessly to create Windows-based applications that are easier to build, manage, deploy, and integrate with other networked systems.

The .NET Framework consists of:

The Common Language Runtime (CLR)

A language-neutral development & execution environment that provides services to help "manage" application execution

The Framework Class Libraries (FCL)

A consistent, object-oriented library of prepackaged functionality

The .NET Framework provides the basic infrastructure that Windows-based applications need to make Microsoft's .NET vision of connecting information, people, systems, and devices a reality:

Support for standard networking protocols & specifications

The .NET Framework uses standard Internet protocols and specifications like TCP/IP, SOAP, XML, & HTTP to allow a broad range of information, people, systems, and devices to be connected

Support for different programming languages

The .NET Framework supports a variety of different programming languages so developers can pick the language of their choice

Support for programming libraries developed in different languages

The .NET Framework provides a consistent programming model for using prepackaged units of functionality (libraries) which makes application development faster, easier & cheaper

Support for different platforms

The .NET Framework is available for a variety of Windows platforms, which allows people, systems, and devices to be connected using different computing platforms. E.g. People using desktop platforms like Windows XP or device platforms like Windows CE can connect to server systems using Windows Server 2003.

4.3.2 OGRE

GALVIS will use OGRE as its 3D graphics rendering engine.

OGRE (Object-Oriented Graphics Rendering Engine) is a scene-oriented, flexible 3D engine written in C++ designed to make it easier and more intuitive for developers to produce applications utilizing hardware-accelerated 3D graphics. The class library abstracts all the details of using the underlying system libraries like Direct3D and OpenGL and provides an interface based on world objects and other intuitive classes.

4.4 Communications Interfaces

None.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

All relevant performance requirements for the 3D graphics engine have been discussed in REQ-4.

5.2 Safety Requirements

None.

5.3 Security Requirements

None.

5.4 Software Quality Attributes

The developers of the algorithms must make sure that the technique that they are using to teach the algorithm is actually working. This can be verified by low/high fidelity usability tests.

All additions to the interface of the abstract algorithm base class must not break functionality within any of the derived algorithm classes.

6. Other Requirements

This software must be free to use and open-source. Setting up project on SourceForge.net would be the optimal solution for this.

Appendix A: Analysis Models