DESIGN AND IMPLEMENTATION OF AN INTERACTIVE INTERFACE FOR POWER PLANT SIMULATION

Zygmunt L. Szpak and Jules R. Tapamo
School of Computer Science
University of KwaZulu-Natal
Durban, 4062, Republic of South Africa
Email: zygmunt.szpak@cs.ukzn.ac.za and tapamoj@ukzn.ac.za

Joe Roy-Aikins
School of Mechanical Engineering
University of KwaZulu-Natal
Durban, 4041, Republic of South Africa
Email: aikinsj@ukzn.ac.za

KEYWORDS

Computer Aided Design, Graphical Model Builder, Brayton and Rankine Cycle, Powerplant Simulation

ABSTRACT

A large number of useful software was written at a time when graphical user interfaces were not the norm. By current standards these programs are often dismissed as being difficult to use and are ignored. *Brakine* is one such program. Brakine is a powerful design and simulation system of arbitrary energy systems that operate on the thermodynamic principles of the Brayton and Rankine cycles. It suffers from a very tedious user interface and has fallen largely out of use. This project creates a new visual user interface for Brakine that facilitates the design of the simulation by providing functionality to draw the entire simulation on screen. The tedious task of generating the text input file and reading the text output file is taken away from the user. The graphical user interface generates the correct input file for the simulation by taking into account the inter-connectivity of the components that have been drawn on the screen and the simulation parameters that have been assigned to these components. The results of the simulation are displayed by labelling the connections between components. This aids in the overall understanding and interpretation of the simulation.

1. INTRODUCTION

One of the greatest contributing factors for the failure of software in the marketplace, is a badly designed user-interface (Hobart, 2001). If end-users feel that the learning curve of the software is too steep they will quickly abandon the product in search of something that is more user-friendly. Many valuable programs are never utilized to their full potential because of steep learning curves. The reason why such a large number of programs end up unusable is because their user-interfaces were implemented as an after-thought, instead of being integrated tightly into the development cycle.

With the introduction of Windows as a world-wide operating system, end-users have come to expect a user-interface that is visually driven. Programs that operate in text-mode are judged to be archaic and are quickly dismissed. Unfortunately, there are many useful and relevant programs that were programmed when a text-mode user-interface was still the norm. A new visually driven interface must be designed for these types of programs before they fall completely out of use.

The aim of our work is to design and implement an interactive visual user-interface for *Brakine* (Roy-Aikins, 1995). Brakine is a versatile design and simulation system of arbitrary energy systems that operate on the thermodynamic principles of the Brayton and Rankine cycles (Cengel and Boles, 2006). It is text-mode driven and suffers from a very tedious user-interface. As such its use is severely limited and it has never been introduced into the classroom. The new visual user-interface will resolve some of the biggest obstacles a user experiences when working with the Brakine simulation and will facilitate the drawing of engineering diagrams so that the entire simulation, including the results, can be displayed on screen in an interactive manner.

The rest of this paper is organized as follows. We start by presenting an overview of the Brakine software and its limitations. Section 3 presents related work and in section 4 we delve into the architecture of our solution. In section 5 we provide a description of JGraph. Section 6 looks at some key functionalities of our software and 7 examines the results and limitations. Finally section 8 concludes our work and provides suggestions for future work.

2. Background

The Brakine simulation program was written in Fortran and designed at a time when graphical user interfaces were not the norm. As such it is text-mode driven and relies on the user to type an input file that specifies information for the simulation. The results of the simulation are then written to an output file.

It is the generation of the input file that poses the biggest problem. Generating the input file is a daunting task because the input file has to adhere to a very strict structure and the amount of information that the user has to specify can also be overwhelming. The entire design of the system including the inter-connectivity between components, all simulation parameters and other information such as scheduled data have to be typed out. This

Proceedings 23rd European Conference on Modelling and Simulation ©ECMS Javier Otamendi, Andrzej Bargiela, José Luis Montes, Luis Miguel Doncel Pedrera (Editors)
ISBN: 978-0-9553018-8-9 / ISBN: 978-0-9553018-9-6 (CD)

is a very tedious and time consuming process. It is also a very error-prone process.

Before the input file can be created the user draws a mechanical engineering diagram of the system that will be simulated. This has to be done on paper because there is no functionality in the simulation program to allow the user to draw diagrams. If the drawing is accidentally lost or misplaced it becomes very difficult to visualize the design of the system, because the input file by itself is rather obscure.

The simulation software was originally designed as a teaching tool for mechanical engineering students because commercial software was too expensive. However, students could simply not overcome the very steep learning curve of this tool. For these reasons the software has fallen largely out of use.

In this paper we present the design and the implementation of an interactive interface that will allow the user to draw the mechanical engineering diagram on the computer. The input file for the simulation program will be generated automatically based on the diagram that the user has drawn and the properties the user has assigned to the mechanical engineering components via the graphical user interface. This will remove the burden of typing the input file from the user. Furthermore the interactive interface will display the results of the simulation by labelling the connections between components so that the user can visualize the outcome of the simulation. By placing great emphasis on making the interface user-friendly the Brakine simulation software can be brought back into use.

3. Related Work

Several computer aided design tools for power plant simulation have already been reported in the literature. One of these initiatives is called Modular Modeling Systems Model Builder (MMSMB) (McKim and Matthews, 1996). With this program a user can quickly build a simulation by selecting from a library of pre-configured power plant components to dynamically simulate their operation. Simulation code is automatically generated by taking the connectivity and parameters of different components into account. Another power plant simulation tool that was developed recently is called PowerSim (Kim et al., 2005). It is targeted at the Korean market and aims to be a more flexible power plant design tool than MMSMB by allowing the user to integrate new custom-made components into the power plant component library. The modeling tool Modelica has also been used as a platform for power plant simulation, however the work of (Casella and Leva, 2003) is incomplete as many components such as compressor, turbine or combustion chambers still need to be developed. The fundamental difference between our work and related work is that their scientific visualization of the simulation was planned and integrated into the design of the program from the start of the development cycle. In our case, we are integrating a visualization tool with a simulation program that was not designed for one, and report on the issues that arise in such an endeavor. One of the first problems is identifying the best tools that will facilitate the intergration of the visualisation functionality.

Generating the correct input file for the simulation based on the diagram the user has drawn and the properties the user has assigned requires knowledge of the structure that the input file must adhere to and is unique to this project. However, a lot of previous work has been done in generating an interactive user-interface that facilitates the drawing of diagrams.

There are numerous *Java* libraries that assist in the drawing of diagrams, each with their own advantages and disadvantages. The leading commercial libraries are *yFiles* (yWorks, 2009) and *Tom Sawyer Software* (Tom Sawyer Software, 2009). They provide numerous layout algorithms as well as a plethora of other features. They are however expensive and the source code is not available meaning that they cannot be customized.

In terms of open-source software libraries there are also a lot to choose from. The *Java Universal Network/Graph Framework* (O'Madahain et al., 2003) is an open-source software library designed to support the modelling, analysis, and visualization of data that can be represented as graphs. Its emphasis is on mathematical and algorithmic graph applications pertaining to the fields of social network analysis, information visualization, knowledge discovery and data mining. It is ideal for fast rendering of very large graphs but it is not suitable for this project because even though the mechanical engineering diagram that the user draws is essentially a graph, a lot more detail and control is needed for the design and layout of the diagram than the library provides.

Another very recent Java library that is rapidly gaining popularity is *Prefuse* (Pendleton et al., 2007). Prefuse is a Java-based toolkit for building interactive information visualization applications. It provides optimized data structures for tables, graphs, and trees, a host of layout and visual encoding techniques, and support for animation, dynamic queries, integrated search, and database connectivity. Once again, it is targeted at very large graphs and some of the features like optimized data structures are not required for this project. Hence is also not suitable.

There are many other Java graph drawing libraries that are available but they either have very poor documentation or are no longer being developed.

A notable exception to this is *JGraph* (Alder and Benson, 2001). JGraph is a flexible *Swing* graph visualization and layout library. JGraph is specifically orientated towards design layout, like the popular *Microsoft Visio* (Microsoft, 2007). It is probably the most well known open-source graph visualization library currently available and has a very vibrant user community. It is actively maintained and comes with an excellent free user manual and numerous example programs. The community forum is also very active and posts are made on a daily basis.

JGraph has been used for numerous applications that

are related to this project in terms of interface design.

Aigner and Mikisch (Aigner and Miksch, 2004) made use of the JGrpah library to develop an interactive visualization method for computer supported protocol-based care in medicine.

JGraph was also used by (Hanish et al., 2004) to create a tool for interactively exploring the meaning of expression experiments in context of biological networks.

Valyi (Valyi and Ortega, 2004) used JGraph to create an open source simulation platform dedicated to systems ecology and emergy studies called *Emergy*. Emergy is an efficient graph sketcher that allows users to draw an energy diagram while providing an intuitive drag and drop interface, high standard graphical features and an extensible interactive and multilingual help system.

Reyes (Reyes, 2004) created a prototype *StreamIT Graph Editor* using the JGraph library. *StreamIT* is a language for the development of streaming applications that has a hierarchial and structural nature that lends itself to a graphical programming tool. The StreamIT Graph Editor provides intuitive visualisation tools that allows developers to work more efficiently by automating certain processes.

Commercial companies have also used JGraph in their applications.

Clearly the JGraph library is well suited to design a very user-friendly and flexible interactive interface. For this reason it was the library of choice for this project.

To our knowledge there is currently no open-source project that can be used to simulate arbitrary energy systems based on the Rankine and Brayton cycle, like the Brakine program can. For this reason no attempt has been made to design a graphical user interface for such an application as an open-source community project. There are however commercial applications that perform a very similar function to what this project aims to achieve. The company that has been at the forefront of designing thermal engineering software for the power and cogeneration industries since 1987 is *Thermoflow Inc* (Thermoflow, 2009). Thermoflow's first product, *GT PRO*, has grown to become the world's most popular program for designing gas turbine-based plants. However it is another one of their products, *Thermoflex*, that is of particular interest to this project. Thermoflex is a modular program with a graphical interface that allows for the assembly of a model from icons representing over one hundred different components. The program covers both design and off-design simulation, and models all types of power plants, including combined cycles, conventional steam cycles, and repowering. It can also model general thermal power systems and networks.

Initially Brakine was written as an alternative to the products that *Thermoflow* provides. The ultimate goal was to provide similar flexibility in design that Thermoflex currently offers. The interactive interface that Thermoflow provides in many ways is a benchmark for this project (see Fig 1).

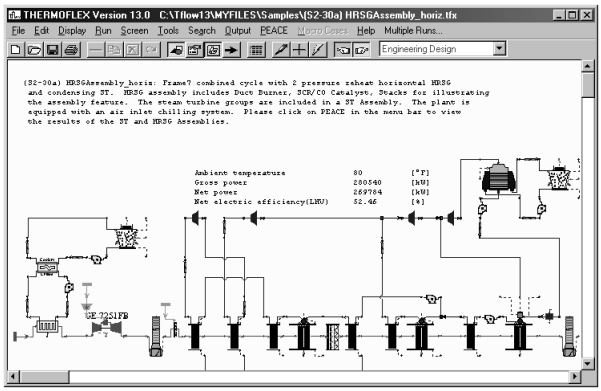


Figure 1: Screenshot of Thermoflex Software

4. ARCHITECTURE OVERVIEW

To understand how the graphical user interface that we created interacts with the Fortran simulation program, it is best to look at the flow of data between these two programs (see Fig 2).

The graphical user interface generates an input file for the Fortran simulation based on the components that have been dragged into the document, the connectivity between the components and the simulation data that the user has assigned to the components. Once the input file has been generated the user runs the Fortran simulation program using the generated input file as the input file for the simulation. The Fortran simulation program produces an output file containing the results of the simulation. This output file is parsed via the graphical user-interface and the results of the simulation are displayed by labelling the edges between the different components on the screen.

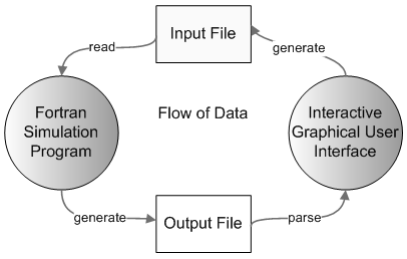


Figure 2: Flow of Data

In this way the difficult and tedious task of creating the input file is taken away from the user. Furthermore, by labelling the edges between components with the correct simulation results the user can better understand the outcome of the simulation.

The Brakine simulation software expects the input file to follow a very strict order and structure. The breakdown of the structure can be found in (Roy-Aikins, 1988). A snippet of data for a text input file for a jet engine simulation is given in Table 1. This kind of input file is now automatically generated by our graphical user

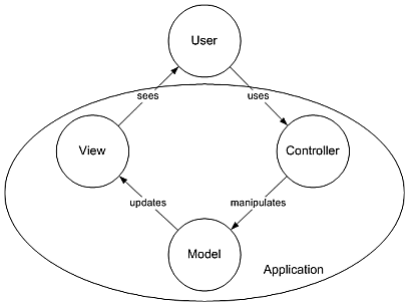


Figure 4: Model View Controller

interface. Most of the information that is required by the input file is extracted from the components on the screen, by considering how a user connects components together, and by capturing simulation data via input dialogues.

To display the results on screen and to label the connections between components with the relevant results the output file of the simulation program is parsed. What is of particular interest to the user is the mass flow ratio, the total pressure and the total temperature between two components. Regular expressions (Friedl, 2006) are used to extract this information from the output file. A snippet of an output file that is generated by the original Fortran program is presented in Table 2. The visual interpretation of these results generated by our program is presented in Fig 3.

5. JGRAPH ARCHITECTURE

JGraph complies with all *Swing* standards, both visually and in its design architecture (Benson, 2001). As such, it is built around the Model-View-Controller (MVC) design pattern (see Fig 4).

The Model-View-Controller pattern is used when the user interface needs to be separated from the functional core of the program (Buschmann et al., 2001). By separating the user interface from the core functionality, several different views of the model can be displayed. For example, in JGraph the logical graph that is specified in the model can be displayed in various different ways at different levels of detail simultaneously.

The Model-View-Controller architecture divides a problem into three areas: processing, output and input. This means that the model contains the core functionality and data, the view displays information to the user and the controller handles user input. The important concept in the MVC pattern is that the model is totally independent of specific output representations or input information(Buschmann et al., 2001). The view obtains the data that it wishes to display from the model and multiple views of the model can exist at any point in time. The controller receives input events, usually mouse or keyboard, that are then translated into service requests for the model and view. In this way the view and controller work together to make up the user interface.

Table 1: Code Snippet of a Sample Input File to Simulate a Jet Engine

```
PROGRAM TITLE////
DP SI KE FP
-1
-1
-1
2
1
7 2
3 7
.6 42.5 .7 42.5 .8 42.5 .85 28.5 .9
14.5 .95 .5 .97 -5.  1.
3
1
7 2
3 5
.6 .13 .7 .115 .8 .1 .8 .05 .8 .05 .9
.05 1.  .05 1.
-1
INTAKE S1,2 D1-5,78,278-279,344
COMPRE S2,3 D6-17 R51 V1,2,6 A86-106
PREMAS S3,10,3 D18-23,268 A107-127
PREMAS S10,11,12 D80-85,269 A128-148
BURNER S3,4 D24-27 R69
MIXEES S4,11,5 D317
TURBIN S5,7 D43-57,341,343 V2,2,44
W3,2,43
DUCTER S7,8 D58-62,149
NOZCON S8,9,1 D63,79
CODEND
T64-GE-415 DATA///
!GAS TURBINE CYCLE
!INTAKE
1 0
2 0
3 0
4 .99
5 -1
78 -1
!COMPRE
6 .85
7 1
8 12.5
9 .82
10 -1
11 1
13 -1
8 12.5
14 .9
15 -1
16 1
17 -1 ...
```

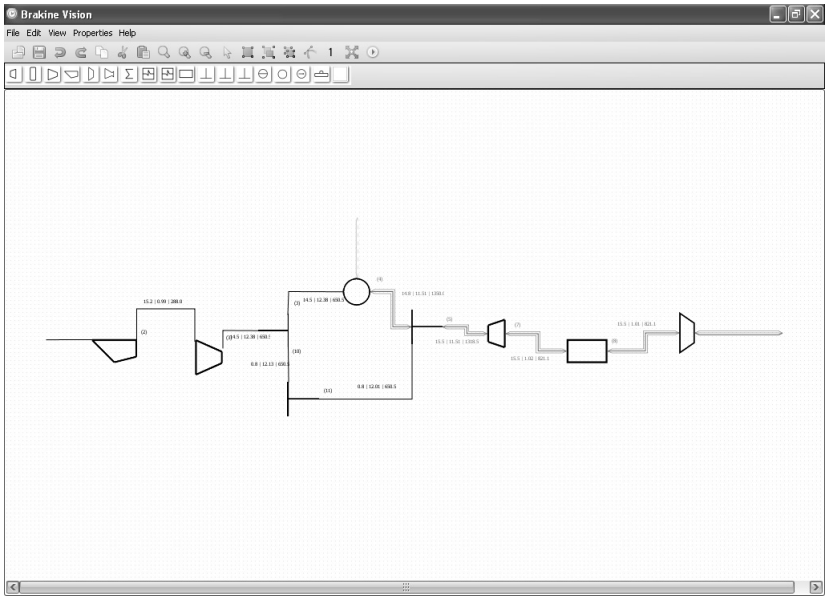
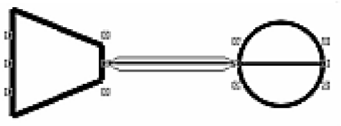
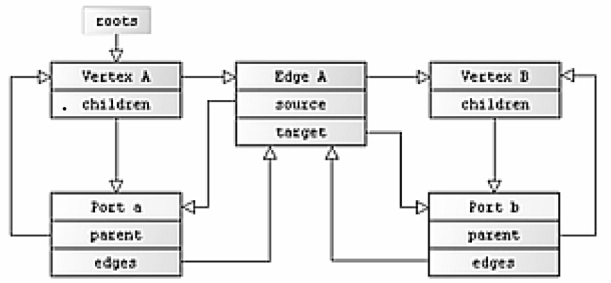


Figure 3: Labelling of Edges with Simulation Results



(a)



(b)

Figure 5: (a) Simple Diagram (b) Graph Topology of the Simple Diagram

The JGraph model provides the data for the graph. It consists of connection information and cells which may be vertices, edges or ports. The connection information that is stored in the model is defined using ports because they make up an edge's source or target (Alder, 2001).

The graph structure provides methods to retrieve the source and target port of an edge and to return the edges that are connected to a port. Refer to Fig 5 for an example of a graph topology for a basic diagram.

Following the JGraph architecture every mechanical engineering component as well as all connections between components can be called cells, because a cell is the superclass of a vertex, edge and port. Cells have attributes that define visual information such as the size,

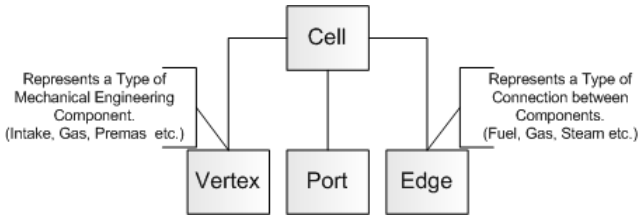


Figure 6: Relationship Between Cells and Vertices, Edges and Ports.

position, shape and rotation of the cell. The attributes of a cell are stored in an attribute map. The maps provide an easy and convenient way to customise the display and behaviour of any cell, and hence any vertex edge and port (because a cell is the superclass). For the purpose of this project a cell, if it is a vertex, also stores an icon representing a particular mechanical engineering component.

To store data that is not needed for rendering, a cell may also hold a reference to `userObject`. A `userObject` is of type `Object` and in this way provides a way to associate any object with a cell. The link between a cell and simulation data for that cell is established via the `userObject`.

In summary, a cell may either be a vertex, a port or an edge and a cell contains an attribute map that defines how it is to be rendered. For the purpose of this project, a vertex represents a mechanical engineering component and an edge represents a type of connection between two mechanical engineering components (see Fig 6).

Table 2: Code Snippet of a Sample Output File for a Jet Engine Simulation

```
...  ***** TURBINE 1 PARAMETERS *****
CNSF = .10000E+01 ETASF = .96437E+00
TFSF = .10000E+01
DHSF = .13827E-03
TF = 48.856 ETA = .85000 CN = 1.000
AUXWK = .35000E+07 DELHN = .11500E-05
ARATIO = .1300E+01
DELHT = .62000E-01
***** CONVERGENT NOZZLE 1 PARAMETERS
*****
Area = .5310 Exit Velocity = 67.66
Gross Thrust = 1016.54
Nozzle Coeff.  = .97021E+00
Scale Factor on above Mass Flows,
Areas, Thrusts & Powers = 1.0000
Station F.A.R. Mass Flow Pstatic
Ptotal Tstatic Ttotal Vel Area
1 .00000 15.194 1.00000 1.00000
288.00 288.00 .0 ******
2 .00000 15.194 ****** .99000 ******
288.00 ****** ******
3 .00000 14.434 ****** 12.37500
****** 650.47 ****** ******
4 .02019 14.726 ****** 11.50875
****** 1350.00 ****** ******
5 .01918 15.485 ****** 11.50875
****** 1318.40 ****** ******
6 .00000 .000 ****** .00000 ******
.00 ****** ******
7 .01918 15.485 ****** 1.01846 ******
821.04 ****** ******
8 .01918 15.485 ****** 1.00863 ******
821.04 ****** ******
9 .01918 15.485 1.00000 1.00863
819.03 821.04 67.7 .5310
10 .00000 .760 ****** 12.12750 ******
650.47 ****** ******
11 .00000 .760 ****** 12.00623 ******
650.47 ****** ******
12 .00000 .000 ****** .00000 ******
.00 ****** ******
...
```

6. SOME KEY FUNCTIONALITIES

Long-term persistence is the technical term given to the concept of saving the state of a program and reloading it at a later stage. Traditionally, Java serialization was used to achieve long-term persistence. Serialization works by writing the entire state of an object into a byte stream. This means that to save a program, all the important objects that make up the state of the program are written to a byte stream. The byte stream can later be deserialized in such a way that the state the program was in when it was saved is recreated (Winchester and Milne, 2009).

There are several problems with this approach. The mechanism used to deserialize an object stream relies on the internal shape of the classes that make up the objects to remain unchanged between encoding and decoding (Winchester and Milne, 2009). This means that any changes to the classes, such as adding or renaming some fields or methods between the time that the program was saved and retrieved, will cause deserialization to fail. Clearly serialization is not a viable solution for this project because a tool such as a graphical user interface is likely to undergo many refinements and extensions in the future.

To circumvent this problem an XMLEncoder was used. An XMLEncoder takes a very different approach to long-term persistence. Instead of storing a bit-wise representation of the field values that make up an object's state, the XMLEncoder stores the steps necessary to create the object through its public API (Winchester and Milne, 2009). The advantage is that many changes can be made to the implementation of a class while preserving backward compatibility through the API. For a thorough discussion on XMLEncoding refer to (Milne, 2009).

Drag and drop refers to functionality that allows a user to click on a mechanical engineering icon and drag and drop it anywhere into the document. While dragging a preview of the icon appears at the location of the cursor. This is commonly also known as drag and ghost since the preview icon is usually somewhat transparent. The drag and ghost functionality was implemented to make the graphical user interface more user-friendly.

To store simulation data the concepts described in the *JGoodies*(Lentzsch, 2009) project, such as data binding and data validation were used. The fundamental concept behind JGoodies Binding is that it synchronizes object properties with Swing components. It also helps to represent the state and behavior of a presentation independently of the GUI components used in the interface. JGoodies Validation on the other hand helps to validate user input in Swing applications and to report validation errors and warnings.

In this project each mechanical engineering component is represented by an object that exposes all simulation properties associated with that component through getter and setter methods. The data for a component is captured by a dialog box. However, before the data is accepted it is passed to a special validator object. Each

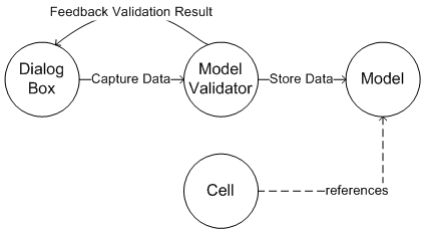


Figure 7: How Simulation Data is Stored.

mechanical engineering component has its own validator. The validator uses regular expressions to validate that the user input conforms to the expected input. If the data does not conform the user receives immediate feedback through the dialog box, because the input fields that contain errors turn red (see Figure 7).

7. RESULTS AND LIMITATIONS

Arbitrary energy systems can be visually drawn with a high degree of precision and flexibility. Flexibility and generalisation is achieved in part by using the most elementary mechanical engineering components as building blocks for the system. More complex components are created by combining the elementary building blocks.

The graphical interface is modelled on popular graph drawing programs such as *Microsoft Visio*. Conventions used in established software packages are followed to make the software more user friendly. With this in mind, the engineering components are represented by icons in the toolbar, which can be dragged and dropped anywhere in the document window. Components are selected by clicking on them, or by dragging a marquee selection window. Multiple components can be selected/deselected by following the *Microsoft Windows XP* convention of holding down the *Ctrl* button and clicking. Zoom functionality allows the user to zoom out to get a birds-eye view of the design or to zoom in on particular regions. Connections between components can start and terminate at any location on the component. This is significant because there is meaning associated with where a line terminates on a component. All engineering line styles are supported to represent the connections between components. The line styles differ in color, dash pattern and thickness. Air connections and fuel connections are represented by parallel lines. The line styles differ in thickness to ensure that they will be distinguishable from one another even if printed in black and white. A grid can also be displayed to make the alignment of components easier.

Several design patterns have been used to maintain a clean separation between the view (the graphics the user interacts with) and the model (the relationship between components and their properties). The design patterns further facilitate easier maintenance and extension of the system.

The entire state of the program, including the graphics, component properties and simulation results can be saved to file and recalled at a later stage.

The properties of components that previously had to be typed into a file are now captured by dialog boxes. The dialog boxes contain input hints that explain what the input fields mean and in what format the input is expected. The dialog boxes are directly associated with the graphical mechanical engineering components, which allows the user to visualize the impact the properties of a particular component should have on a system, by considering the connections between components.

The custom router written for this project, to draw parallel lines between the source and target components orthogonally, has been accepted by the JGraph community as a valuable contribution and is available on the JGraph Community Forum.

8. CONCLUSION AND FUTURE WORK

We managed to transform the Brakine software into a useful and versatile simulation program by creating a new intuitive graphical user interface that links seamlessly with the original Fortran program. However, there is still a lot of work that can be done to further improve the usability of the software. The bulk of the future work should focus on the following:

- JUnit Tests
  A comprehensive test suite has to be written for the entire project. This will not only find bugs in the current implementation but will ensure that any future modifications do not break the existing code base.

- Support for Printing
  Printing support still has to be coded. This will involve implementing the Java `Printable` and `Pageable` interfaces. Care has to be taken to ensure that diagrams scale and fit onto a page.

- Automatic Generation of Brick Data Indices
  Currently the user is still required to keep track of a lot input information for the simulation that could be generated automatically. This is particularly the case with brick data indices. More information on what brick indices are and what role they play in the Brakine simulation program can be found in (Roy-Aikins, 1988).

REFERENCES

Alder, G. (2001) Design and implementation of the jgraph swing component. `http://ontwerpen1.khlim.be/projects/netsim/jgraph-paper.pdf`

Alder, G. and Benson, D. (2001) Jgraph. `http://www.jgraph.com`

Benson, D. (2001) Jgraph and jgraphlayout pro user manual. `http://www.jgraph.com/pub/jgraphmanual.pdf`

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (2001). *Pattern-Orientated Software Architecture - A System of Patterns*, volume One. WILEY.

Jeffrey Friedl (2006). *Mastering Regular Expressions*, volume Three. O'Reilly.

Cengel, Y. A., and Boles M. (2006). *Thermodynamics: An Engineering Approach*, volume Four. McGraw-Hill.

Casella, F. and Leva, A. (2003). Modelica open library for power plant simulation: design and experimental validation. In *Proceedings of the 3rd International Modelica Conference*, pages 41–50, Linkoping, Sweden.

McKim, C. S., and Matthews, M. T. (1996). Modular modeling system model builder In *Proceedings of Intersociety Energy Conversion Engineering Conference*, pages 2039–2044, Washington D.C., USA.

Casella, F. and Leva, A. (2003). Supporting Protocol-Based Care in Medicine via Multiple Coordinated Views. In *Proceedings of the Second International Conference on Coordinated Multiple Views in Exploratory Visualization*, pages 118–129, London, England.

Hobart, J. (2001) Articles on usability and design. `http://www.classicsys.com/css06/cfm/articles.cfm`

Kim, D. W., Youn, C., Cho, B.-H., and Son, G. (2005). Development of a power plant simulation tool with gui based on general purpose design software. *International Journal of Control, Automation, and Systems*, 3(3):493–501.

Hanisch, D., Sohler, F., and Zimmer, R. (2004). ToPNet an application for interactive analysis of expression data and biological networks. *Bioinformatics*, 20(9):1–2.

Lentzsch, K. (2009) Jgoodies. `http://www.jgoodies.com/`

Tom Sawyer Software. (2009) Tom sawyer visualization.

Microsoft. Microsoft visio. `www.microsoft.com/office/visio/`

Milne, P. Using xmlencoder. `http://java.sun.com/products/jfc/tsc/articles/persistence4/`

O'Madahain, J., Fisher, D., and Nelson, T. (2003) Java universal network/graph framework. `http://jung.sourceforge.net/index.html`

Pendleton, B., Heer, J., Li, J., and Beckmann, C. (2007) The prefuse visualisation toolkit. `http://prefuse.org/`

Reyes, J. C. (2004). A graph editing framework for the streamit language. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute Of Technology.

Roy-Aikins, J. E. A. (1988). *The VATEMP Manual - A Supplement to the Turbomatch Users' Guide*. Cranfield Institute of Technology,Cranfield.

Roy-Aikins, J. E. A. (1995). Brakine: A programming software for the performance simulation of brayton and rankine cycle plants. In *Proceedings of the Institution of Mechanical Engineers, Part A, Journal of Power and Energy*, 209(2):281 – 286.

Thermoflow (2009). Thermoflow. `http://www.thermoflow.com/`

Valyi, R. , and Ortega, E. (2004) Emergy Simulator, an open source simulation platform dedicated to system ecology and emergy studies In *Proceedings of IV Biennial International Workshop Advances in Energy Studies*, pages 349–360, Campinas, Brazil. .

Winchester, J. and Milne, P. (2009) Xml serialization of java objects. `http://jdj.sys-con.com/read/37550.htm`

yWorks. (2009) yfiles. `http://www.yworks.com/`

AUTHOR BIOGRAPHIES

ZYGMUNT L. SZPAK is a Master's student at the School of Computer Science at the University of KwaZulu-Natal, South Africa. His general research interests include Artificial Intelligence, Image Processing, Computer Vision and Pattern Recognition. Currently, the central theme of his research is on real-time tracking and modelling of the behavior of ships, in a maritime environment. His email is `zygmunt.szpak@gmail.com`.

JULES R. TAPAMO is Associate Professor at the School of Computer Science at the University of KwaZulu-Natal, South Africa. He completed his PhD degree from the University of Rouen (France) in 1992. His research interests are in Image Processing, Computer Vision, Machine Learning, Algorithms and Biometrics. He is a member of the IEEE Computer Society, IEEE Signal Processing Society and the ACM. His email is `tapamoj@ukzn.ac.za`.

JOE ROY-AIKINS is Associate Professor at the School of Mechanical Engineering at the University of KwaZulu-Natal, South Africa. His email is `aikinsj@ukzn.ac.za`