Glyndŵr University
Glyndŵr University Research Online

Computing | Computer Science

11-1-2009

Integration Methodologies for Disparate Software Packages with an Emphasis on Usability

Lyndon Evans

Vic Grout
Glyndwr University, v.grout@glyndwr.ac.uk

Dave Staton

Dougie hawkins

Follow this and additional works at: http://epubs.glyndwr.ac.uk/cair

Part of the Computer Engineering Commons

Recommended Citation
Evans, L., Grout, V, Staton, D. & Hawkins, D., (2009)'Integration Methodologies for Disparate Software Packages with an Emphasis on Usability. (Proceedings of the Fifth Collaborative Research Symposium on Security, E-Learning, Internet and Networking SEIN 2009, 26-27 November 2009, pp81-90 held in Darmstadt, Germany: Plymouth University

This Conference Paper is brought to you for free and open access by the Computer Science at Glyndŵr University Research Online. It has been accepted for inclusion in Computing by an authorized administrator of Glyndŵr University Research Online. For more information, please contact d.jepson@glyndwr.ac.uk.

Integration Methodologies for Disparate Software Packages with an Emphasis on Usability

Abstract
This paper describes a novel approach to program script generation. The goals are twofold: to allow a number of different software packages to be implemented together and to permit a user with little or no programming skill to produce executable code. The principles and requirements of such a system are discussed and an outline approach is suggested. A case study based on electric motor design is presented for which early results are encouraging. Future development is considered in conclusion.

Keywords
activeX, automation, flowchart, integration, machine-generated script

Disciplines
Computer Engineering

Comments
This paper was presented at the Fifth Collaborative Research Symposium on Security, E-Learning, Internet and Networking (SEIN 2009), 26-27 November 2009, which was held in Darmstadt, Germany. It was published by the University of Plymouth and the symposium proceedings are available at http://www.cisnr.org

This conference paper is available at Glyndŵr University Research Online: http://epubs.glyndwr.ac.uk/cair/28

Integration Methodologies for Disparate Software Packages with an Emphasis on Usability

Lyndon Evans[1 2], Vic Grout[1], Dave Staton[2] and Dougie Hawkins[2]

[1]Centre for Applied Internet Research, Glyndŵr University, Wales, UK
[2]Motor Design Ltd., Ellesmere, Shropshire, UK
{lyndon.evans|dave.staton|dougie.hawkins}@motor-design.com,
v.grout@glyndwr.ac.uk

Abstract

This paper describes a novel approach to program script generation. The goals are twofold: to allow a number of different software packages to be implemented together and to permit a user with little or no programming skill to produce executable code. The principles and requirements of such a system are discussed and an outline approach is suggested. A case study based on electric motor design is presented for which early results are encouraging. Future development is considered in conclusion.

Keywords

ActiveX, Automation, Flowchart, Integration, Machine-Generated Script

1. Introduction and Background

An ever-increasing number of software systems are becoming available to industry with the aim to increase productivity and efficiency. The drive to integrate systems is becoming a priority for business managers (Larrucea, 2008). It is also becoming increasingly important for industry to reduce the effort required of customers to integrate their software (Mazhelis *et al*, 2007).

The components within these software packages can be mission critical to an organisation's success, and the importance of providing these components for use within an automation process is becoming more widely recognised (Crnkovic *et al*, 2005).

Research for this paper is based on a software development project to facilitate the automation of a range of CAD software packages supplied to the electric motor industry by Motor Design Ltd. Each package specialises in different aspects of the design process for a motor e.g. electromagnetic analysis (SPEED), thermal analysis (Motor-CAD) and drive circuit simulation (Portunus).

Both Motor Design Ltd. and end users of these packages are increasingly exploiting the benefits of automation to improve the accuracy and delivery times of proposed designs. The project and proposed methodology are in the early stages of development, but is able to interact with 2 CAD packages to gather and analyse data.

The remainder of this paper is organised as follows: Section 2 discusses the principles and methodology being investigated, with an introduction to automation methods. Section 3 details a case study employing the methodology being researched, including some illustrations of a developing software project employing the researched methodology. Finally, Section 4 offers some early results and conclusions and discusses future work.

2. Principles and Methodology

2.1. Linking disparate software packages together

The ability to successfully control a software package by means other than manual use of its user interface requires a precise sequence of actions to be carried out. These actions can perform tasks such as the input or output of data, selecting options and performing processing tasks which are likely to be a speciality of the application being automated. Methods for software automation fall into two main categories:

1. Recording user interaction with an application's Graphical User Interface (GUI).

2. Direct interaction with an application's automation interface.

1. Recording user interaction with an application's GUI.
This method uses software to record a sequence of user actions applied to a GUI in one or more applications. These actions can include keystrokes, mouse button clicks or click and drag operations such as selecting a block of text prior to copying.

2. Direct interaction with an application's automation interface.
The use of this method is dependant on an application having a built in automation interface. By using compiled code or script, the automation interface can be used to perform tasks within the application under external control. This method bypasses the GUI, and as such the application is not required to display it.

Both methods have advantages and disadvantages. The first method has simplicity and an ability to use almost any software. However it requires the user to record the actions required in the automation process being designed, and requires some careful planning to achieve the desired results. The second method is more efficient while executing, particularly in the transfer of data. Its disadvantage however is that automation using this method requires programming skills which a user may not have.

Due to the requirements of the case study, the second technique is used and requires the use of scripting. Consequently, the issue of programming skills and syntax knowledge must be overcome, and it is the methodology for addressing this which is discussed in this paper. The methodology relies on the Common Object Model (COM) discussed by Minich *et al* (2008). COM is also referred to as ActiveX and it this technology employed in the applications which are utilised by the script.

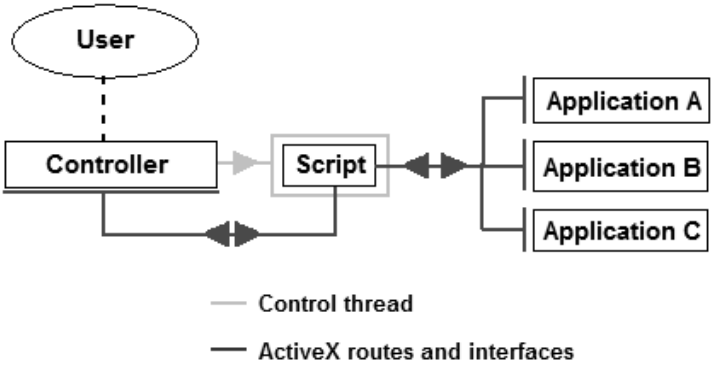


Figure 1: Automation framework

Figure 1 shows the framework for the methodology being researched. The controller is an application which can build a script according to the user's design, and control it during execution. The script can control one or more other applications using their built-in automation interfaces.

2.2. 'Programming for non-programmers'

The concept and challenge behind the methodology is to allow complex automation scripts to be generated by users without manually having to write them and test for syntax errors etc. This is discussed in 3 sections: Structure, Detail and Control.

2.2.1. Structure

The automation process is to be designed as a flowchart. This is to be achieved by the use of an interface incorporating a graphical flow diagram builder to describe the automation process required by the user.

Block type	**Linkage rules**	
	Preceding blocks	**Following blocks**
Start (Terminal)	None	One only
Stop (Terminal)	Min 1	None
Process	Min 1	Max 1
Decision	At least one process block	At least one defined for both true and false paths

Table 1: Flowchart block rules

A summary of the rules for creating a flowchart is shown in Table 1. The main issue with converting a flow chart created by the user into a script is to ensure that the generated script executes the automation process stages in the order, branching and iteration cycles dictated by the flowchart.

One method which has been researched is to analyse the diagram to identify the boundaries of loop and branching structures in the flowchart, in order that the blocks within may be contained within the required script syntax.

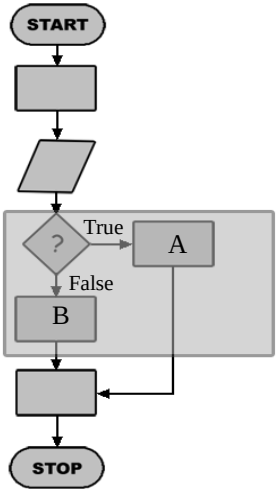


Figure 2: Boundary of a branch structure

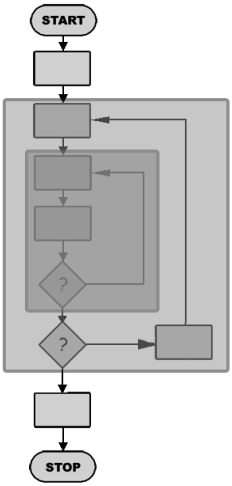


Figure 3: Nested loop boundaries

In the case of a branch structure (Figure 2), it is necessary to start from the originating decision block and trace along both true and false paths to the point where they converge. The instructions from each block within the boundary can then be placed appropriately within the branching syntax of the script e.g.

If '?' then
  Perform A
Else
  Perform B
End If

A similar approach can be used with loops in the flowchart. In this case, a decision block is at the base of the loop, and the top is defined where the return path converges with the main stack. Figure 3 shows an example of a nested loop with the boundaries identified.

Nested structures require that the function to build them in the script be called recursively, and some limited success was achieved with this technique. However, these methods of following the flowchart structure can be technically challenging with complex diagrams, and does impose a limitation on the way they can be built, see Figure 4.

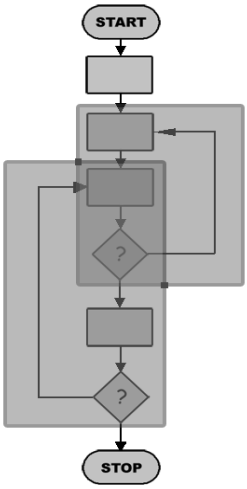


Figure 4: Overlapping loops

The diagram has been built using the simple rules defined in Table 1. As can be seen, these simple rules do allow the forming of overlapping loops or branching structures, which cannot be implemented in script syntax without causing an error.

An alternative technique for converting the flowchart into a script is to define each block (apart from both terminal blocks) as discrete functions. The output of each function is a reference to the next function (block) to be executed.

To implement this alternative method, a loop is built into the main procedure of the script which calls a function by reference from output function which preceded it. Once the loop has been initialised with a function to start with, the loop calls the remaining functions in the order dictated by the flowchart design. In the context programming this in Visual Basic (VB), the function selection would be performed in a *Select...Case* statement.

```
Do While blockToExecute <> ""
Select case blockToExecute
   case "FlowActionBlock1"
   blockToExecute = FlowActionBlock1()
   case "FlowActionBlock2"
   blockToExecute = FlowActionBlock2()
   case "FlowActionBlock3"
   blockToExecute = FlowActionBlock3()
   case "FlowTerminalBlock2"
   blockToExecute = FlowTerminalBlock2 ()
End Select
Loop
```

In the above example the loop is halted by returning an empty string (which is essentially what is done by the STOP block function). In evaluating methods for

turning a flowchart into a script, the second technique offers far more flexibility and simplicity in execution. It is counter-productive to use the first technique to try and build an efficient script (i.e. as would be written by an experienced programmer) by analysing the flowchart and identifying the boundaries of loops and branching structures.

2.2.2. Detail

The methods discussed can deal with the structure of the flowchart. The issue of building syntactically correct script statements within each block according to the user's requirements, in the context of data input, output and manipulation, also needs to be addressed. It will be a requirement to shield the user from the syntax of the script language, while presenting the user with the necessary information which preserves his or her awareness of the block functionality.

2.2.3. Control

It is important to note that the script handler, once passed a script for execution is an independent process. It would be risky to allow this process to be executed without some form of user control mechanism. The control thread illustrated in Figure 1 allows the process to be initiated or paused. The use of individual functions allows a check for a stop request to be performed at each block stage, thus allowing the user to stop the automation process in the event of a problem.

3. Case Studies and Examples

3.1. Introduction

Research into this methodology is underpinned by a software development project called 'Workbench' to facilitate the automation of several CAD packages used in the design of electric machines. Automation via scripting is used to run repeated calculations within the CAD packages for the purposes of optimising designs for objectives such as cost, performance or efficiency.

3.2. Design screen

At the heart of the research, and key to the effectiveness of the software is the flowchart design area, see Figure 5. This is where the user builds an automation process graphically using both standard and customised diagram blocks and links to indicate the direction of process flow.

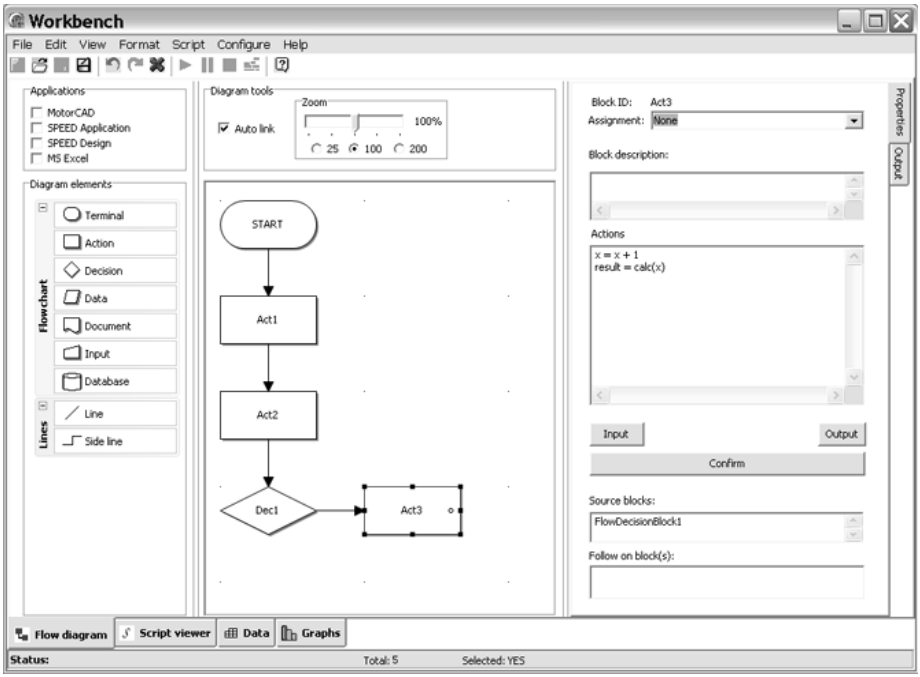


Figure 5: Flowchart being constructed

Some features of the design interface have been added specifically to aid the diagram building process. An automatic linking facility has been included to save the user from having to manually link blocks. The user can nominate an existing block by selecting it, before adding a new block and the application will automatically link both blocks.

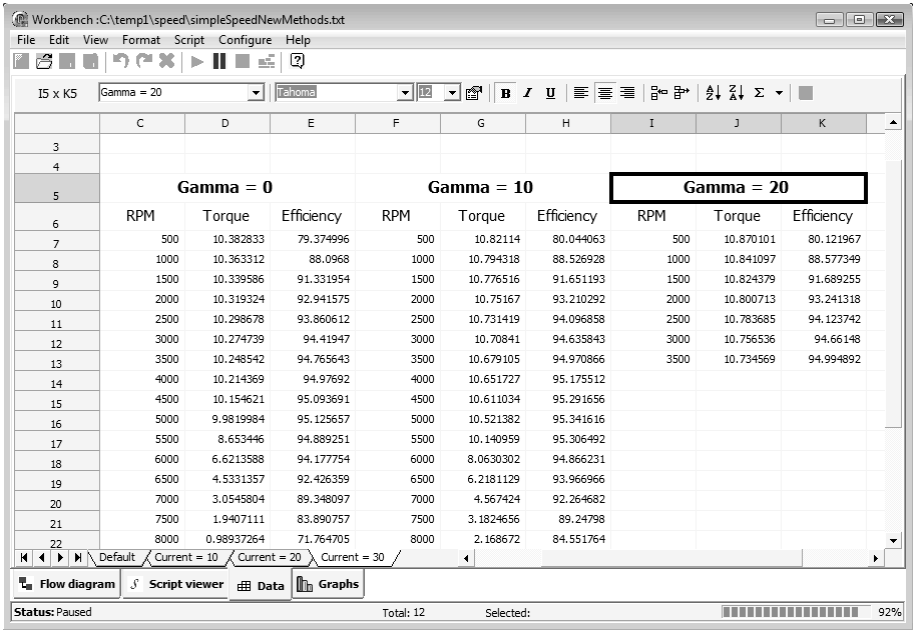
Immediately to the right of the design area is a panel where the user can define the actions which occur within the selected block. At present, VB script is typed directly into this area, including references to automation objects and any data manipulation which is required. The improvement of this area is the next phase of the development project and will increase the user benefit of Workbench and lies at the heart of the researched methodology.

When requested by the user, Workbench performs some checks on the flowchart to ensure it follows the rules defined in Table 1. If successful, the flowchart is processed by Workbench, which constructs a self-contained VB Script to perform the desired automation operations. This is passed over to an internal script handler which, when executed, will interact with the various software packages using their ActiveX interfaces.

3.3. Spreadsheet

One of the existing methods of automating the CAD applications is to use a VBA script from within Microsoft Excel, using the spreadsheet to store the data generated

by the various applications. To avoid this reliance on Excel, Workbench has its own spreadsheet for the collection and storage of data.



Gamma = 0			Gamma = 10			Gamma = 20		
RPM	Torque	Efficiency	RPM	Torque	Efficiency	RPM	Torque	Efficiency
500	10.382833	79.374996	500	10.82114	80.044063	500	10.870101	80.121967
1000	10.363312	88.0968	1000	10.794318	88.526928	1000	10.841097	88.577349
1500	10.339586	91.331954	1500	10.776516	91.651193	1500	10.824379	91.689255
2000	10.319324	92.941575	2000	10.75167	93.210292	2000	10.800713	93.241318
2500	10.298678	93.860612	2500	10.731419	94.096858	2500	10.783685	94.123742
3000	10.274739	94.41947	3000	10.70841	94.635843	3000	10.756536	94.66148
3500	10.248542	94.765643	3500	10.679105	94.970866	3500	10.734569	94.994892
4000	10.214369	94.97692	4000	10.651727	95.175512			
4500	10.154621	95.093691	4500	10.611034	95.291656			
5000	9.9819984	95.125657	5000	10.521382	95.341616			
5500	8.653446	94.889251	5500	10.140959	95.306492			
6000	6.6213588	94.177754	6000	8.0630302	94.866231			
6500	4.5331357	92.426359	6500	6.2181129	93.966966			
7000	3.0545804	89.348097	7000	4.567424	92.264682			
7500	1.9407111	83.890757	7500	3.1824656	89.24798			
8000	0.98937264	71.764705	8000	2.168672	84.551764			

Figure 6: Spreadsheet being populated with data sourced from SPEED

Figure 6 shows the spreadsheet cells being populated with data as it is generated by the SPEED application as it is automated via the script. The ActiveX commands for Workbench itself allow data to be sorted into columns and headings defined in the script.

3.4. Graphing

Given that the applications in the case study produce mostly numerical data as output during automation, the inclusion of graphing features to interpret the gathered data has been necessary.

A graph type (e.g. 2D, 3D and contour type) associated with an individual sheet in the workbook can be defined when a sheet is added either before or during the automation process. Figure 7 shows a graph being plotted from data being generated during an automation script interacting with SPEED, as it calculates torque and efficiency values from a motor design over a range of operating parameters.

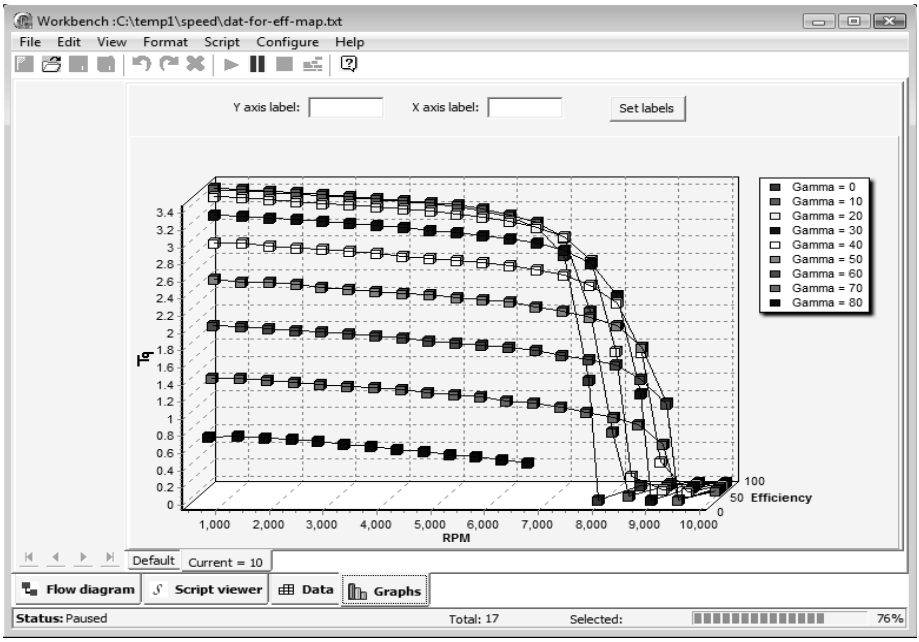


Figure 7: Graph generated from SPEED data

4. Results and Conclusions

The project is in the early stages of development, but some important benefits of the automation technique have been realised. By gathering the data in an organised fashion, as facilitated by Workbench, it is possible to summarise data and perform specialised analysis, such as the creation of efficiency maps for proposed motor designs. Figure 8 shows an efficiency map generated from data gathered using a script built and executed from Workbench. Efficiency maps are used to identify the most efficient zone of operation for a motor, particularly for their use in electric vehicles.

In order to further enhance the usability of Workbench and the underlying methodology, further investigation is to undertaken into the block detail issues highlighted in Section 2.2.2. It is likely that the user interface will offer options for filtering or other processing requirements on raw data. These as well as other aspects of block detail will be presented to the user in such a way as to avoid the manual entry of script syntax.

In conclusion, the authors believe that the proposed methodology can offer an effective script construction tool for non-programmers, and allow the complex automation of disparate software packages to be achieved with less effort.

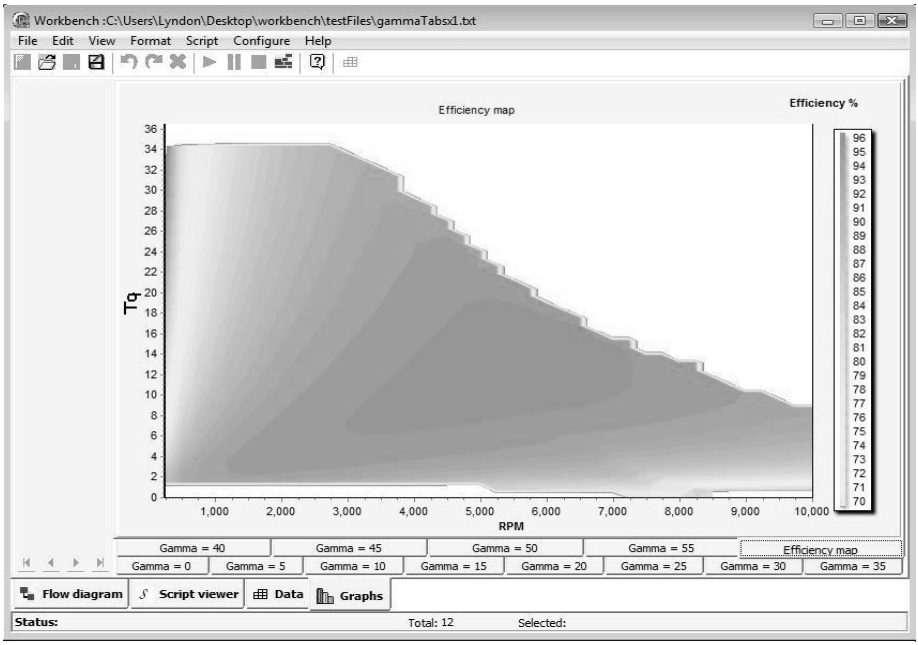


Figure 8: An efficiency contour map generated from gathered data

5. References

Crnkovic, I. Schmidt, H. Stafford, J. and Wallnau, K. (2005) ‘Automated Component-Based Software Engineering.’ *Journal of Systems and Software*, vol. 74, issue 1, 1 - 3.

Larrucea, X. (2008) 'Method Engineering Approach for Interoperable Systems Development'. *Software Process Improvement and Practice*, vol. 13, 127-33.

Mazhelis, O., Tyrvainen, P. and Viitala, E. (2007) 'Modelling software integration scenarios for telecommunications operations software vendors'. *2007 IEEE International Conference on Industrial Engineering and Engineering Management*, vol. 1-4, 49-54.

Minich, M., Harriehausen-Mühlbauer, B., Wentzel, C. and Phippen, A. D. (2008) 'Software Industrialization in Systems Integration.' *Proceedings of the Fourth Collaborative Research Symposium on Security, E-Learning and Networking SEIN 2008*. Glyndŵr University, UK.