Web Service Workflow
Individual Project Report, June 2001

Dinesh Ganesarajah, MEng Computing
Imperial College Dept of Computing
http://www.doc.ic.ac.uk/~dg197/project
Acknowledgements

*Emil Lupu* for helping me stay focused at the start, acting as a sounding board for my ideas throughout and constantly re-iterating the qualities of a ‘good project’.

*The inhabitants of the Project Lab*, with *inhabitants* being the operative word, for general amusement along the way.
Abstract

Workflow originates from organisational management and is concerned with processing nodes and the lines by which work flows between them. As workflow has become automated and entered Distributed Systems, it has largely remained faithful to the graph-based approach of lines and nodes, and the flow of work between them.

Web Services are a new Distributed Systems technology based on SOAP. SOAP is an XML W3C protocol that provides an RPC mechanism that wraps around heterogeneous object models, such as DCOM and CORBA, allowing these objects to call each other’s functions.

This thesis describes the design and implementation of a Workflow Management System for building Web Services from a workflow of existing Web Services. In this endeavour, the contributions of this work include:

- **SafeFlow**, a workflow style that tackles the problems of current approaches by considering workflow as a programming (structured graph) problem rather than a pure graph problem. Recursive encapsulation of workflow is introduced as a structuring mechanism.

- **Probabilistic Method Calls**, which given data and an invocation, analyse the data to probabilistically determine the precise method call for automatic invocation. E.g. A CD player is invoked, but the exact track to invoke is decided by room temperature and other data.

- **Concurrent Visitors**, an advanced implementation of the Visitor Pattern used for workflow enactment that allows for multiple concurrent visitations over several nodes in a data structure, with co-ordination between Visitors achieved via an overlaid publish-subscriber protocol.

The conclusion reflects as to whether workflow over Web Services is a Silver Bullet in Software Engineering processes.
1 Contents

1.1 Table Of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Contents</td>
<td>4</td>
</tr>
<tr>
<td>1.1 Table Of Contents</td>
<td>4</td>
</tr>
<tr>
<td>2 Introduction</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Chapter Summary</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Web Services</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Workflow</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Scenario</td>
<td>10</td>
</tr>
<tr>
<td>2.5 Motivation</td>
<td>11</td>
</tr>
<tr>
<td>2.6 Scope</td>
<td>12</td>
</tr>
<tr>
<td>2.7 Reading Guide</td>
<td>13</td>
</tr>
<tr>
<td>2.8 Contribution And Pointers</td>
<td>14</td>
</tr>
<tr>
<td>3 Background</td>
<td>16</td>
</tr>
<tr>
<td>3.1 Chapter Summary</td>
<td>16</td>
</tr>
<tr>
<td>3.2 Building Web Services</td>
<td>16</td>
</tr>
<tr>
<td>3.3 Workflow Management System Considerations</td>
<td>20</td>
</tr>
<tr>
<td>3.4 Enterprise Workflow</td>
<td>23</td>
</tr>
<tr>
<td>3.5 Candidate Workflow Systems Of Interest</td>
<td>26</td>
</tr>
<tr>
<td>3.6 Other Systems</td>
<td>30</td>
</tr>
<tr>
<td>3.7 Interoperation of Web Services</td>
<td>32</td>
</tr>
<tr>
<td>3.8 Web Service Issues</td>
<td>34</td>
</tr>
<tr>
<td>4 Overture</td>
<td>36</td>
</tr>
<tr>
<td>4.1 Chapter Summary</td>
<td>36</td>
</tr>
<tr>
<td>4.2 The Approach</td>
<td>36</td>
</tr>
<tr>
<td>4.3 General Architecture</td>
<td>37</td>
</tr>
<tr>
<td>4.4 System Map</td>
<td>41</td>
</tr>
</tbody>
</table>
2 Introduction

2.1 Chapter Summary

- The Web Services model of Distributed Computing is introduced.
- Workflow over Web Services as a means to creating Web Services is motivated.
- A guide to the remainder of this thesis is given, with pointers to sections of interest.

2.2 Web Services

2.2.1 Origins

The pioneers of Web Services describe them as the natural evolution of the Web. [AR001] describes three development phases:

- **The Document Web** is the original WWW phenomenon, used mainly by organizations and individuals to purely publish information on their work, products, services or anything else.

- **The Application Web** is the advancement whereby businesses started selling products over the Web. Additionally, more interactive web sites appeared, like Web Mail. Web Sites of this phase were more than just published web pages – they were Web Application Servers running EJB or other programming code, possibly connected to other distributed computers sharing the workload when there are large transaction rates.

- **The Services Web** is the current emerging phase of the Web, where the Web Application Servers of the previous phase now communicate with each other, typically using Simple Object Access Protocol (SOAP), to do useful work. The Services Web evolves out of the desire to perform transactions in an open and automated environment like the Internet, rather than going through other mechanisms such as EDI or manual processing. The servers of this phase no longer host Web Sites, but ‘Web Services’.

Web Services can be described in terms of the following characteristics:

- Web Services are black box components hosted on HTTP servers that encapsulate behaviour. They are represented in such a way that the object model they use underneath is hidden, though the functionality of the component are not.
Web Services typically communicate with each other using SOAP over HTTP. SOAP is an RPC mechanism that is independent of the object model used by a communicating party or Web Service. SOAP will usually wrap around DCOM, CORBA, EJBs or other objects, allowing them to interoperate.

A Web Service will publish its API, the description of itself, in a standard format (WSDL) that allows other Web Services to inspect the API and invoke the Web Service dynamically at run time; in essence a liberal form of reflection.

Web Services can be discovered at run-time for dynamic binding of collaborations. Several Web Services can be orchestrated to perform a series of functions in a workflow, and the orchestrating party can itself be a Web Service.

Web Services are units of extremely low coupling and high cohesion that can communicate with each other with low dependence on the other party. The use of reflection and SOAP, a protocol that imposes very little on communicating parties, allows Web Services to interoperate in a minimal dependent way. The ability to orchestrate Web Services allows for the creation of complex applications.

2.2.2 Technology Comparison

The Services Web is essentially another example of distributed computing, and it could be argued that existing technologies like DCOM and IIOP/CORBA already provide an environment in which it is possible for distributed computers to interoperate to perform transactions and do useful work on their users’ behalf. Several papers [IBM00I, GG00I, PL00M] have recognized this argument, all of which put forward similar counter propositions.

The existing approaches, like DCOM and IIOP/CORBA, have produced systems that are highly coupled. These methods require agreement and shared context among systems in different organizations to allow for an environment of open and dynamic transactions – typically the same distributed object model needs to be used at either side by communicating parties. Additionally, in a system composed of independent sub-systems run by separate parties, when the output of a subsystem changes, the old statically bound collaborations will cause the overall system as a whole to break.

The essential difference to the existing approaches lies within the forced decoupling of components. While in existing approaches it is possible to obtain for components the characteristics of Web Service components, these are not enforced. All the components in the Services Web system are Web Services, and all adhere by given characteristics, such as reflection, dynamic discovery, dynamic binding and SOAP transport. The use of SOAP, and providing a communication mechanism a layer above object models, allows for greater heterogeneity in communicating parties, as they only need provide a SOAP wrapper over their object model. Additionally, the desire of the Services Web is to allow parties to not only freely and completely interoperate, but also do so in an environment that is ubiquitous and widely accepted, such as over HTTP.
2.3 Workflow

2.3.1 Brief History And Context

Workflow is a truly vast area that is situated at the juncture of Management, various Social Sciences and Computing. Manolescu [MWF] gives a brief history of the origins of its current form. In the 1970s, among research done focusing on procedures for Office Information Systems, Xerox PARC developed an experimental office information system called Officetalk and consequently recognized the importance of developing models of office procedures. Research during the 1980s placed more emphasis on process models based on Petri net models and collaborative interaction models based on Speech Acts. From then on workflow technology expanded firstly into fields like office automation and document imaging, and then to a wide range of others. Mohan [WFNATO] identifies four broad categories of workflow style: production, administrative, collaborative and ad-hoc. These styles, as well as the different domains in which workflow is used, dictate how the workflow system is designed and used. Different products in different domains range from groupware like Lotus Notes/Domino to publishing workflow used after Adobe Production Technologies and to those that are e-commerce orientated like Ariba CSN. Work on workflow in the Web Services domain, the focus of this project, is almost non-existent.

2.3.2 Definition

The Workflow Management Coalition, a leading organisation in the field, categorises the workflow subject largely in terms of three core concepts: business process, workflow, and workflow management system. Firstly, a business process is:

'A set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships.' [WFCM-G]

Within an organisation, this could be anything from the human resources recruitment process to ordering stationary. Although the term ‘business’ is used, because this technology is used largely in businesses at the moment, the business process concept could represent effectively any series of activities in any establishment that abstractly is an ‘organisation’, be it a company, household or anything else. Workflow is distinctly different to a business process in that it is:

'The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.' [WFCM-G]

The key difference from business process to workflow is that the later implements the former, with automation referring to there being a procedure to follow. An automated business process could be ordering stationary where in the workflow the user checks what stationary is required, enters the stationary requirements into a system, which verifies the order and dispatches it to a supplier automatically. The supplier eventually delivers this order after performing some form of workflow on their side. All human intervention is also considered part of the workflow. In addition to these two concepts, there is also the concept of a workflow management system:
A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.’ [WFCM-G]

The workflow management system (WFMS) is the part where Computing plays a role. The WFMS co-ordinates and monitors the workflow. Returning to the stationary example, the application into which the order was entered would be linked to a workflow engine, which would keep track of the order, possibly though other application software, until completion. The workflow engine and the application software supporting it make up the WFMS as a whole. The workflow engine is involved in all parts of the runtime execution of the workflow, including the dispatching of work to the right participant at the right time. It is also worth noting here the separation between the higher-level abstraction describing the process logic and the programming code of the software, which performs the required actions to achieve the process logic [MWF].

2.4 Scenario

‘The problem today is that integrating with other services and touching different devices remains difficult, because tools and common conventions for interconnection are lacking’ [PL00M], and this is the problem that Web Services aims to tackle. Web Services aim to reduce manual work by introducing a way for independent systems to communicate with each other to more effectively and tackle a related set of problems in achieving a goal. A common example explained in [AR001] gives the scenario of corporate travel.

Usually, if someone wishes to plan a business trip, they will contact various parties independently. These parties could be the airline, car-rental service and hotel. Assuming they all have an Internet presence, the information from these services will typically be copied and pasted by the traveller to a corporate travel program to log the travel information for invoicing purposes. This alone requires some work, registering with the independent services and co-ordinating the information. However, if for example the airline were then to inform the traveller that the airline would be late, the traveller would then have to do even more work in communicating with the other services and the corporate travel program to make the necessary changes. The alternative scenario given by [AR001] is pictured as follows, where the corporate travel program and the different parties involved are Web Services:
In this new scenario, the corporate travel program co-ordinates all the information between the traveller and the different parties. This means that the traveller only needs to specify all the information once, and the program will then make the necessary arrangements with the other parties. If a detail changes, for example, the airline being late, the relevant party can contact the corporate travel program, which can then make changes in other arrangements and inform the traveller.

The example given by [AR001] can be extended to make clear some other features of Web Services. If the car-rental service cannot accommodate the changes due to the airline being late, it may be necessary to find a new car-rental service. Car-rental services will typically be catalogued in a Yellow Pages style Web Service, which can be accessed in a standard way. The corporate travel application can look-up a set of new car-rental services in the Yellow Pages Web Service. The returned set of car-rental Web-Services can be contacted to see if they can accommodate the traveller. The messages sent to each car-rental Web Service will be exactly the same, as they will be defined in a standard business domain protocol based on top of SOAP. Car-rental Web-Services that can accommodate the traveller can be pruned according to other criteria, such as cost, and then the arrangements can be made after they are confirmed with the traveller.

The above given example illustrates the Web Service ability of dynamic discovery and binding at runtime. In the example, it is likely that reflection will be used with each car-rental Web-Service to discover the function to invoke on the Web Service, before communicating the relevant query using the standard business protocol.

2.5 Motivation

The scenario introduced Web Services and gave an illustration of how independent Web Services can be used effectively in collaboration. However, the scenario did not explain how this collaboration is defined. One possibility is to programmatically define in a Web Service its dependencies on other Web
Services, or to programmatically define mechanisms in the Web Service that it will need to find and bind with other Web Services.

However, Web Services are expected to become as ubiquitous as web pages. HP has a project called Cool Town [COOL] where everything has a URI and can be represented by a Web Service. This ranges from businesses to handheld devices and even people. In such a world, there will be value in being able to put together workflow between Web Services with less effort than having to build a Web Service from scratch to define the collaboration.

For decades, the concept of workflow has existed as a way of separating at an abstract layer the control flow and information between components from the actual execution of the code in the components underneath. In defining the top-level flow abstractly above the components, the ability to rearrange and change the components is facilitated. But aside from this, there is also a reduction in defining these dependencies between components and enacting this collaboration.

This project aims to exploit the ideas behind workflow to enable collaboration of Web Services. While workflow largely descends from business process automation, the belief of this project is that the principles and ideas can be used in the composition and collaboration of Web Services representing businesses, devices and even people.

2.6 Scope

The objectives of this project are:

1. To build a Workflow Management System (WFMS) which will support the creation of new Web Services from existing ones.

2. To explore the area of Web Service workflow and consider the issues involved.

The deliverables for this project are split into the separable core functions that a Workflow Management System must be capable of.

- **Specification Tool**: This tool will allow for the creation of a workflow schema that can then be enacted.

- **Execution Environment**: The environment will be used to enact workflow described for one or more workflow schema instances.

- **Monitoring Tool**: This tool monitors and tracks the progress of workflow instances.

In considering the creation of a Workflow Management System, the workflow language used will be the vital aspect of the system. To this extent, a view perpendicular to the deliverables influences the deliverables in terms of the workflow language.

- The design and structure of the language needs to be considered to ensure extensibility and coverage of necessary functionality.
The process by which the language is used to create the workflow required by the user must be investigated.

The implementation of the language with respect to all parts of the system, to ensure above mentioned extensibility and coverage of necessary functionality.

An aspect that is not considered in this thesis is the semantics and specification of business-to-business communications, in terms of protocols for negotiation and commerce that may sit over SOAP communications.

2.7 Reading Guide

Background (Chapter 3) gives an overview of workflow and Web Services related work.

The body of the report, Overture (Chapter 4), Shared Mechanisms (Chapter 5), WorkflowEngine (Chapter 6) and User Application (Chapter 7), follow a style loosely inspired by Problem Frames [MJ].

An overview of the entire system is given at the end of Overture, and the different parts split into frames. The remaining three chapters describe each of the frames. Each frame, described by a section of a chapter, will sometimes have reoccurring sub-section titles.

The reoccurring sub-section titles include: Aim, giving the design aims within the frame. Derivation, describes how the problem was solved. Related Work and Alternatives compares the work in the frame to the state-of-the-art, referencing items mentioned in Background. There will also be sub-sections describing more detailed design and implementation, as well as sometimes sections assessing the solution within the context of the problem.

Overture, and in part Shared Mechanisms, act to organize and describe overlapping frames. An In Outline subsection at the start of each chapter organises the frames within the chapter as a whole.

Evaluation (Chapter 8) reinforces assessment of parts of the system in the body of the report, as well as reviewing more general and other aspects.

Conclusion (Chapter 9) gives possible extensions, insights from this work and evaluates the significance of this work to the wider context of Computing and the state-of-the-art.

Problem Frames [MJ] describe a method of identifying solutions to problems. In essence, Problem Frames identify the principal parts of a problem, and then apply a well-defined solution from several different problems that have the same principal parts. The Problem Frame itself is the general type of a set of similar problems and has principal parts, and a solution, but without the context of any particular problem, allowing for the Problem Frame to be applied to several scenarios. For example, the Workpiece Problem Frame identifies workpiece documents, with operation requests from users that share phenomena with the workpieces. A machine contains the workpieces, receiving them via the shared phenomena. Certain operation properties must hold after the machine has worked on the workpiece. The associated solution for this Problem Frame is to build a machine that changes workpieces in response to operation requests such that the operation properties hold. This Problem Frame is used, for example, in machines for document production, as well as ticketing machines; the skill is in identifying the principal parts of a problem and matching them to a Problem Frame. For
example, the Workpiece Problem Frame can be applied to the problem of submission and marking of Individual Project Reports – the machine to be built accepts reports, perhaps electronically, and processes them such that the operation properties of ensuring the reports are distributed and marks are entered in the system are achieved.

The idea of Problem Frames is used very loosely in some parts of this project; new Frames are not identified, but similar problems with similar principal parts are, so as to derive a well-defined solution that can be appropriately tailored.

### 2.8 Contribution And Pointers

Current workflow languages largely consider workflow as a graph problem, with control flow and data flow described in terms of lines in a graph; this is exemplified by the most recent language and first for Web Services, Web Service Flow Language [WSFL]. However, such an approach produces systems that are difficult to maintain and modify because control lines are similar to goto statements. There is also difficulty in structuring this workflow during the development stage.

The approach considered in this project is that of encapsulating boxes of control; an approach inspired by object-orientated methods for programming and the Interpreter Pattern. In this approach, each ‘box’, a Non-Terminal Expression, defines control within that box such that a Sequence Expression entails all constituent expressions are executed in order, while a Concurrent Expression entails all constituent expressions are executed in parallel; there are no lines defining the control path between the expressions, and control is in terms of recursively encapsulating Non-Terminals. Because each Expression in the language has the same characteristics as any other Expression, each Expression, irrespective of whether it is a Terminal or Non-Terminal, is able to hide the sub-workflow that it encapsulates. Modifications to any part of the workflow are contained within the expression concerned, and Non-Terminals additionally act as a structuring mechanism for decomposing large complex problems, and creating large workflow. Additionally, the entire workflow or any subsection can be deployed as a new Web Service, for use in another workflow itself or otherwise. The SafeFlow dialect described here is explained in sections 5.4 and 7.3.

It is sometimes the case that it is difficult to automate a method invocation because the process by which an invocation is normally made is systematic, whereas a non-systematic process is more desirable. There are cases where the un-deterministic nature of the real world means that the exact method call to invoke needs to account for probability. A stock trader, for example, may wish to buy stock, but cannot use a systematic if-then-style selection method to automate the process because if-then-style selections have difficulty in modelling conditional probabilities like 60% probability a certain method should be called given several data items, with the quantified probability changing dependent on these data item values. To resolve this problem, a naive Bayesian network is used as a wrapper around a set of possible method calls, modelling the relation between these different data items and the possible method calls; the method call with highest posterior probability is invoked. Using this wrapper, a stock trader could reduce the number of possible buy methods that could be invoked, because some of them encapsulate several methods, with the exact one decided based on the Bayesian model wrapping them. Probabilistic Method Calls are described in sections 5.7 and 6.5.
The Visitor Patterns allows for a way to separate functionality from data, such that the functionality can be defined independently and be modified without affecting a data structure. However, the standard implementation uses only one Visitor, which is inconvenient for operations on a data structure that require computation at two points in a data structure, particularly concurrently, as is the case with workflow enactment. Instead, several Visitors can be used over a data structure and co-ordination achieved by using a publish-subscriber mechanism between them. In this implementation, the effect is that parts of the data structure are flooded with Visitors that communicate between each other to achieve a common goal, such as executing several workflow expressions concurrently, before aggregating the results. *The Concurrent Visitors implementation of the Visitor Pattern is given in section 6.4.*
3 Background

3.1 Chapter Summary

• Web Service technologies are surveyed and introduced.

• Current workflow systems are examined and candidate systems for benchmarking against identified.

3.2 Building Web Services

3.2.1 Base Technologies

Creating a Web Service is relatively low overhead, since anything that implements a SOAP interface over HTTP and can return a WSDL self-description document could be a Web Service. This means even relatively simple mechanisms like Java servlets could become Web Services if SOAP messaging extensions were used. Existing distributed system applications can use wrappers to give them the appropriate SOAP interface and Web Service properties. The flexibility of implementation is one of the strengths of Web Services and is due to SOAP being an object model independent lightweight mechanism for messaging – an object model does not even need to be used by a communicating party as long as the SOAP interface is implemented [PL00M]

Because the Internet is made up of heterogenous technologies, one characteristic of the Internet is standards that use shared protocols over shared implementations. The three key Web Service standards at the time of writing are SOAP, WSDL and UDDI. In order to create a Web Service, these are the only standards that any implementation must observe. SOAP is the XML format for messaging, typically over HTTP. WSDL is an XML language for precisely describing Web Services’ programmatic interface. UDDI is a standard for business registries, a form of broker, that index Web Services.
Figure 2: [AR00I, figure 4] describes the role of the three standards. The XML Repository and the Application Server is one unit Web Service, and the Business Registry is another Web Service. The lines here do not describe communication but pointers to information held by the descendents of the line.

Simple Object Access Protocol (SOAP)
The SOAP specification [SOAP11] is in three parts. The first part describes the message structure itself using the concept of an envelope. This part of the specification defines the content of the message, who the message is meant for, and whether the message is mandatory in terms of whether the message is an optional or necessary part of series of message exchanges. The second part of the specification describes a way for encoding application-specific data-types, which can be exchanged after initial handshaking between Web Services. The third part of the specification defines a way of representing remote procedure calls and responses in messages. This part is used for invoking functions of a Web Service. SOAP is currently defined for use over HTTP, and is designed to be simple, extensible and independent of the application semantics or programming model used at either end of the connection.

Web Service Description Language (WSDL)
WSDL [WSDL1] describes the programmatic interface to a Web Service. The WSDL file of a Web Service defines precisely details like the protocol to be used, the port number, host number, the functions that the Web Service can perform, the input / output format of the functions and the exceptions that can be thrown. One of the main features of WSDL is its support for abstraction. Through the use of import facilities, it is possible to define a high-level Web Service description. This description can then be linked to lower-level descriptions that define more detailed bindings, such as specific port numbers, in a sense ‘binding’ the higher-level description to the lower level description. This allows different vendors to implement the same abstract WSDL description. The WSDL file itself is usually quite complicated and best left to tools to be automatically generated for the Web Service developed.

Universal Description, Discovery and Integration (UDDI)
UDDI [UDDIEW] specifies a distributed registry of Web Services. The registry contains descriptions of each Web Service in terms of the business or organisation it represents, the services offered by the
Web Service, the model of each service offered, called the tModel, describing the function abstractly, and a bindingTemplate that described the specifics of the tModel implemented. The listings in a registry are of three types:

- White Pages: listing Web Services by their address and contact information
- Yellow Pages: listing Web Services by the industry they are in, within geographical groups
- Green Pages: listing Web Services by technical information of the services offered, including references to the specification of the Web Service.

The companies that created it have also implemented the UDDI registry.

3.2.2 Implementation Styles

The potential success of Web Services will be due to industrial support, and the leading pioneers in this area have all defined ways of building Web Services using products affiliated to them. These methods and technologies define a more structured approach to building Web Services than just using a servlet with the appropriately enabled technologies.

Hewlett Packard has for the last few years been developing a technology called E-Speak. E-Speak comes in two parts: the E-Speak Engine and Service Framework Specification (SFS). Used in combination, these two parts enable the building of E-Services (a synonym for Web Services).

![Figure 3: HP00H-3, slide 13](image)

Figure 3: HP00H-3, slide 13 gives the tiered architecture of E-Speak.

The SFS sits at the top level of the E-Speak architecture and defines conventions for business exchanges, including transactions like negotiation, with other Web Services. This level of detail in messaging is not defined in the general architecture of Web Services, and is unique to E-Speak. The SFS is analogous to Microsoft’s Biztalk. The SFS sits on top of the SOAP ‘document exchange model’, which is used for messaging between Web Services, as is usually the case. However, it is also possible for E-Speak Web Services to communicate using a network object model, in the style of traditional distributed computing systems. Though this is only recommended for intranets, this allows for faster communication as the overhead of XML parsing and translation is not present. The E-Speak Engine sits at the bottom of the architecture and represents the actual Web Service.
Microsoft announced its .NET strategy on 22nd June, making SOAP and Web Services a central part of its plan. It is important to realise that .NET is a software development model rather than anything else. In the .NET framework, programming code can be written in almost any language, including Microsoft’s C#, and targeted for deployment on a variety of mediums.

These mediums include Win Forms, which are the traditional Windows based applications, as well as Web Forms and Web Services. A Web Form is analogous to a web site powered by EJBs or other programming, as in sites like Yahoo!, and creates products of the Application Web (as defined earlier). In some senses Web Forms can be thought of as Web Services with a web page interface for users, as opposed to a programmatic SOAP interface for other Web Services. The crucial point is that all the deployment models produce self describing components that are not directly dependent on other components, with the only main difference being that they are targeted for different environments: the desktop, Application Web and Services Web. Microsoft has a suite of products to help developers build and deploy products to these mediums. Web Services in this model are analogous to the Engine in the E-Speak model.

IBM has taken the approach of facilitating the evolvement of Web Services without attaching initially products to the idea. Recently, IBM products like VisualAge and WebSphere, tools for creating Web Applications and other products, have started support for creation of Web Services. IBM has developed a lot of the ideas and thinking behind Web Services, and has played key roles in developing standards like UDDI, SOAP and WSDL. IBM also currently offers a suite of freely available test tools for Web Service development from AlphaWorks, while also showing how Web Applications can be created using entirely open source servlets, XML parsers and SOAP software [GG00I2].

Sun has joined the Web Services initiative with the Sun ONE Architecture for creating and deploying Web Services. Sun ONE is essentially a method of Web Service development centred on Java.
The ONE architecture describes micro services that are developer written or pre-built components that are then composed together to create macro services during the Service Creation & Assembly stage. These macro services are consequently deployed as Web Services & Applications, at which stage security, management and related issues are considered. A Web Service will typically reside on a Service Container, a run-time environment such as an application server. Service Integration facilities give the Web service access to other Web Services, databases, legacy applications and the like. Service Delivery provides an interface for users and other Web Services. The entire environment runs on a service platform, for example a mainframe or even a smart card, which provides access to local operating system services such as databases and directories.

### 3.3 Workflow Management System Considerations

#### 3.3.1 WFMS Reference Model

Workflow Management Systems (WFMS) generally have similar structures, irrespective of the application domain or type. The general pattern used in WFMS systems is characterized by the Workflow Management Coalition’s Workflow Reference Model [WREF]. This Reference model looks at a general scenario of how workflow is used, as is described in Figure 6.
According to [WREF, terminology adapted] the major roles in the WFMS are:

- **Workflow Specification Tool**, similar to the one described in this thesis, which is used to create the definition of how the workflow will act.

- **The Workflow Specification**, essentially the schema used to describe the workflow.

- **Workflow Execution Environment**, like that described in this thesis, which interprets the workflow specification schema, invoking and coordinating as necessary constituent tasks in the schema.

- **Run Time Software**, like the Execution Environment and Monitoring Tool in the project, that actually execute the workflow and monitor its progress.

- **Workflow Relevant Data** is data that the workflow schema uses to navigate execution in the workflow. These include data derived from applications. In the case of the project, this is largely the output of Web Services.

- **Monitoring Facilities** include a tool like the project’s Monitoring tool allows for the supervision of the workflow during execution.

- **Other Roles**: The Reference Model also considers Worklists and associated software. Worklists are essentially lists of tasks for the attention of humans involved in workflow. These are not relevant for the project.
The Reference Model exists largely to define interfaces between the different roles described. In doing so, different parts of different WFMS systems implementing these interfaces, and produced by different vendors, should be able to interoperate.

Figure 7: The Reference Model, as seen here [WREF2]

The Workflow Enactment Service (Execution Environment) consists of one or more workflow engines that are responsible for the execution of some set of tasks in the overall workflow. The Enactment Service itself is largely a unified front for these engines. Individual workflow engines can be homogeneous or heterogeneous, though in the latter case there need some mechanism for information interchange.

[WREF3] gives an outline of each of the interfaces in the model that is summarized here.

- **Interface 1** gives a definition of a standard interface between the workflow schema and modeling tools and the workflow engine(s).

- **Interface 2** defines APIs for client applications to request services from the workflow engine to control the progression of processes, activities and work-items.

- **Interface 3** gives a standard interface definition of APIs to allow the workflow engine to invoke a variety of applications, for example Web Services, through common agent software.

- **Interface 4** enables workflow interoperability, giving models and the corresponding standards to support inter-working of workflow engines.

- **Interface 5** gives the definition of monitoring and control functions.
Because the Reference Model was designed with business process automation in mind, in the case of this project, not all of the details are relevant to the Web Services domain, and interoperability with existing workflow products in other domains is neither a goal or considered possibility. However, the way the Reference Model works can offer hints at how to proceed creating the project WFMS, as the general model, ideas and principles are similar.

### 3.4 Enterprise Workflow

#### 3.4.1 Distributed Computing Middleware

The various areas of distributed computing middleware technologies fall largely into the following areas [CMU-M]:

- **Transaction Processing (TP) Monitors** emerged 25 years ago, and provides the distributed client/server environment the capacity to efficiently and reliably develop, run, and manage transaction applications from a single server to thousands of clients.

- **Remote Procedure Call (RPCs)** allow for application programs distributed over a network to be invoked as though they were executed as a local routine.

- **Message-Oriented Middleware (MOM)** enables distributed applications by program-to-program message exchange. [CMU-M] describes MOM as ‘analogous to email in the sense it is asynchronous and requires the recipients of messages to interpret their meaning and to take appropriate action’.

- **Object Request Brokers (ORBs)** enable the objects that comprise the applications to be distributed and shared across heterogeneous networks.

Most Workflow Management Systems fall into the last three areas.

#### 3.4.2 Within Businesses, Application To Application (A2A)

Traditional workflow software that derived from business automation origins had largely been the venture of a specialist group in the MOM area. Most of this software was designed to support automation of business processes, such as ordering stationary. In recent years, the emergence of the requirement of being able to integrate different applications has resulted in increased interest in this sort of software. Enterprise Application Integration (EAI) is the process of linking together various different applications to allow for the transfer of information in an organisation. Almost all traditional business automation workflow software have evolved to become EAI software. However, there are instances of EAI software that do not account for interaction with staff, in which case these are not the same as traditional process automation software. EAI is designed to bring about communication between different applications designed to handle specific business functions. Although point-to-point programming between one application and another application is possible, [JMIBM] describes how EAI uses workflow, where communication between applications is described at a flow level. TSI’s Mercator Enterprise, IBM’s MQSeries and Sun Forte’s Fusion are examples of EAI.
3.4.3 Between Businesses (B2B, B2C)

Besides workflow within organisations, there is also been technology for supporting workflow between organisations, evolving also largely in the MOM area. [ARMK] describes the different forms of these technologies and how they have evolved:

- **Electronic Data Interchange (EDI) and Enterprise Requirements Planning (ERP)** have been used by larger companies for decades. These expensive mechanisms are essentially direct connections from one company to another, where there is a well-defined trading relationship, and were originally introduced to remove the use of forms. Instead of typing out information to an electronic form, printing it, sending it off, only for it to be transcribed again into another electronic form, these systems used a direction connection from one system to another.

- **Sell Side Storefront, Business To Consumer (B2C)** is essentially the Application Web defined previously. These introduced shopping and trade over the Internet. However, the buyer was still left responsible for comparing different Internet ‘stores’ for prices, and so made this expensive for corporate procurement organisations. Packages like Weblogic and WebSphere readily build these kinds of ‘Sell Side Storefronts’.

- **Buy-Side eProcurement, Business To Business (B2B)** allows corporate procurement organisations to aggregate many supplier catalogues into a single “universal” catalogue, from where a user sitting at a desktop can choose between different suppliers offering products. This mechanism allows for quicker comparison of prices and details, and some also wrap around legacy ERP systems. Applications in this area include Ariba CSN and Commerce One’s Commerce One.net.

The use of workflow techniques is most advanced in Buy-Side eProcurement software.

3.4.4 How These Systems Work

Though there are differences between B2B and A2A systems, the distinction is not huge, and the same principals apply. [IBMWFA] describes how a suite of products that provides all these different types of enterprise applications can be used together using a common workflow.
In such a system as that given, the workflow engine will typically sit on an application server, working with supporting software and workflow clients over a network, to create a Workflow Management System [KT96B]. The business process descriptions representing the flow, which directs the workflow engine, normally resides in a database. This database repository will also hold information about transactions and instantiations of the workflow.

[KT96B] describes two types of workflow engines:
- **Active** workflow engines that monitor each task that is performed. When a client or supporting software completes a task, the result is returned to the engine, which will use this to start the next task.

- **Passive** workflow engines do not monitor all tasks. This means each client or supporting software performing a task is expected to pass the result of its work to the next task.

Active engines can start and stop workflow tasks at any time, and hence are able to interrupt and handle any unexpected results in the workflow. However, if the engine fails, the whole system stops. This is not the case with Passive engines. The conceptual model of the workflow entities and mechanisms is normally the same as that described in WFMC standards.

### 3.4.5 Significance For Web Service Workflow
Workflow within and between businesses cannot be used to orchestrate Web Services. The main reason for this is because Web Services do not immediately fit into the components that make up the entities used. Web Services additionally also have properties like reflection that are not readily considered in these systems. Most significantly, Web Service workflow does not need to consider interaction with staff, as they are implicitly represented through the Web Services they are using. However, the techniques and ideas used in these systems can facilitate discovering how the orchestration of Web Services is possible, as in terms of EAI, Web Services can almost be thought of as self-describing applications.

### 3.5 Candidate Workflow Systems Of Interest

#### 3.5.1 OTSArjuna, University of Newcastle upon Tyne [WSRC], [W1SRC], [W2SRC]
OTSArjuna is a WFMS for implementing workflow based on CORBA. OTSArjuna uses a graph like notation to represent workflow. The graph is made up of nodes called tasks, representing a unit of computation to be completed by some component. Each task has a group of input and output sets. Each input set describes individual inputs it must receive from other tasks it relies on. Each output set describes individual output it will send to tasks relying on those. Consequently, the different sets describe the different input and output combinations that are possible.
Figure 10: Task t1 is a compound task containing tasks t2 and t3. When all the inputs for an input set are complete, the task will process these and send output from an appropriate output set to tasks dependent on it. [WSRC, figure4]

At runtime, a Workflow Repository service holds schemas of different workflows. These are executed by a Workflow Execution service, which co-ordinates the workflow. The Execution service has a user interface for administering the workflow at runtime. Actual responsibility for executing and managing tasks are delegated to a Task Controller associated with each task, decentralizing the management of the workflow.

3.5.2 METEOR2, University of Georgia [MET], [WXG]

The description of workflow used by this system is different from that used by OTSArjuna, in that a more declarative language approach is taken. Each task is associated with a directed graph representing the states into which the task can change to. The changes from one state to another arise through inter-dependencies to other tasks. These inter-dependencies are described using Workflow Intermediate Language (WIL), which is specific for METEOR2. This inter-dependency description can describe both control and data dependencies, and can be generated from a GUI application.
Figure 11: This diagram shows some simple tasks, which can be transactional (ACID supported) and non-transactional (ACID not supported). [WXG, figure 3.1]

METEOR\textsubscript{2} includes a GUI based workflow designer based on WIL and a code generator to create complied run-time components from the WIL descriptions. There is also a Workflow Repository to hold template workflows and the like. A Workflow Enactment System holds several workflow engines. Each engine will hold several complied WIL components. Task Managers and Schedulers co-ordinate the execution of these complied WIL components.

3.5.3 RainMan, IBM T. J. Watson Research Center [PPCI]

The creators of RainMan argue against centralized Workflow Management Systems and consider some novel but real use cases for their system that will be typical for an Internet environment. These include the ability to download and run workflow schema, dynamic reconfiguration of workflow at runtime and a use case concerned with accounting for devices that can go offline but be assigned to tasks.

The workflow in RainMan has Performers and Sources. Performers perform tasks while Sources hold activities, which are essentially different workflow schemas. Each activity is made up of a number of tasks. Sources request Performers to complete a task. A Performer will have a task-list of different tasks it has been asked to perform. The RainMan system defines interfaces for Performers and Sources, so these can be implemented in different ways.
RainMan has been implemented as a RainMan Builder, which is essentially an applet for creating workflow activity specifications that can also act as a Source and monitor the completion of the workflow activity. Though RainMan has been implemented using Java RMI rather than CORBA, the transport has been designed to be replaceable, so it can be re-implemented on top of any messaging system.

3.5.4 Web Service Flow Language (WSFL), IBM Software Group [WSFL01]

In May 2001, IBM released a discussion paper on WSFL, a workflow language for Web Services. There is no implementation as yet, but the paper provides sufficient functional details. WSFL has two models of ‘flow’, Flow Model and Global Model. The former is concerned with describing workflow between several parties. The latter describes interfaces for Web Services and patterns of interaction between these different interfaces, essentially making explicit interactions between business parties.

Conceptually, a Flow Model is made of Control Links between Activities. Activities represent a business task within a business process, and can be interpreted as a method call within a conversation of methods calls in the business process. Control Links can have Transition Conditions, a Boolean expression that acts like a guard. Data Links are superimposed over Control Links and must respect Control Links.
3.5.5 Significance For Web Service Workflow

Looking at the WFMS solutions produced in this domain, it is possible to see that a variety of strategies are possible in terms of a WFMS for Web Services. These current strategies largely describe workflow specifications in terms of graphs, particularly WSFL. WSFL was released too late to influence this thesis, but includes the idea of deploying workflow as Web Services. The workflow model is not particularly different to that of other approaches.

With respect to the other systems, RainMan is possibly the most relevant for Web Services, partly because of the use cases the system considers. While the general principals learnt from these systems are helpful there are however significant differences with respect to Web Service workflow. The first distinct difference is that Web Services use XML messages as opposed to objects. Additionally, Web Services are self-describing and can change dependencies, so consider the ad-hoc creation and change of workflow.

3.6 Other Systems

3.6.1 jointFlow / Workflow Management Facility (WMF), OMG [OMGWM], [MTS]

The Workflow Management Facility specification for CORBA accepted by OMG is jointFlow, a submission by 19 companies describing a set of interfaces that can be implemented to create WFMS systems built on parts that can interoperate with other WFMS systems.

The interfaces include WfRequesters that have WfProcesses, representing a workflow schema, they would like completed. WfProcesses hold a set of WfActivity, each of which is a task that needs to be completed. A WfActivity task is associated with a WfResource that can complete that task. This relationship is represented as a WfAssignment. There are also interfaces to monitor events and create WfProcesses.

![Figure 14: Note that WfActivity can have a WfProcess that can be a different sub-workflow [MTS, figure 3]](image)

There are as yet no known implementations of these interfaces.
3.6.2 Biztalk SOAP Orchestration [BIZ]

In September 2000, Microsoft released a beta version of BizTalk Server 2000 Enterprise Edition, which includes BizTalk Orchestration, which provides the means for coordinating applications and components it supports in a workflow. One of the supported components is SOAP. At the basic level BizTalk Orchestration similar deliverables to that described in the Introduction (Chapter 2). However, BizTalk Orchestration is lacking in the advanced aspects. The language supports concurrent tasks, synchronization and dynamic assignment of a task to a component. Significantly, functionality such as choice is not provided in the workflow, meaning an absence of alternatives. In addition to this, BizTalk Orchestration concentrates on largely enterprise level workflow between components, and in this case does not consider some of the more general use cases of Web Service workflow. During execution of the workflow, monitoring is limited to simply querying the state of each component.

![Figure 15: Biztalk Orchestration splits the view into two parts, the left representing the abstract workflow, and the right representing how it maps to the actual real components [BIZ, figure 2]](image)

The system offers a view on one simple approach to the Web Service workflow. However, since this is a very constrained system, it may not be an approach that can solve other problems considered by this thesis, such as recursive encapsulation of workflow.

3.6.3 Architecture Description Languages (ADLs)

Architecture Description Languages describe the composition of large distributed systems. In this sense they are similar to describing workflow, and could be considered a possible candidate for driving workflow engines and systems. Both ADL and workflow languages can describe dependencies between different components in a distributed system. However, as [CDMIT] notes, ADLs focus on the architectural structure of a distributed system whereas workflow is concerned with computational and informational flow in a system. In this sense ADLs are analogous to maps of components, whereas workflow languages describe the routes computation and information take through these maps.

3.6.4 Hewlett-Packard Industrial Placement (‘Wrapping E-Services In Cocoon’)

During my industrial placement with Hewlett Packard Research Labs, I was involved in a project involving E-Services. This project considered building a web site for online business, such as Amazon.
The front end of the web site interacted with users, producing dynamic content, while the backend of the web site interacted with E-Services, such as banks and warehouses.

The technology used was based on the Apache Internet Publishing Framework called Cocoon. Cocoon uses a publishing workflow to create the content returned to the user dynamically. The workflow consists of different processors that add and format the document that is eventually returned to the user. The project at HP looked at adapting Cocoon so that some of the processors in this publishing workflow could represent Web Service, which could process the document remotely.

While initially there may seem to be some aspects of the HP project relevant for this thesis, this is not the case. The version of E-Speak used for E-Service interaction was based on a distributed computing model called The E-Speak Service Bus, which was deprecated at the end of 2000 in favour of the SOAP model of Web Services launched by Microsoft in mid-2000.

The publishing workflow used by Cocoon is also linear, in that there is only one document / data item in the workflow which is successively modified by different processors; concurrency and synchronisation, major considerations of this thesis, were not issues. The singular and linear data flow of the publishing workflow also does not map to that of multiple streams considered by this thesis. An important assumption of the HP project was that E-Services were aware of their role in the workflow; this assumption does not hold true in the general case considered by this thesis. There is also inherently a significant difference to workflow that ultimately produces web pages, and that which is ultimately deployed as a Web Service, in terms of the requirements of the respective systems.

### 3.7 Interoperation of Web Services

In this part we look at how Web Services interoperate, to give some background to considerations of workflow between Web Services.

According to [IBM001], in a networked environment of Web Services, each Web Service will typically play one of three roles. These are:

- **Service Provider:** The Web Service providing some useful functionality that can be invoked by other Web Services.

- **Service Requester:** The Web Service seeking another Web Service to perform some required functionality or work that needs completion.

- **Service Broker:** The Web Service that will perform the process of matching up Service Providers and Service Requester.

The relationship of these three types of Web Services is summed in the following diagram:
The Provider will be created, with the appropriate WSDL definition, and publish its presence with a Broker. Alternatively, Brokers may pro-actively search for Providers to list in their Web Service. A Requester will typically use a Broker to find a desired Provider, unless the Requester already knows the Provider it wants to use. Once the location of the Provider is found, the Requester will then bind with the Provider to allow for the appropriate collaborations. This is the typical pattern for brokering in distributed computing.

Once a Provider has been found, [SB00I] defines three kinds of binding the Requester can have with the Provider, relating to how the Provider was found:

- **Hardwired binding at design time**: The Requester knows which Provider to use, and other exact details, like how to communicate with it, as the Requester has this information coded into it at design time.

- **Dynamic binding to a static choice of collaborator**: The Requester is hard coded with a specific query to use with a Broker that returns a specific determinable Provider. The consequent interaction with the Provider is dynamic, as security protocols, business domain protocols and other information are decided at run time.

- **Dynamic choice of collaborator**: The Service Requester has knowledge of the desired API and semantic requirements of the Provider. The Requester uses a search pattern with a Broker that returns a set of alternative Providers. The choice of Provider used is determined at runtime.

In situations where there is some level of dynamic binding involved through a Broker, Burbeck et al [SBSG00I] go to some depth to explain the importance of the brokering mechanism used by the Broker. It is not sufficient for a Requester to search for all Providers that implement some API function, as some Providers may not use the same function name, and even after search results are procured, there may be several other requirements that do not match up with what the Search Requester desires, such as Provider’s cost or location. Instead, [SBSG00I] suggest that Yahoo!-style categorisation of Web Services into taxonomies of hierarchies is required. Requesters then can traverse the hierarchy, finding more specific groups of Web Services that match the Requester’s requirements. The usefulness of a Broker to a Requester will be determined by how relevant the traversal of hierarchy structure is to
the business need the Requester needs to fulfil. There is also great importance in the description of the Web Service held by the Broker, and whether it contains further relevant information.

A Provider will not maintain state information on the state of Requesters using it, as the Requester holds control of Provider Resources, through database locks for example. According to [MS00M], this means a Web Service's functions are:

**Stateless.** All the information required to perform the service is either passed in with the request message or can be retrieved from a data store based on some information provided with the request.

**Atomic.** Each service represents a complete unit of work that leaves data stores in a consistent state. For example, if clients need to be able to move money between bank accounts, the service should accept a MoveMoney request message, not just Debit and Credit requests.

It is important to note here that the Provider itself can also be a Requester, in which case it will maintain state information only for itself as a Requester.

### 3.8 Web Service Issues

There are several issues surrounding Web Services that pose challenges to this project. The reason for this is largely due to the Web Services phenomenon still being very young and undeveloped.

Perhaps the most important issue to consider is Security, since SOAP does not provide any provision for this, and insecure workflow prohibits the number of applications it can be used for. [IBM00I] have identified the following areas of threats with regard to Web Services:

- *The security of information that is shared between the broker, the requester, and the provider at runtime.*
- *The security of the network in which the runtime is deployed.*
- *The security of the programming model (APIs, skeletons, stubs, and so on) at design time.*

[PMS00M] gives some counter-measures direction on this. At the basic level, security for a Web Service is analogous to security for a web page, as the underlying technology of HTTP is the same. In this case, Secure Socket Layer (SSL) could be used. Authentication could be provided using specific password – user style information in the message sent to a Web Service.

Other issues identified in [GG00I] include manageability and accountability of Web Services. Work needs to be done to look at how Web Services and collections of Web Services, perhaps in a workflow, can be managed to ensure details like quality of service. The payment model for using a Web Service will also need to be researched, in terms of subscription versus pay-as-you-go, although this may vary with the application domain. It is also necessary to look at the legal contracts that Web Services are agreeing to fulfil by supplying some service or function.
Dependability of Web Services will be another issue, since the performance of some Web Services will be better than others. Communicating this form of information will also be difficult. As in some cases, the idea of Web Services is to use other Web Services to perform some function, such as perhaps credit card validation, or even a shopping basket, there has to be some level of guarantee in these lower level Web Services performing as required, or a way for a Web Service to turn to other backup alternatives.

Another difficulty within a Web Service environment will be testing, as if there is a bug, it may be difficult to track down where the bug is, especially since all Web Services are black box components. When a Web Service crashes or there is some other problem, a Web Service may throw an exception that will need to be handled by the calling Web Service. In this case, these exceptions will need to be catalogued and be will need to be identifiable [SB00I].
4 Overture

4.1 Chapter Summary

- The general architecture of the Workflow Management System is described and justified.

- A summary System Map is given, allowing the reader to cross-reference between different chapter sections and understand the different parts of the Workflow Management System.

4.2 The Approach

4.2.1 Tools

SOAP Implementation. The Workflow Management System will use an underlying SOAP implementation, which means that the lower-level procedures of communicating using the protocol do not need to be considered in great detail. There are now several alternative implementations available, though at the start of the project the two main implementations were those belonging to Apache and Microsoft. None of the implementations are ‘better’ than the other, with all of them having deficiencies and strengths, in terms of usability, dependability and functionality of how and what they implement of the SOAP standard. The main concern with all of them, that there are some interoperability problems between them due to the interpretation of the SOAP standard. These differences may be resolved.

Apache SOAP is an Open Source implementation created using Java as part of Apache’s XML Project. The SOAP Project has a lot of support, and being vendor neutral, is generally taken as a more ‘standard’ implementation than the Microsoft alternative. Microsoft SOAP Toolkit is partly tied into Microsoft’s .NET strategy for creating Web Services. While this implementation does provide a good way of creating Microsoft style Web Services, it also requires the use of Microsoft specific components and technology, not all of which are related to SOAP.

Given this choice, the Apache implementation was used for the purposes of this project because it is simple, open, vendor neutral and it fits with the other technologies used.

Hosting Server And Programming Language. Although there is a vast choice of different servers available, there is not a great deal of difference in terms of the influence it will have on the project. Since this is the case, Apache’s Tomcat is used, because it is a lightweight server with strong support for Java Servlets; the Apache SOAP implementation is essentially a Servlet.

To support platform independence, particularly in the creation and use of the workflow, Java is also an appropriate language on which the Workflow Management System can be developed.
4.2.2 Techniques
The general principals behind Refactoring [MF] and Problem Frames [MJ] are used throughout as guidelines rather than strict rules. Use of the former is not described in great detail, as the descriptions of the changes would be tedious, but the residual design effects and motivations are clear. The latter is used as an aid to the design process, in finding initial solutions, as well as improving solutions found. Recognised Design Patterns [GHJV] are used as possible strategies for solving problems.

4.2.3 Design Philosophies
Certain design philosophies are followed throughout the design and implementation of the system.

Encapsulation of Concurrency within a single object. All new threads of execution are encapsulated within their own objects. This allows for cohesion of functionality and execution, in a form similar to Active objects, improving understandability of the execution paths of the threads, as well as extendibility of the functions of the encapsulating object.

Separation of data and functionality. Data and functionality are always defined separately, with objects encapsulating functionality acting on objects encapsulating data. This allows for both to be extended with less difficulty than otherwise. Ensuring highly cohesive objects encapsulating functionality also allows for code reuse more effectively, as objects are not accessing other objects logically not related to them, they are instead accessing objects encapsulating functionality that is related to them.

Extensibility of data structures and functionality. Building on the separation of data and functionality, a running theme is the extensibility of both beyond simple inheritance or other object-orientated mechanisms; devices are built into objects to allow for extensibility, be it by double dispatching or otherwise. The General Architecture, described in the next section, which essentially allows method calls to be added dynamically to the Web Service created using workflow, exemplifies this.

4.3 General Architecture

4.3.1 Aims

- The WFMS will be able to create workflows that can be used by other WFMS systems. Other WFMS systems should be able to invoke this WFMS to enact workflow created with this system.

- The Execution Environment will allow for the concurrent execution of more than one instance of a workflow schema that has been deployed.

- The WFMS will allow for the execution of different workflow schema to execute concurrently, using the same Web Services.

- The Execution Environment will support the deployment of workflow schema defined with the Specification Tool. The Execution Environment will support the persistence of workflow schema once they have been created and deployed. This requirement ensures workflow schema can be maintained in some repository or otherwise for future use.
• The User of the WFMS should be able to invoke its use it from any computing platform. This requirement allows the ability of using workflow from almost anywhere.

4.3.2 Derivation And Separation Of Concerns
Component Based Programming has made many advances in these areas through component encapsulation. With Web Services being Components, it is possible to apply the idea of workflow as encapsulated Components:

• The workflow should be encapsulated in a Web Service that enacts the workflow, the WorkflowEngine.

• The workflow should be able to be invoked as a black box service, hence be able to be used from any SOAP client. SOAP clients exist for Java and many other computing platforms. The SOAP client could also itself be a WorkflowEngine that treats the WorkflowEngine enacting the workflow in the same as any other Web Service.

• The workflow should be able to be invoked several times and concurrently with other WorkflowEngine Web Services encapsulating workflow.

Through these critical design decisions it is possible to satisfy the given aims.

4.3.3 Structuring The System
The system derived thus far interacts with Web Services in the workflow and is itself a Web Service. However, this system still needs a way to interact with the User who creates and observes the workflow. There are two main possibilities:

1. The WorkflowEngine has a user-interface web page portal through which workflow can be managed.

2. There is a User-side application (henceforth called ‘User Application’) that interacts with the Workflow Management System as necessary.

With the first option there is difficulty in creating a HTML or similar based system that can sufficiently present and be used to manipulate complex information, as is the case with workflow. The second option also provides benefits in general design terms, of de-coupling the user-interaction processing from the workflow enactment. In essence, this means that different User Applications can be built, perhaps to suit different needs. These different User Applications can interact with the same Workflow Web Service, so long as the same protocol is used between them. There are also benefits in changing the WorkflowEngine without affecting the User-side application.

In consequence to this, the system can be split into two parts, one that is used to create and monitor the workflow, interacting with the User, and one that enacts the workflow, interacting with the different Web Services in the workflow. It is important to note here that the WorkflowEngine that enacts the workflow and the User Application that creates and monitors the workflow are both in combination the Workflow Management System.
4.3.4 The WorkflowEngine Web Service

As the WorkflowEngine, which enacts the workflow, is itself a Web Service, it provides the mechanism by which it is possible to recursively encapsulate workflow within other workflow. This means that in the same way workflow that has been deployed on the WorkflowEngine Web Service call other Web Services, the WorkflowEngine Web Service itself could be called by the workflow, invoking other workflow that has been deployed on the WorkflowEngine Web Service.

The ability to invoke Web Services encapsulating workflow from another workflow provides a means for hierarchical composition of the workflow. Each encapsulating Web Service hides complexity and unites highly cohesive parts of a larger workflow. A complicated business workflow managing interactions between businesses, such as the order, billing and dispatching of a book purchased online, could be composed of Web Services encapsulating workflow within each business, which at the highest level the workflow designer does not need to be concerned with. Other benefits include separation of design and construction of each component workflow, and the ability to produce standard interfaces between these workflow components - Web Service method calls.

4.3.5 Protocol Between User Application And Workflow Web Service

In order to maintain high cohesion and low coupling between the WorkflowEngine and User Application, it is necessary for the protocol between them to be as uniform as that between the WorkflowEngine and any other Web Service that may wish to use it. The services provided by the WorkflowEngine to the User Application must be open and available to other clients and Web Services.

The effect is that the User Application must invoke methods on the WorkflowEngine Web Service using standard SOAP RPC to perform the necessary actions of:

- Setting a workflow schema on the WorkflowEngine so it can enact it (i.e. workflow deployment).
- Invoking the relevant workflow on the WorkflowEngine.
- Obtaining necessary information on the progress of the workflow enactment.
• Performing other operations as necessary, such as un-deploying the workflow.

An important factor between the WorkflowEngine and any of its clients, such as the User Application, will be the language that is used to describe the workflow. The language will be a common standard between Workflow Web Service and User Application.
Figure 18: System Map without frames
Figure 19: Frame numbers correlate to Chapter Sections numbers
5 Shared Mechanisms

5.1 Chapter Summary

- The workflow language is introduced, with its respective control-flow and recursive encapsulation centric SafeFlow dialect and data-flow centric CircusFlow dialect.

- Probabilistic Method Calls are introduced as a non-systematic automated choice mechanism that accounts for probability distributions.

- Data Binding, a process of converting Java Beans to XML and vice-versa that is independent of the Bean and XML definitions, allowing the definitions to be extended, is implemented and described.

5.2 The Workflow Language In Outline

5.2.1 Aims

- To define a language that can consequently be used on both the Workflow Engine and User Application with minimal modification for the context of use.

- To allow for basic expected functionality: sequencing of Web Services in order, parallel execution of Web Services, synchronisation and choice mechanisms.

- To enable the language to be extended with minimal complication to the User Application and Workflow Engine, while maintaining backward compatibility.

5.2.2 Derivation

A competent starting point for any form of language is the Interpreter Pattern [GHJV], which defines a general form. However, the normal implementation of the pattern will define an Abstract Expression at the highest level of the hierarchy, descending into Terminal and Non-Terminal Expressions. The top-level expression of the language (such as ‘class’ or ‘module’) will usually descend from the Non-Terminal, but in this implementation the top-level node, called a WorkflowService, also extends from the Abstract Expression. This design allows for the implementation, with less bureaucracy, of more than one language-type, or dialect, that can share some of the expressions within the Terminal and Non-Terminal descendants. While different dialects may have common top-level features, they may not necessarily share features common to Non-Terminal expressions of all dialects.

Specifically, there are two different descendents of the WorkflowService.
• **SafeFlow** provides a structured approach to designing workflow, with tight control over execution.

• **CircusFlow** is an experimental dialect concerned with information flow rather than execution.

Each of these dialects behaves, and is designed, differently. WorkflowService acts as an interface or marker class for both, rather than describing specific functionality.

In a normal implementation of the Interpreter Pattern, Non-Terminals usually are expressions like loops. Applied to the context of workflow, they allow for a finer control grain of sequential and concurrent actions instead. The Non-Terminals include, as well as Concurrent and Sequence, FailureSequence and Bayesian. The former is a mechanism for catching failures in Web Services, while the later is a complex choice mechanism based on probabilistic inference. The Terminals are either Sync, for synchronisation, or Web Services. Web Services can be statically defined at design time or dynamically defined at run time.

Although the nature of the entire superset of the expressions means that any Terminal or Non-Terminal can be used with SafeFlow and CircusFlow, the distinct design method used for each presented by the User Application restricts both WorkflowServices to a subset of what is available, in order to preserve the intent of the method of workflow design for each. The enactment of SafeFlow and CircusFlow schemas, by their respective interpreters on the Workflow Engine, is different, particularly in terms of concurrency.
5.2.3 From Class Diagrams To Java Beans And XML

Central to the implementation of the language is the tight binding of the language between the representations at the WorkflowEngine and User Application, as well as the transport between them. In terms of internal representation, the language is based on Java Beans that encapsulate data only. Decorators and visitors can add functionality to the Beans as required by the program using the Beans; all programs using the language use the same set of Beans.

The transport used between the WorkflowEngine and User Application is an XML representation that has a direct one-to-one mapping with the Bean representation. This tight mapping enables the conversion process between Beans and XML to be independent of the Beans and XML themselves, describing only how to map any of the language’s Bean to an equivalent XML form and vice versa. This enables extension of the language by simple addition of Beans, without modification of the mapping to and from the transport. This is in essence a custom and specialised implementation of Sun’s ambitions for Java-XML Data Binding [SPA].

More specifically, the Data Binding process describes how different parts of a Java Bean map to different XML equivalents. A Java Bean is mapped to an XML element with the same name, and a field in a Java Bean is mapped to an XML attribute, with the field name, within the appropriate element. During conversion to and from Java Beans to XML, this direct mapping is adhered to, creating the necessary instance of a Bean or XML element. Consequently, the introduction of new Beans is not problematic, as it is automatically mapped to a new XML element of the same name. The Data Binding process is described in further detail in section 5.8.
5.2.4 Related Work
None of the current available ORB workflow languages use the structured Interpreter Pattern style approach to their design, which is largely exploited in SafeFlow. Consequently, the significant distinction between other languages is that of style. The SafeFlow language presented here is programmatic and systematic, while other languages are orientated more towards scripting. OTSArjuna and WSFL partly compensate for this through the possibility of composition, but control is still unstructured. The CircusFlow language is again different from existing languages and SafeFlow in disregarding control flow over information flow. A more in depth comparison is presented in the Evaluation. Notes on other languages can be found in the Background, section 3.5.

5.3 The Abstract Expression

5.3.1 Aim
To provide inherent characteristics and a language architecture common to all expressions in any workflow.

5.3.2 The AbstractExpression Bean

The mechanism of interaction between one expression and another is through:

- **ImportParameters** – the values that an expression requires to execute; these values can come from ExportParameters of neighbour expressions or ImportParameters of a parent expression.

Figure 21: The AbstractExpression and its related field Beans. The Beans implement Drawable to allow them to be decorated. This facility is used in the User Application and described there.
• **ExportParameters** – the values that an expression exports, available for other expressions to import

The import and export data for each expression is represented through respective lists of EParameter instances held by the AbstractExpression Bean. An EParameter extends the Parameter Bean defined in the Apache SOAP implementation. The Parameter holds four fields representing the parameter Name, Value, Type and XML Namespace defining the Type. The EParameter adds the Reference field, which describes which neighbouring or other expression the Value for this parameter can be derived from. Only one of the Reference or Value field will usually have an assignment.

**Guards.** In addition to lists of ImportParameters and ExportParameters, the AbstractExpression also maintain a list of Guards. Guards ensure that an expression will only be allowed to execute if the conditions of all the Guards are fulfilled, and is a mechanism of preventing interaction with another expression where required. Each Guard is associated with an ImportParameter, and each ImportParameter can have many Guards. Three fields define the Guard Bean itself: the name of the ImportParameter to test, the value to test against and the type of test. Four test types are supported: ‘equals’, ‘not equals’, ‘less than’ and ‘greater than’. With data types that are not obviously numerical, ‘less than’ and ‘greater than’ obey lexical order, as per the Comparable Interface implemented by Java’s native data types.

As an example, if the an ImportParameter exists that takes an integer, and the ImportParameter has a Guard specifying the test type is ‘less than’, with test value 6, only if the ImportParameter is satisfied with data less than 6 will the expression be allowed to execute. If the ImportParameter takes a string instead, and the test is ‘less than’ with test value ‘banana’, only data lexically before ‘banana’ can be accepted, such as ‘africa’.

### 5.4 The SafeFlow Dialect – SafeFlowService

**Expressions Normally Used With:** Concurrent, Sequence, FailureSeq, Bayesian, WSSstatic, WSDynamic

#### 5.4.1 Aims

• To support a structured and reasoned approach to workflow design.

• To provide for strict management of the workflow’s control flow.

#### 5.4.2 Derivation

The distinct benefit of object-orientated programming is encapsulation of complexity within an object. A program can be made of many objects, with each representing a specific type of function and data. Access methods are provided to make use of what an object encapsulates. Components, like Web Services, are similar to heavy weight objects. Hence, recursive encapsulation of expressions can provide an elegant way to recursively encapsulate complexity, allowing for bigger problems to be decomposed into smaller ones.
Expressions, particularly Non-Terminals, can hide all their internal workings from all other external expressions. The AbstractExpression, which all expressions descend from, provides the common access method style ways for one expression to interact with any other expression, without knowing what the expression is, in a black-box way. WorkflowServices themselves, representing a workflow schema, can be encapsulated and reused in other schemas.

Most of the expressions in the language were designed with the SafeFlow dialect in mind, particularly the Concurrent, Sequence and FailSeq expressions, which CircusFlow makes no use of. Concurrent and Sequence not only allow for encapsulation within the workflow schema, but also how control is encapsulated within the expression. Management of control via expressions like Concurrent and Sequence is an important aspect of SafeFlow.

![Figure 22: The SafeFlowService and related Non-Terminal Beans](image)

### 5.4.3 Walkthrough Of A SafeFlow Schema

At the top level exists a SafeFlow Service node, which holds the top-level expression of the schema of the SafeFlow Service. In the case of this example, it is a Sequence. The SafeFlow Service takes in runtime input A, B and C and returns output Y and Z.
Figure 23: A, B, and C are parameters passed to the SafeFlow Service, which returns X and Y. In this scenario, the SafeFlow Service has a Sequence as its top-level node, so can itself be assumed to be a Sequence.

If the SafeFlow Service’s root expression, a Sequence, is opened up we find several constituent expressions, i.e. The SafeFlow Service is a Sequence made up of a Sequence followed by a Concurrent followed by a Sequence. Within the SafeFlow Service it is possible to see how the runtime data travels through the constituent expressions, how new data becomes available as a result of processing existing data, and how some expressions are supplied with only certain data and not others. It is also possible to see that where the data returned by the Workflow Service at the top level derives from.

Figure 24: Within the expression, data is routed appropriately.

If the Concurrent Expression is opened up, it is possible to see that in this case it is made up of Terminal Expressions – actual Web Services. Since these constituent expressions run concurrently, they do not pass data between each other. In this case, it is also possible to see that some new data local to this expression has been introduced, N, which is static data defined at design time for use in this Expression. It is also possible to see that though data item X is generated by a constituent Expression, it is filtered out by this Concurrent Expression and not made available to outer Expressions.
Figure 25: This Expression shows how data can be statically defined and filtered within an Expression as appropriate.

The movement of data is represented through the arrows, while the movement of control is through the recursively encapsulating perimeter boxes that represent the expressions.

5.4.4 SafeFlow In XML And Visual Form

The XML version of the above example is too long for reproduction, but the following gives a sample of how the schema would be represented in XML.
Figure 26: This sample XML schema shows a SafeFlow Service with a Sequence expression that has two Static Web Services. The second highlighted import parameter refers to the parameter imported from its outer expression, using the reserved '#import', and the name of the parameter, 'sourcedata'. The first highlighted import parameter follows a similar convention except 'data' refers to the name of the parameter passed in at runtime. Export parameters use a different referencing convention. The export parameter for 'someproxy' refers to the parameter called 'return' returned from the Web Service. This return value is accessed using the reserved '#details'. The import parameter for 'someproxy2' refers to the export parameter from 'someproxy'. The export parameters for the Sequence and Workflow Service refer to export parameters from their constituent expressions, as is the convention with export parameters. The export parameters for the SafeFlow Service refer to what is returned at the end.
5.4.5 Non-Terminal Expressions For Control

The Non-Terminal Beans maintain a list of constituent expressions. With the exception of Bayesian, the actual descendents of the Non-Terminal Bean, such as Concurrent and Sequence, act as marker classes for interpreters.

- **Sequence Expressions** use the list of constituent expressions they hold as an in-order execution pattern.

- **Concurrent Expressions** use their list of expressions as a set of expressions that are executed concurrently. All the constituent expressions finish executing before the hosting Concurrent Execution can be considered to finish executing. This is effectively a synchronisation mechanism.

It should be noted that constituent Expressions in Concurrent Expressions cannot have ImportParameters that refer to ExportedParameters from adjacent expressions in the same Concurrent Expression. In Sequence Expressions, constituent expressions can only have ImportParameters that refer to ExportParameters that of adjacent expressions that precede the given expression.

5.4.6 FailureSeq for Error Handling

WebServices have the potential to fail. Consequently, with each WebService Bean it is possible to define within it a FailureSeq Bean as a field. This part of the WebService expression is only executed if the Web Service that is called fails. FailureSeq is a Sequence expression intended to define actions necessary to overcome the failure. FailureSeq can also obtain ImportParameters from the WebService Bean its resides in, and will usually output ExportParameters as substitute ExportParameters values that the Web Service ought to have returned, or as an indication of the WebService's failure. This is a basic but powerful Error Handling mechanism for Web Services.

5.4.7 Assessment Of The Workflow Model

Because the flow of control is defined through composite non-terminal expressions, rather than a graph, the concurrency model is restrictive, as all the constituent expressions of a Concurrent synchronise before execution leaves the Concurrent expression. It is possible to use permutations of
Concurrent expressions within Concurrent expressions to work around some problems, but the following graph representation of control can still not be represented within the SafeFlow language.

![Graph Representation of Control](image)

Figure 28: A graph-based workflow that cannot immediately be described in SafeFlow

In this example, A and C could be Concurrent or A and B a sequence within a Concurrent which also has C. Alternatively, C and D could be a sequence concurrent with A. In the first case, A and C must synchronise before D can occur, which is undesirable since C may finish early. In the second case, A may finish before D. The problem is due to the perimeters of the control expressions acting as implicit synchronisation mechanisms for starting and stopping of an expression, and consequently motivates the need for the CircusFlow dialect. SafeFlow can be combined with CircusFlow to overcome this problem, as Web Services deployed using CircusFlow can be used in a SafeFlow workflow. This approach is described in more detail in the Evaluation.

5.4.8 Related Work

The difference between SafeFlow and other ORB workflows is that of control structure. While with OTSArjuna or WSFL, for example, control is implicit and described by lines, SafeFlow considers an approach that does not describe control in terms of graphs, but encapsulations. WSFL workflow can also be deployed as WebServices, but does not have any form of encapsulation. OTSArjuna allows for encapsulation of workflow as intermediate Composite Tasks, but these do not represent control, and instead are perimeter sub-graphs of control and data lines. The significant advantage of graph-based approaches is that of freedom, while that of a structured approach like SafeFlow is encapsulation and tight restraints over control of execution in the workflow.

5.5 The CircusFlow Dialect – CircusFlowService

Expressions Normally Used With: Sync, Bayesian, WStatic, WDynamic

5.5.1 Aim

- To allow for a more liberal concurrency model for workflow than the SafeFlow dialect.
- To provide for strict management of the workflow’s data flow.

5.5.2 Derivation

Eminent scientist Werner R. Loewenstein [WRL99] argues a hypothesis that information flow is the ‘prime mover of life’, and that the circular flow of information from source to sink back to source, with mutations in the process, refines the information to bring order and create an ‘organism’ that preserves the information against disorganising pressures – in essence a view of evolution.
Most approaches to computing consider the flow of control as of primary importance, with data flow a secondary concept overlaid above it. In deriving the CircusFlow dialect, the consideration is in taking data flow as the primary flow in the system, with control used as a secondary concept to constrain the way in which data flows. The general idea is not new [WAN99], but has thus far been mainly the focus of highly parallel super-computers.

Figure 29: The CircusFlow Bean

5.5.3 Walkthrough Of A CircusFlow Schema

CircusFlow follows largely the same intuitions as SafeFlow, due to the inheritance of expressions from the AbstractExpression. Below the top level CircusFlowService, other expressions exist, normally WebServices, though Bayesian Expressions are possible. Other Non-Terminal Expressions, such as Sequence and Concurrent, are not made use of, as they are control orientated. Unlike SafeFlowService, CircusFlowService does not require a single root expression below it, as it already has a specified behaviour – enact all constituent expressions that have the necessary data to execute, and as these output data check it other non-started constituent expressions can be enacted.

Figure 30: A CircusFlow schema, with arrows indicating data flow dependencies

In the above diagram, the data flow dependencies are specified through the ImportParameter and ExportParameter links. The determination of which WebService should execute is through resolving which of the WebServices have their ImportParameters parameters fulfilled, and can execute. In this case, WebServices 1 and 3 are ready for execution and execute immediately, with all others waiting for their ImportParameters to be fulfilled.
Once WebService 3 has finished, the data is made available to WebService 2 and 4. WebService executes immediately, as it has all the necessary information it needs. When WebService 1 is finished, its result is made available to WebService 2, which can begin executing given it has the information from WebService 3 also.

The Sync Expression serves to synchronise data from WebServices 2 and 4. Once this is complete, data is sent from the Sync to WebService 5. The Sync serves to filter out particular data that is not required by WebService 5, while ensuring that the unrequired data is first produced before WebService 5 is started.
From the above example it is possible to see that through careful manipulation of data links of ImportParameters and ExportParameters, as well as the use of Syncs, it is possible to overlay partial control over the data flow.

5.5.4 CircusFlow In XML And Visual Form

The CircusFlow schema is similar to that of SafeFlow, except expressions like Sequence and Concurrent do not appear; the same intuitions for relations in terms of data movement between expressions hold – its is largely the enactment of the representation that changes, not the underlying form of the schema data structure. The same is true for the visual representation. Consequently, reproducing both would be superfluous.

5.5.5 Assessment Of The Workflow Model

A high level of concurrency is catered for by the CircusFlow dialect, and the given example walkthrough is one that overcomes the limitation identified of SafeFlow. The CircusFlow model, however, enables this through the omission control being described through composite Non-Terminals. Additionally, encapsulation is not possible, so complexity is not managed. Although it is possible to add encapsulating Non-Terminals, which do not represent any control expression but are constituent CircusFlows, this would cause implicit synchronisation of data to occur at the at the Non-Terminal expression’s entrance and exit. An indirect way of achieving this is still available through deploying a CircusFlow as a WebService and including it within another CircusFlow.

The advantages of pure data flow, with no implicit synchronisation, and only carefully used explicit Syncs to twist any control, resulting in high parallelism are contrasted by some dilemmas. The case of WebServices that require no data for invocation is not fully catered for. It is possible to send ‘null’ data to invoke such a WebService, and specify that the WebService needs ‘null’ data for invocation, but this is largely a philosophical argument as to whether null data represents control.

CircusFlow is best used in combination with SafeFlow, to overcome the problems of both. In particular, Web Services deployed using CircusFlow can be used in a SafeFlow workflow, to overcome SafeFlow’s synchronisation problem. This approach is described in more detail in the Evaluation.
5.5.6 Related Work

OTSArjuna is similar to the CircusFlow, with the significant difference being that of design intent. OTSArjuna mixes control flow and data flow, with the former represented as notification requests, and all control and data is implicitly synchronised at each Task or Composite Task. CircusFlow does not allow for expressions similar to Composite Tasks, and instead provides for explicit synchronisation of data in preference to overuse of notification style control. With CircusFlow, control is through manipulation of data flow, while in OTSArjuna, control flow complements data flow in the parallel.

In looking for existing work on data-flow based computing, the simple approach used in CircusFlow is found to be similar to Dennis dataflow graphs [WAN99], but it is not clear from the description in [WAN99] how similar, and the citation to Dennis dataflow graphs has not been possible to track down. Data-flow based computing is described as having ‘no notion of a single point or locus of control - nothing corresponding to the program counter of a conventional sequential computer’ [WAN99], and has largely been of interest mainly in multithreaded execution, signal processing, and reconfigurable computing.
5.6 Terminals

WebService Beans contain fields for the locating URL of the Web Service, the particular Service Name, Method Name to be called, the HTTP SOAP Action (header information on call intent) and the XML Namespace defining encoding style of the call. With WebService expressions, the intention is that the ImportParameters will be cast into Parameters (Apache SOAP Beans) and sent to the Web Service, with the returned Parameters converted to ExportParameters. The WebService expression is not itself instantiated, instead leaving that to an inheriting expression, both of which simply serve as marker sub classes for intent of the Bean:

- **WSStatic** is an expression where the details of the Web Service are defined with the Expression at design time, as fields.

- **WSDynamic** is an expression where the details of the Web Service are passed to the Expression as special ImportParameters at runtime. These special ImportParameters correspond to their respective fields, using the names ‘#soapend’, ‘#nameuri’, ‘#method’, ‘#encode’ and ‘#action’.

In the SafeFlow dialect, WebService Beans may also have a FailureSeq field defined.

Sync Beans are purely markers for CircusFlow Interpreters that enact the workflow, and do not specifically have any data specific to them. All ImportParameters / control must arrive at the Sync before ExportParameters, which are identical to the incoming ImportParameters, are produced for continued execution.
5.7 Bayesian Probabilistic Method Calls

5.7.1 Aim
Provide a more powerful choice mechanism than simple Guards, accounting for the automation of choices that may otherwise require some level of human intuition. E.g. invoking the CD to play a particular track is determined by the weather, room temperature sensors etc. The invocation by the user is explicit; the exact nature of the invocation is not. This approach accounts for probability distributions over different choices that an automated system using if-then-style selection statements do not account for, and hence provide an automated choice mechanism that reflects the non-systematic decision processes of humans.

5.7.2 Derivation
A computer machine will have different states dependent on the input it receives and its previous state. In a distributed system, the input for a computer may come from other machines it is connected to in a network, which themselves will have states. The different computers in the network can be correlated to variables in a Bayesian Network, and the states of the variables to the states of the machine.

In this way, it is possible to see that given several ‘child’ computers that provide input to a ‘parent’ computer, the state of the parent computer could be deduced from the states of the child computers. Since the child computers are black boxes, and their state not easily deducible, the output from the children, which is input for the parent, can be used for probabilistic determination of the parent’s state. The transition to the required parent state itself is best achieved through a method call that performs the desired function.

A good introduction to naïve Bayesian networks can be found in [DFG, L2].

5.7.3 Implementation
Probabilistic Method Calls are based on the idea of having a set of possible method calls, with the exact invocation of one of the method calls probabilistically determined based on the input from various sources. A simple naïve Bayesian network is used to analyse the input. For each input, quantisation is used to determine the appropriate ‘child’ state. Conditional probability matrices relating the different states of each input to that of the ‘parent’ methods, along with prior probabilities of the methods, allows for the calculation of the probabilistic likelihood of each method call. The method call with the highest posterior probability is invoked.
Within the workflow language, as Bayesian is a non-terminal, the method call correlates to one of the constituent expressions. If the constituent expression is a WebService expression, then it is a method call that is probabilistically determined. If the constituent expression in the Bayesian is a Non-Terminal expression, then a Sequence or a Concurrent, or possibly another Bayesian, is invoked.

The Bayesian Bean is similar to other Non-Terminals in holding a list of constituent expressions, but has an additional string field that represents the probabilistic information. The conditional probability matrix information, the quantisation information for the input and prior probability information could be represented in any number of ways, and is further complicated by the possibility of having any number of inputs of the Bayesian expression, and any number of constituent expressions, both of which will change the size of the previous mentioned information. Consequently, for the purposes of this project, a simple string field is used to represent all the information, which the interpreter parses as appropriate. A more elegant implementation could allow for constituent Conditionally Probability Beans and the like, but time constraints did not allow for this.

Exact algorithmic details of the execution of the Bayesian Expression are given with the Interpreter, in section 6.5.

5.7.4 Probabilistic Method Calls Use Case

The most realistic applications of Probabilistic Method Calls are those that are not accuracy critical or where systematic methods have difficulty in producing an accurate model for basing an invocation on.

There may exist a Route Web Service that will automatically generate a route on a map, given source and target locations and any necessary intermediate locations. Typically a Traffic Route Web Service would use the Route Web Service to generate a route by providing the source and target locations, and intermediate locations with least traffic. Instead, in this simple sequential workflow, a Bayesian Expression wrapper can be used over the Route Web Service, with the wrapper taking additional information from other Web Services. This additional information could be details like the time of day. The Bayesian Expression wrapper could hold a naive Bayesian network model accounting for the time of data against intermediate route locations, with the model reflecting probabilities of likely increases and decreases in traffic in those intermediate locations according to the time of day. The set of parent

\[
P(M | I_1 \& I_2 \& \cdots \& I_n) = \frac{\alpha \cdot P(M) \cdot P(I_1 | M) \cdot P(I_2 | M) \cdot \cdots \cdot P(I_n | M)}{P(M)}
\]

- \(P(M | I_1 \& I_2 \& \cdots \& I_n)\): Probability distribution of method variable \(M\) given input variables \(I_1\) to \(I_n\).
- \(\alpha\): constant that can be generally disregarded
- \(P(M)\): Prior probability of \(M\)
- \(P(I_k | M)\): Conditional probability of \(I_k\) given \(M\)
method calls could correspond to the intermediate locations and slight deviations from them to
neighbouring locations. The method call corresponding to the route deemed to have the least likely
traffic is invoked for creation of the route map. A systematic approach would need to account for non-
determinism, such as 60% probability traffic will increase at a given intermediate location, which is not
easily computed with if-then-style selection statements.

Other examples may include a business where there is high airline travel, and rather than invoking the
same Airline Web Service for each booking, there is a possible set, with the appropriate invocation
based on a naive Bayesian networking taking variables for the time of year, person travelling as well as
information necessary for the booking. A systematic approach may not be able to account for
probabilities of fluctuations in prices versus traveller’s preference; a cheaper airline may not necessarily
be chosen if the traveller has a high enough preference, represented by probability, against it, though
this preference may not be high enough for an extremely low price.

5.7.5 Related Work
The normal use of Bayesian networks thus far has been in interpolating data, such as identifying
images. The limitations of this Bayesian approach is of the remote possibility that the user may wish for
a result that is significantly different to the distribution of probabilities upon which the system is built.
The current mechanism can be extended to handle for such occurrences, learning as necessary.

5.8 Data Binding

5.8.1 Aims
The mechanism used for converting between the XML representation of the Schema and the internal
representation of connected Beans needs to consider the concern that the language could be extended
to handle extra functionality / expressions, not just during the design and implementation of the whole
system, but after this when in use.

5.8.2 Derivation
It is desirable that the mechanism that converts between XML and Bean representations is self-
contained and has no dependency on the language or the internal Bean representation. In this respect,
the approach of using Java-XML data binding is a desirable choice. Sun is currently in the process of
producing an API for compiling an XML schema into one or more classes [SPA]. The generated
classes handle all the details of XML parsing and formatting and ensure that the constraints expressed
in the schema are enforced. As the API is not yet available, and the general idea is simple, it is possible
to build a custom and specialised version of a conversion process based on Java Reflection.
5.8.3 XML to Beans

The one-to-one mapping XML from to Bean representation follows these guidelines:

- An XML element is equivalent to a Bean in the internal representation.
- An XML attribute is equivalent to a primitive type that is a field in the Bean.
- An XML element that is nested in another element is equivalent to a field that is an internal representation Bean within another Bean.

These guidelines can be exploited with the following operations in Java:

- Given the name of the XML element, it is possible to create a Bean instance of it using Java Reflection.
- Once a Bean is created, if the attributes of the XML element are known, it is possible to use the Bean’s ‘set’ mutators to set the fields of a Bean to that of the element’s attributes, matching equivalent attribute names to mutator names.
- An inner element in the XML implies that the Bean has a mutator for a field with that element name, and a new instance of the inner element bean must be created for assignment to the field of outer element bean. This can be a recursive process, building a large internal tree.
- Where there are several inner XML elements, or the Bean’s field represents a list of instances of a particular type of Bean, the mutator in the Bean has the effect of
adding to, rather than setting, the field. This is the case for ImportParameters / ExportParameters and lists of expressions in Non-Terminals.

As long as each XML element type has an equivalent Bean, this mechanism should be able to recreate the Bean from the XML using Java Reflection. A significant advantage with this approach is that the length of code is less than with other typical applications, as there is no matching to map one representation of the data to another – just straight conversion. The loading mechanism is implemented as the Class SchemaXML2Bean.

5.8.4 Beans to XML

The mechanism that converts internal Bean representations to XML is similar to the loading mechanism, but in reverse. The Saving mechanism follows these guidelines:

- Each Bean is turned into an element.
- For each mutator that begins ‘getX’ on a Bean, ‘X’ is a field of the Bean. The field is represented as an XML attribute if the mutator returns a Java primitive. If the mutator returns a language Bean, the field represented as an XML element, and this process is applied recursively on the inner Bean.
- The ‘getClass’ mutator that all Java Objects have is ignored as irrelevant, as well as the ‘get’ method of EMethod, which is inherited Bean from Vector. EMethod is a Bean used during schema invocation communication between the User Application and WorkflowEngine.
- Data that is stored in internally as lists, such as ImportParameters and ExportParameters, whose original ‘set’ mutator served to add the data are handled by the ‘get’ mutator returning an Enumerator. Each entity that is enumerated is converted an XML Element on the same level of the XML hierarchy, rather than within a Vector or Enumerator Element. This enables several XML Elements of the same type to reside on the same level, as is necessary in the Schema language.

The Bean to XML mechanism is encapsulated in SchemaBean2XML.

Potential applications for Data Binding exist in any system where there is a conversion from XML to Java Beans or vice versa. The XML generated from Beans need not necessary be translated back to the Java Bean form – a C++ application could process it in XML form; XML becomes an intermediate data format for different object models. Many of the XML based languages, like Scalar Vector Graphics (SVG) for example, could be extended in future with new XML elements and attributes; extending the systems that use them largely entail adding / changing the Beans for these elements, though there may be other issues.

5.8.5 Related Work

There is one known implementation of Java-XML Data binding [XJ]. However, apart from a slightly different use of Java Reflection, [XJ] handles a different context of requirements. It does not consider
multiple instances of inner fields or the intricate handling of Class objects (necessary for data passing in the workflow), both of which is a central part of expressions of the workflow languages, and seems to consider specific server configuration file issues, which are relevant to the context of [XJ] but not here. [XJ] specifically also uses XML Schema for Bean creation and validation, which is not considered here.
6 WorkflowEngine

6.1 Chapter Summary

- The WorkflowEngine Bean is described, as well as its deployment as a Web Service and how it enables workflow monitoring from the User Application.

- Concurrent Visitors, permitting several Visitors acting concurrently on a data structure, is introduced as a way of interpreting the workflow.

- Algorithmic details are given for interpretation of the Bayesian Expression.

6.2 The WorkflowEngine In Outline

6.2.1 Aim

- To encapsulate the functionality of the workflow enactment.

- To allow the encapsulated system to be easily accessible, respecting low coupling with clients, allowing for the deployment of several Workflow Schemas.

6.2.2 Derivation

The different workflow schemas are easily stored in a data structure, and enactment of the appropriate workflow can be handled through a delegation pattern to the necessary workflow. Separating the concerns of enactment from the storage mechanism allows for the enactment to be applied as necessary, and there are various patterns that allow for this, such as a Visitor, or those found with the Java Collections Framework.

6.2.3 The Workflow Engine As A Web Service

The WorkflowEngine is deployed as a Web Service, and the general WorkflowEngine Web Service system allows for several WorkflowServices to be deployed. A WorkflowService is one instance of a Workflow Schema and is associated with an Interpreter, and possibly further Child Interpreters of the Interpreter, to enact it. A User Application, or other client, can create one or more WorkflowServices to deal with one or more workflow schemes that they would like enacted, and deploy them on the WorkflowEngine for enactment. In this way, a WorkflowEngine Web Service can hold different workflow for creating, removing and manipulating travel arrangements, as is possible in the scenario considered at the start.
6.3 WorkflowEngine

6.3.1 Aim

To implement a simple mechanism for the deployment and enactment of workflow schema, in such a way that they could be considered new method calls available on a server, for use by any client.

6.3.2 The WorkflowEngine As A Bean

The point of contact for clients wishing to invoke a WorkflowService is the WorkflowEngine. The WorkflowEngine is installed as a Web Service on a server supporting a SOAP implementation that allows for the deployment of SOAP services. In the case of the SOAP implementation used with this project, the SOAP implementation is a servlet that allows for the invocation of Beans that it holds.

The WorkflowEngine is a single Java Bean with static methods and a persistent Hashtable that maintains several WorkflowServices. A WorkflowService is a particular workflow schema. Each deployed schema behaves like a new method of the WorkflowEngine that is possible to invoke with parameters that become ImportParameters during enactment of the schema. The Bean is deployed as a WebService and has the following methods:

- $thisInstance
- WorkflowEngine
- schemaList : java.util.Hashtable
  - add()
  - remove()
  - retrieve()
  - enact()
  - run()
• **Add** a new Workflow schema. This method takes an XML String of the schema and converts it to an internal WorkflowService representation. The name of the Workflow Service, the top level ID attribute of the schema document, is used as the name with which the WorkflowService is invoked.

• **Remove** a Workflow schema. The String name of the WorkflowService passed to the WorkflowEngine is removed from the Hashtable.

• **Run** a Workflow scheme. An XML String version of an EMethod object is passed to the enact method. An EMethod object is essentially a sub-class of the Vector Java Class, except with an ID attribute used to store the Workflow Service name. Once the EMethod XML String is converted to an internal representation, the Enact method retrieves the WorkflowService name and passes the EMethod object as a Vector of ImportParameters to the WorkflowService that is to be executed. The returned Vector of ExportParameters from the WorkflowService is converted to an XML version of a new EMethod object and sent back to the client.

• **List** returns a string of WorkflowServices that have been deployed, together with the parameters they each require, in a regular formatted form.

• **Enact** is similar to the Run method, but is used only by the User Application and other clients capable of using the monitoring protocol. Enact is asynchronous, whereas Run is synchronous invocation of a WorkflowService.

• **Update** returns information from the MonitorPost on the current state of a WorkflowService that is being enacted. When the WorkflowService has finished executing, this method returns the final result. SOAP over HTTP does not allow for callbacks, so relaying of calls to Update is necessary.

More complex management of the Workflow Schema is possible though combining these operations on the client side, with another Web Service that acts as a wrapper for the WorkflowEngine Web Service.

### 6.3.3 Alternatives

Several options were considered in implementing the deployment of the Workflow Schema, with particular interest in ensuring encapsulation of the workflow as a method that can be invoked. The alternatives considered include:

1. Dynamic addition of method names to the Workflow Engine, one for each Schema; this gives the impression of the workflow as a real method, rather than passing the method as a parameter of the ‘enact’ method. Technology limitations did not allow for this. While this is possible to some extent, the new Workflow method to be deployed needs to be defined in a Class, with the Class in the Classpath of the Java Execution Environment. This entails creating a Class dynamically for the Workflow Schema, compiling it, stopping the server and restarting with the new Class. Shutting down the server is not tolerable, particularly since the server is what would actually be running compilation, shutdown and restart sequence.
2. The Enact method does not take an XML string representing an EMethod object, but instead has at least two parameters, one for the name of the Workflow Service to invoke, and others for the parameters to pass to the Workflow Service. This solution is problematic, in that the number of parameters will vary according to the Workflow Service to be invoked. Though to some extent, this can be overcome by providing several Enact methods, each overriding the other for different number of parameters, this solution does not scale well and is inelegant. There is also difficulty in the return type of the Enact method, unless there is overriding for the different return types also, there can only be one return type.

The solution chosen, that using an EMethod object passed as XML is both scalable and low overhead. The EMethod XML passed is only ever one level deep, and the EMethod object is simple to implement. This solution also overcomes the problem of needing to constantly update the WSDL file describing the methods available from the Workflow Engine Web Service, which can be problematic if the first option is chosen; with this solution the List method is the single point from which the available Workflow Services can be found.

6.3.4 The MonitorPost For Monitoring

The MonitorPost is a shared monitor singleton. Interpreter instances that enact the workflow register with the MonitorPost the identity of the AbstractExpression they have entered, together with the workflow schema the AbstractExpression is in, as they are entering it. When they have finished enacting an AbstractExpression, they unregister the identity of the AbstractExpression.

When an update call comes from the WorkflowEngine, with a specified workflow schema, the MonitorPost returns an EMethod object identifying a list of active AbstractExpressions for the schema. This system for monitoring the workflow enactment is very simple, but is no more than necessary, as it provides all the necessary functionality. While the singleton instance means that the system is constrained in terms of scalability, for the foreseeable short-term use of this system it should not be problem. A more complicated implementation using Publisher-Subscribers is possible.

6.4 Interpreter – A Concurrent Visitor

6.4.1 Aims

- Provide a highly cohesive and maintainable mechanism to interpret Expressions.
- Enable concurrency in the workflow to be handled.

6.4.2 Derivation

The enactment of the workflow, once it has been converted from XML to the internal representation, is due to the Interpreter, which is a Visitor that visits the hierarchy of Beans that make up the internal representation of the workflow, as per the Visitor Pattern [GHJV]. This allows the encapsulation of interpretation control and co-ordination in one logical object. Otherwise, there would be interpretation code across the different Expression Beans, making future modifications difficult because of interdependencies.
However, the implementation of the Visitor Pattern used is modified, in that each time a new Bean is visited, a new instance of the Interpreter evaluates it rather than the current one evaluating the current Bean. This design decision is justified as follows:

- To allow for the Concurrent Expression, it is necessary to split the execution to several threads running concurrently. In this scenario, one Interpreter is not satisfactory.

- It is possible for new instances of the Interpreter to be created only for the Concurrent case. During the visiting process, the Visitor enters the visited, and then the visited enters the Visitor; this is the point at which the new thread needs to be started to allow the Visitor to perform operations on the visited. This means the visited needs to check whether the current Interpreter Visitor is due to a Concurrent case, to see if a new thread should be forked. A Concurrent case is where the visited Bean has a Concurrent Expression as its parent; where are all off the Concurrent Expression’s children need to interpreted concurrently. Restricting new threads to the Concurrent case causes unnecessary complexity.

- All Beans having their own Interpreter thread makes Concurrency a natural part of the design, rather than it being an exceptional case that has to be handled differently, which would be outside the Interpreter’s interpret() function for the Concurrent Expression type. The overall effect of each Bean having its own Interpreter thread is the same as that of the alternative described above, as other non-active threads are put to sleep.

- During Interpretation, the information held in the Bean is not changed. Instead, the Interpreter holds temporary information deriving from processing the Bean. This allows for several instantiations of the workflow by different clients, all represented by their own sets of Interpreters. If instead a single instance of an Interpreter were to visit all the different Beans, it would be necessary to use a stack to hold temporary information that must be saved before traversal returns to the Expression where that temporary information was needed. This is likely to be more complicated than creating a new Interpreter instance.

### 6.4.3 Pattern of Enactment

When the workflow is invoked in a WorkflowService, such as SafeFlowService or CircusFlowService, a new instance of the Interpreter is created for the first time, and the accept function of the top-level WorkflowService schema Bean is passed the instance. Within the Bean, the Interpreter that has been passed to it has its visit function invoked, with ‘this’, the Bean itself, passed as a parameter.
The initial entry process starts as per the normal Visitor pattern:

1. The new Interpreter instance has access to its hosting Bean and the Interpreter’s visit function then sets the passed Bean as the associated Bean for the Interpreter – each Interpreter instance has a field that is the Bean associated with that Interpreter.

2. Within the visit method, the Interpreter calls its start function, and execution forks. The Interpreter’s run method runs concurrently with original execution path in the visit method, which returns to the line after the original accept function. This accept method will have been called by WorkflowService’s enact method, or the interpret function in any other Expression Bean.

3. The visit of the original Parent execution path is over, and is put to sleep, but the Child Interpreter instance continues another execution path that remains with the visited Bean, processing it. The original Parent execution path now waits for the Child Interpreter instance to finish execution, yielding processing time when execution falls on it.

The new execution path, within the run method of the now active new Interpreter thread,

1. Evaluates its associated Bean to determine its type.
2. Following this, the Interpreter calls its interpret function which overloads for the different types of Expressions.

3. Interpretation of the Bean occurs, dependent on the type of the Interpreter sub-class, for SafeFlow or CircusFlow, which is enacting the schema. This could involve the spawning of one or more further Child Interpreters, for each constituent Expression, acting concurrently. Each Interpreter visits the Beans’s constituent Expression.

![Diagram](image)

*Figure 46: Continuing from the previous figure, except generalising to an AbstractExpression – any Expression Bean. If the Expression Bean has several constituent expressions, a new Interpreter instance is created for each.*

![Diagram](image)

*Figure 47: Left: The Child Expression has its accept function called, which is passed the new child Interpreter instance. In a normal Visitor Pattern implementation, the parent Interpreter would have passed ‘this’ – itself. Right: This child Expression calls the child Interpreter’s visit function passing itself to it; the child Interpreter’s visit function takes the child Expression as its associated Expression and calls start(), forking execution. The original execution path returns to the child Expression’s accept and then the parent Interpreter’s interpret function. In the case of a Sequence interpret function, the parent Interpreter will sleep until the child Interpret finishes, and then start the visitation process for the next child expression. In the case of a Concurrent, the visitation process next child expression begins immediately. In the case of a CircusFlow, if any of the child expressions has sufficient data to start, the visitation process starts for it, and the child expressions are rechecked each time a child Interpreter finishes and produces more data.*

4. Once interpreting has finished for the Bean associated with the hosting Interpreter, the thread for this Interpreter then dies.

Because of this enactment pattern, several concurrently acting Interpreters can be spawned if necessary by the original execution path, which can wait for them all to finish interpreting.
6.4.4 Co-ordination Between Concurrent Visitors

As well as executing and interpreting the Expression Beans, the Interpreters also need to communicate between each other the ImportParameters and ExportParameters that become available and are necessary for different Expression Beans to execute. This is achieved by overlaying a double-dispatched Publisher-Subscriber version of the Observer Pattern over the Visitors.

1. When the Parent Interpreter creates the Child Interpreter, the Parent is registered with the Child as the Parent, and the Child is registered with the Parent as one of its Children.

2. The Parent publishes to its Children the ImportParameter data necessary for a Child to execute. The method the Child listens on (i.e. the method invoked by the Parent) is protected as a synchronised method, for concurrency purposes.

3. During the interpretation process, once a Child has finished interpreting, the Child publishes the ExportParameter data to the Parent. The method the Parent listens on is protected as a synchronised method.

4. Actual co-ordination of information is dependent on the nature in which an Expression handles any constituent Expressions it has.

   - **SafeFlow Concurrent**: In the case of many Children running concurrently, once all of the Children of the Parent have published ExportParameter data to the Parent, the Parent can publish its own ExportParameter data to any parent it itself has.

   - **SafeFlow Sequence**: In the case of a several children running one after the other, the ExportParameter of one finished Child, along with that for other finished Children, is made available to the next Child, until the final Child in enacted.

   - **CircusFlow**: In the case of some Children running concurrently, with other Children waiting for ExportParameters, which will become their ImportParameters, from perquisite Children, ExportParameters data is published
as it becomes available to any Children yielding execution time. A Child's ExportParameters is published to the Parent, which publishes the data to waiting Children.

5. Once the co-ordination of information with children is over, the Parent can publish any data it has to any Parent that it itself has.

6.4.5 The SafeFlow Interpreter

For each of the Expression Beans normally used with SafeFlow:

1. If the Bean is a Terminal Expression, the relevant processing is done for the Expression, e.g. calling a Web Service.

2. The SafeFlowService will only have one root Child Expression, which will be parsed over to interpret the root Child Expression.

3. If the Bean is a Non-Terminal Expression, for each constituent Expression Bean, a new Interpreter instance is created.

   • In the case of a Sequence, each constituent Bean is processed by the Interpreter instance that is created for it, but each instance is only allowed to process its Bean in order of the Sequence.

   • In the case of a Concurrent, each constituent Bean is processed by the Interpreter instance that is created for it, and all the instances are started concurrently. All the instances are grouped in a Thread Group, and when the
Thread Group indicates all its threads are dead and all the data from their enactment accumulated, the original execution of the hosting Interpreter continues, effectively synchronising all constituent Expressions.

- FailureSeq Expressions are interpreted in a similar way to Sequences. Interpretation of the Bayesian is explained at the end of this chapter.

### 6.4.6 The CircusFlow Interpreter

For the Expression Beans normally used with CircusFlow:

1. Terminal Expressions are interpreted as expected, e.g. calling a Web Service or synchronising data with Sync. Bayesians are also available.

2. CircusFlowService processes all constituent Beans that are ready for execution, in terms of the constituent expression having all their ImportParameters satisfied. As constituent expressions finish, more data, in the form of ExportParameters, becomes available. If this new data item allows for any of the un-started constituent expression’s ImportParameters to be satisfied, then they are started. When all the constituent expressions have finished the relevant accumulated data is filtered and returned.

### 6.5 Interpreting The Bayesian Expression

A string field of the Bayesian Expression, representing the probabilistic information for the constituent expressions, might be as follows:

![Figure 50: In this example, the main delimiters and delimited content are as follows, and given in the format content(delimiter)content: variable information#parent posterior probabilities, variablename | input range | conditional probabilities.

The motions of the interpreter in analysing a Bayesian Expression are given below, and is an efficient implementation of the normal Naive Bayesian Network calculation, taking account of the fact that all the child input variables have been instated with a value.
1. The interpreter will first create an internal version of the above parsed information. It is possible to infer from the constituent expressions of the Bayesian, as well as the given data, information such as the number of parent and child states.

2. For each input that the Bayesian Expression receives, it will calculate the state of that input variable by determining between which ranges the variable value falls in. For example, for a variable in S with value 1, the state will be the first state, state one. For a variable D with value 2, the state will be the second state, state two.

3. Based on the value of a variable, it is possible to determine the row of the conditional probability matrix that is relevant. Continuing with the example of variable S, this will be the first row of (0, 0.33, 0.14). Each item in the row represents the conditional probability of a parent method state; given the instantiated variable S state, state one.

4. The equivalent row for each of the other variables is determined. For each of these discovered rows, all the values that are placed in the same position in the row are multiplied together. In this example, D is the remaining variable, which with state two returns the row (0.4, 0.33, 0.14). The culminated probability after the row from variable S is calculated is (0, 0.1089, 0.0196).

5. The posterior probability of each of the parent states is factored into the culminated probability evidence. In this case, the posterior probabilities are all one, so the result of this operation is (0, 0.1089, 0.0196).

6. Given that each row item represents a parent method state, the state with the highest probability is taken. In this case, the method call that is represented by state two is invoked.

The initial values from the above example are taken from part of [DFG,TUT1Q3], where the tutorial answer is given as (0, 0.11, 0.02).

Section 5.7 gives further details on the Bayesian Expression with real use cases for it.
7 User Application

7.1 Chapter Summary

- The top-down hierarchical approach for creating workflow is motivated and described.

- The internal architecture of the User Application is described.

7.2 User Application In Outline

7.2.1 Aim

- Arguably the most important part of the system, the user interface needs to present a clear, ‘easy to use’ and intuitive way for the user to create workflow schemas.

- The User Interface has to present a method for designing workflow; a way to use the language.

7.2.2 Derivation

Several different approaches were considered in deciding how individual expressions, as well as the workflow schema as a whole, could be created. Considering the necessary complex manipulation of control and data, the User Interface needs to be clear, concise and intuitive. Workflow creation using wizards, or control boxes, was considered. This would be similar to how certain types of documents are created in Microsoft Word, for example. The approach would have only been for some more complicated expressions, like Sequence, where the ordering of constituent expressions is intricate, and data links restricted. Another alternative would have been a more text-orientated approach, giving control of the language directly. After some thought, the principal parts of this problem were identified as creation of a complex structure of data, but where structure is made of constituent components; similar to the principal parts of user-interface design. In the same way many IDEs now present standard functions, like buttons, that can be drawn on a canvas, a good solution would be to follow the same style – workflow creation through drawing on a canvas standard expressions, and defining the properties of the expression in a ‘properties box’.
7.2.3 The Work Area

![Image](image.png)

Figure 51: Left: properties window, right: canvas, bottom: monitor window

A single canvas is presented with buttons at the top of the display for the manipulation and addition of expressions using the mouse. A properties box displays the values of the fields of each of the expression Bean, as they are clicked with the mouse. The buttons that are available for use to manipulate expressions on the canvas are of three types:

- **Select**, for selection and editing of properties of an expression
- **Link**, for linking two different ImportParameters / ExportParameters.
- **Drop**, which consists of several buttons for adding the different expressions available. The Delete button is also of this type of button, as the characteristic of clicking and deleting is similar to adding.

Use of the Monitor Browser, which is the means by which workflow is deployed to and invoked on the WorkflowEngine, is presented as a separate mode where some of the functions of the Canvas and Properties windows are disabled, and the Monitor Browser window is introduced to the screen. The Monitor Browser consists of buttons to deploy the current open workflow schema, undeploy it, invoke it and to query the WorkflowEngine as to whether the schema has been deployed.
7.2.4 Alternatives

There are some packages freely available, sometimes with source code, for manipulation of graphics and graphs visually. Initially, it seemed possible to use these to reduce development time in building the user interface, but none of them satisfied the requirements to provide a good fit with the necessary user interface for this application. Of these, the most promising were IBM Graph Foundation Classes [GFC], and OpenJGraph [OJG]. However, both of these were far too complex for use with the User Application, and consider pure graph problems – data structures consisting of edges and vertices. The workflow language does not explicitly use expressions equivalent to edges, but references within one expression tracing to another, and the visual representation of expressions could be many levels deep, with expressions inside Non-Terminal expressions, whereas in graphs vertices are only one level deep. Additionally, [GFC] and [OJG] seemed to use more than one data structure for each graph; one mathematical representation, and possibly many graphical representations, all linked by action listeners so changes are propagated. For this application, this is not necessarily desirable as all data is neatly unified in one data structure. Consequently, the entire User Application was written from scratch.

7.3 A Method Of Designing Workflow – B Method

7.3.1 Aim

Present a structured design method and approach for creating workflow with the available workflow language dialects, Safeflow and CircusFlow. The envisioned process should motivate the user interface of the canvas areas for creating workflow.

7.3.2 Derivation

There are several ways in which workflow could be created.

- **A bottom up approach**, where the user draws the Web Services that will be used, and composes them as necessary to create the required end workflow. This method is direct and intuitive, but unstructured, as the final result may not have been well thought out.

- **A top-down approach**, where the user defines higher-level elements and compositions and decomposes them to the final bottom level Web Services. This method is initially not as intuitive as a bottom up approach, but does force the workflow designer to think about the problem before putting the workflow together.

- **A bottom-up / top-down mixed approach** has the benefits and drawbacks of both. The workflow designer can be forced to think about the problem, or they could simply put something together without thought, consequently is the most preferred by some. However, the resultant outcome is not a method for designing workflow per se, but more a set of available options for adopting a method.

The style of approach that is adopted that of the top-down method. More particularly, the approach is one inspired by the B Method, and allows for further possibilities beyond the scope of this project, as described in the Conclusion.
7.3.3 The B Method

The B Method describes a process of defining a high level specification, which is consequently refined to produce more specialist and detailed specifications for each higher-level segment. This is a process that is continued until the specification is refined enough such that implementations, often pre-built Machines, can be substituted as the lowest level refined specification. Each specification and refined specification is proven to be correct, as well as the machines that are plugged in. In this way, higher-level aims can be met by pre-built machine components; the final result of software designed in this method is software that is proven to be correct against its specification.

Machines are like objects, and in this context are analogous to Web Services. With the inherent process of recursive encapsulation in the language, particularly for SafeFlow, each Non-Terminal expression within another can be analogised to a refinement. This, combined with the idea that workflow themselves can be deployed as new Web Services, produces a good match for a method of design for creating workflow.

The workflow can be designed at a high level and decomposed into Non-Terminals and predefined workflow that are now Web Services; this can be a recursive process. At each level of decomposition, the workflow would not have to be proven to be correct as such, but since as the workflow language is a far simpler language than that of the B Toolkit, restricting the allowable linking of data and composition control should be sufficient. Proving that the bottom level Web Services satisfy some specification is beyond the scope of the User Application.

7.4 Initialisation And Window Management

7.4.1 Layout And Initialisation

The User Application, like most Java Swing applications, consists of a central main JFrame, here the GUI Class, on which layers of other graphical components are added.
Figure 52: The hierarchy of visual components in the User Application. The reader need only observe the leftmost / highest level components to obtain an understanding of the User Application’s visuals. It is worth noting that this is the tree that the Java Graphics Visitor will traverse during painting.

The design of floating windows, JInternalFrames, on a desktop background, JDesktop, was chosen over a single window with frames packed within it. The two main reasons for this were to

- Further give the impression of an IDE style development environment; the user can resize and reposition each JInternalFrame as they require.

- To overcome the use of Java Layout Managers, none of which appropriately gave a professional enough look to the User Application, especially during resizing to different screens
The process of creating the different components is delegated by the GUI Class to the CanvasManager, PropertiesManager and MonitorManager, each of which encapsulate control and management for each of the JInternalFrame ‘windows’. Each of these descendents of AbstractManager is not a graphical component themselves, but use aggregation style instance creation of the JInternalFrames and other graphical components that are recursively added to higher level graphical components before they themselves are added to the JInternalFrame. The JInternalFrames created by each AbstractManager is created as a singleton instance, available through a static method in the AbstractManager. Each of the three JInternalFrames will be manipulated by other functionality occurring elsewhere, and use of a singleton allows for easier referencing of the JInternalFrames. The use of a Publisher-Subscriber pattern, like action listeners, was considered, but since it is very likely the number and nature of each JInternalFrames will never change, singletons were used in preference.

7.4.2 The Menu Bar

The MenuManager, that sits generates the JMenuBar, which sits at level higher than the components from the other AbstractManagers, controls all the functionality of the Menu Bar seen on screen. The MenuManager tags action listeners to all events generated by the JMenuItems, and performs the
necessary actions, which are contained in their own functions in the MenuManager. These functions are:

- **Loading and Saving.** This is not the same as conversion of Beans to and from XML, as there is a lot of graphical information, more complex than simple Bean fields that are associated with each expression Bean that is not easily amenable to the process. Java Serialization was used to load and save the CanvasPanel, within the JInternalFrame Canvas, to and from a file. This reduced the number of lines of code needed for the process to a dozen, and it is reasonable to use Java Serialization as this is data that is specific to the User Application, not needed elsewhere in an open form, unlike the definitions of workflow. However there are two difficulties with Java Serialization, relating to classes that do not implement the Serializable interface:

  1. Shapes in the Java API do not implement the Serializable interface. This problem was overcome by looking at the other graphical packages mentioned previously to see how they overcame the problem. Since Shapes are aggregated instances in the VisualExpr decorator that decorates the expressions on screen, the standard technique of iterating over the points in the General Path of the Shape instance was used. The VisualExpr decorator includes the necessary writeObject and readObject methods to manage writing and reading the points of the Shape during Serialization and Deserialization, and the instances of the Shape in VisualExpr are made transient so the automatic Java Serialization knows they are handled manually.

  2. EParameter, which has ImportParameter and ExportParameter as descendents, inherits Parameter, a Bean in Apache SOAP. Parameter does not implement the Serializable interface, but the difficulty is unlike that of Shapes, as inheritance, rather than aggregation is involved. Whereas with Shapes the problem could be trapped to VisualExpr, and handled with the creation of the instances in that class, EParameters are used in several places across the entire system, and it is unfeasible to manually handle Serialization in all these places where instances of EParameter is created. However Java Documentation [JEX] notes that the Externizable interface offers a way to handle serialisation from within the Class that is to be serialised, so this route was taken. The Externizable interface entails that during creation of the object instance by Java Serialization, all the necessary interactions by superclasses and internal instances of classes is handled by the writeExternal and readExternal methods. Consequently, Java Serialization will not refer to the unserializable superclass of EParameter, with writeExternal and readExternal setting the relevant values required by the Parameter superclass.

- **Edit/Create Mode and Monitor Mode.** These two CheckBoxMenuItems serve to show and hide the Monitor JInternalFrame, and only one can be selected. Additionally, the MenuItem associated with Monitor Mode disables all Canvas buttons accept Select, and also disables editing of the Properties JInternalFrame. Switching back to the
Create/Edit Mode enables all the disabled functionality and hides the Monitor JInternalFrame.

- **New SafeFlow / CircusFlow** creates the appropriate new CanvasPanel instance within the Canvas JInternalFrame, via the CanvasManager. The appropriate expressions used with each CanvasPanel is also added or removed from the Canvas JInternalFrame’s JToolBar.

### 7.5 Interaction Via The Canvas

#### 7.5.1 Aim
Provide a simple and elegant way design for handling manipulation of the canvas and expressions drawn on to it.

#### 7.5.2 Derivation
Editing and manipulation of graphical canvases is a well-defined problem, with well-documented general solutions [GHJV], [PM00]. It is hence possible to apply some of these to arrive at a tailored solution for the User Application.

#### 7.5.3 User Interaction As States
The CanvasManager creates JButtons for each expression / functionality, residing on a JToolBar, and adds action listeners to them. The JToolBar sits adjacent to the CanvasPanel. The CanvasPanel can hold at any time one of three states that define the main functionality that occur as its listens for, and acts on, mouse actions that occur on the CanvasPanel. The CanvasPanelState object that CanvasPanel holds at any time to delagate mouse actions to is determined by the JButtons sitting on the JToolBar, which invoke a state changes on the CanvasPanel through the action listeners associated with them in the CanvasManager.
Figure 55: How the different toolbar selections affect the state of the CanvasPanel; the Drop State is used in combination with the Flyweight Pattern. This configuration of design in user interfaces is well known.

7.5.4 The CanvasPanel And The CanvasPanelStates

The CanvasPanel holds the top-level WorkflowService Bean for the workflow schema being created. The appropriate CanvasPanelState will traverse the WorkflowService and its constituent expressions, manipulating the workflow schema as appropriate. MouseMotion and MouseEvent actions are delegated to the CanvasPanelState held by CanvasPanel object.
In the CanvasPanelDropState, for adding expressions and deleting them:

- When a mousePressed event occurs the value of the flyweight string passed, indicating the button selected, is evaluated, and the appropriate add or delete function performed, based on VisualExpr and its related classes evaluation of the context of the action. The Properties JTable is updated with the new Drawable’sTableModel, via PropertiesManager singleton.

The CanvasPanelLinkState, for linking ImportParameters and ExportParameters:

- When a mousePressed event occurs the clicked VisualExpr source is evaluated to see if a link is allowed from this source.

- When a mouseDragged event occurs, if the source VisualExpr is not null, a probable line is painted, via CanvasPanel, from the centre of the source VisualExpr to the current mouse X and Y co-ordiantes.

- When a mouseReleased event occurs, if the source VisualExpr is not null, the sink VisualExpr, where the mouse currently is, is evaluated with the source to determine if a link is allowed, and if so, is set and drawn.
The CanvasPanelSelectState, for selecting, moving and resizing expressions:

- When a mousePressed event occurs the current mouse co-ordinates are saved as the ‘previous’ co-ordinates. If the cursor is near a VisualExpr edge, this is Resize mode. Otherwise it is Move mode.

- When a mouseDragged event occurs, if the source VisualExpr is not null, and if the cursor-action is set to Resize mode, the VisualExpr’s using the current and previous X and Y co-ordinates, via functionality available through VisualExpr. If this is Mode mode, the VisualExpr is moved instead.

7.5.5 VisualExprFactory
CanvasPanelDropState references the VisualExprFactory for adding expressions to the canvas and CanvasPanel references it for creating the WorkflowService top-level expression. VisualExprFactory returns a new Drawable instance with default values for their fields; generally null values. All AbstractExpression, ImportParameter and ExportParameter Beans implement the Drawable interface, which requires that they maintain a VisualExpr Bean. The VisualExpr Bean is also initialised, set with appropriate values. Generally all AbstractExpressions are set with rounded rectangle shapes, with ImportParameter and ExportParameters set with circles. Each type of Drawable is of a different colour, with Non-Terminals set with a mid range alpha transparency level.
7.6 Class Diagrams – Visual Manipulation On The Canvas

Figure 57: Summary class diagram of classes involved in visual manipulation on the canvas. Most of the methods’ functionality in VisualExpr are delegated to one of the classes it uses.

7.7 Decorating The Language Using VisualExpr

7.7.1 Aim

- To add the capability to the existing language data structure for visualising the workflow graphically.
7.7.2 Derivation

There are several types of objects that can be drawn on the canvas, ranging from AbstractExpressions like Web Services, which have no constituent expressions, to ImportParameters, which do not descend from AbstractExpression, but are distinct fields of AbstractExpressions, but need to be drawn as well. Given that a data structure of Beans already exists for the language, it is not desirable to duplicate it to simply create a version that includes the necessary graphical information. This would cause redundancy and inconsistency.

Consequently, decorating the language Beans seems the most appropriate solution, but this is slightly problematic. A single Decorator Class can be defined that decorates any kind of AbstractExpressions or EParameter, as the graphical information is relatively similar for all. However, it is necessary to traverse the hierarchy of language Beans during graphical operations, to add constituent expressions to the right Non-Terminals, for example, and while a Decorator can refer to a language Bean, and the language Bean to its constituents, it is necessary to derive the Decorator for the constituents also, for example when resizing an inner expression. It is necessary to use the Visitor Pattern style double dispatching between each object that can be drawn on the canvas and the Decorator Class that contains that object’s graphic information.

7.7.3 Managing Decorating

Each AbstractExpression and EParameter implements the Drawable interface. This interface requires that each implementing Bean provide methods for getting and setting a VisualExpr Decorator Class associated with it. On the WorkflowEngine side, these methods would not be used, and the VisualExpr field set to null. On the GUI side, the VisualExpr field is instantiated to the appropriate value by VisualExprFactory, which creates Drawables.

The VisualExpr Decorator Class does not itself ‘decorate’ each Drawable, instead delegating the necessary functions to other classes. Instead, it maintains references to the delegated classes and manages the graphical and language data associated with each Drawable. Because it contains links to the data structures that Drawables (like AbstractExpression and EParameter) contain, VisualExpr becomes the single point of contact on the User Application side for manipulating language or graphical data associated with the Drawable.

VisualExpr also acts as a broker for method calls to the delegated classes. Classes wishing to add, remove or perform any graphical / language manipulation function call VisualExpr, which then calls the appropriate delegated class. This enables the delegated classes to remain hidden, and for any aspect of the decorating classes to be easily changed, as changes only need to be reflected in VisualExpr; the class managing constraints on drawing, VisualExprFormatter, could be changed, for example, to another of similar functionality.
Figure 58: The VisualExpr and AbstractExpression (which implements Drawable) keep references to
each other. This enables the VisualExpr to decorate the AbstractExpression, as is typical of the
Decorator pattern (Left). Additionally, if the AbstractExpression it is decorating has child Expressions
which VisualExpr needs the (child) VisualExpr of, the VisualExpr can obtain the child VisualExpr from
the child AbstractExpression (Right).

Specifically, VisualExpr inherits its graphical data from AbstractVisual. AbstractVisual maintains the
GeneralPath, which will be set according to a shape, the outline Color, fill Color and Font. VisualExpr
also maintains methods for adding, removing and obtaining an enumeration of elements for constituent
AbstractExpressions, ImportParameters, ExportParameters and Guards. In the case of VisualExpr
decorating a Drawable that does not have one or more of these fields, the VisualExpr maintains an
empty data structure for each instead. Enumeration of an empty data structure has the same effect as
the Drawable not having the data structure, while maintaining a uniform interface for graphic
manipulation functions that use VisualExpr. Attempts to add or remove to Drawables values to fields
that they do not have is managed by VisualExprLogic, which ensures language rules are adhered to.

Each of the instances of the delegated decorator classes maintained by VisualExpr are singletons,
mainly because each of these classes encapsulate functionality only, so it is not necessary to create
duplicate instances for each VisualExpr that decorates each Drawable.

### 7.8 Enforcing Language Rules With VisualExprLogic

VisualExprLogic contains functionality for accessing and manipulating the hierarchy of expressions in
the schema hierarchy that starts from the top level WorkflowService field in CanvasPanel.

#### 7.8.1 Adding, Removing And Finding Drawables

The Drawable to be added to is evaluated against the Drawable its is being added in:

- Guards, ImportParameters and ExportParameters can only be added to
  AbstractExpressions

- FailureSeqs can only be added to WebServices

- Non-Terminals can only be added to other Non-Terminals or a WorkflowService

- Terminals can only be added to Non-Terminals

Subtleties are catered for, such as a SafeFlowService only having a one AbstractExpression, normally
a Non-Terminal, so its VisualExpr Decorator will only have one constituent AbstractExpression, as
required.
A getNode recursive function is used to traverse the WorkflowService from the CanvasPanel for locating a specific Drawable at a specific location on the CanvasPanel. When each Drawable in the hierarchy is inspected, the passed X and Y location is checked to see if the Drawable contains the co-ordinates, and then a recursive call on its children does the same. When a Drawable is reached that contains the X and Y co-ordinates, but none of its children do, this is returned by the function. The getNode function is used in several places, including add, remove and some graphical manipulation functions. With respect to remove, the returned Drawable is simply removed from the parent expression that holds it.

7.8.2 Links
An addLink function takes two EParameters, normally identified by CanvasPanelLinkState, and determines whether to create a link between them. This is decided by evaluating the two Eparameters, sink and source:

A. ImportParameter sinks and ImportParameter sources can have a link if the sink’s grand parent is the same as the source’s parent in the hierarchy of language expressions. This is the case of an expression importing a value that has been imported by its parent.

B. ImportParameter sinks and ExportParameter sources can have a link if they both have the same parent. This is the case of imported values passing straight through an expression to be exported.

C. ExportParameter sinks and ImportParameter sources can have a link if they both have the same grandparent. This is the case of an expression exporting a value to a neighbouring expression – both of these expressions, which the EParameters are in, must be within the same expression. Additionally, a link is not allowed between expressions in Concurrent and Bayesian grandparents, and in a Sequence grandparent, the parent expression must be in order: the source parent must be before the sink parent.

D. ExportParameter sinks and ExportParameter sources can have a link if the sink’s parent is the same as the source’s grandparent in the hierarchy of language expressions. This is the case of an expression exporting a value that has to also be exported by its parent.
Each link case and its visual form. Each EParameter resides in the AbstractExpression of the shape it is on the edge of. Hence the EParameter that is furthest left has Untitled 4 as parent, and for the next two EParameters to its right have Untitled 4 as grand parent.

If a link is allowed to be created, the sink EParameter has its reference field set to the source, and the CanvasPanel is requested to repaint.

### 7.9 Restricting Visual Interaction With VisualExprFormatter

VisualExprFormatter allows for the computation necessary for the visual manipulation of expressions on the canvas.

#### 7.9.1 EParameters Stick To Expression Boundaries

When a new ImportParameter or ExportParameter is created by a user clicking a VisualExpr on the CanvasPanel, it is automatically placed on the nearest edge of the VisualExpr. Consequent dragging of the EParameter will constrain movement to maintain the EParameter on the edge.

This effect is achieved by formatting the location of EParameters before they are set to a new location. The parent of the EParameter’s GeneralPath is obtained, and the all the points on the GeneralPath iterated through. The distance between the mouse cursor and a line segment between any two points on the GeneralPath is obtained. The point on the line segment the mouse cursor point is compared to is calculated using the Mathematical Method for Closest Point On Line, as described in [DVE00]. Of all the points that the mouse point is compared to, the point closest to the mouse point is set as the new location of the EParameter.

#### 7.9.2 Expressions Can Not Move Out Of Parent Expressions

All VisualExpr that decorate an AbstractExpression, as opposed to EParameter, have their location formatted based on the boundaries of their parent expression.

If the VisualExpr’s boundaries, possibly as dictated by after mouse drag, is beyond the boundary points of its parent, then the location of the VisualExpr is set to the parent’s boundary.
7.9.3 Other Visual Manipulations

Moving of Drawables is performed during mouse dragging in CanvasPanelSelectState, in drag cursor action mode, by taking the difference between the previous and current X and Y co-ordinates and changing the location of the Drawable, within the above formatting constraints, by the calculated difference.

Resizing of Drawables is performed also during mouse dragging in CanvasPanelSelectState, in resize cursor action mode, by taking the difference between the precious and current X and Y co-ordinates. The calculated difference is added to the height and width of the VisualExpr shape. This new calculated height and width are divided by the original height and width distances to obtain scaling ratios for an Affinetransformation matrix, which is applied to the GeneralPath of the VisualExpr shape, resulting in the resized shape.

When a Drawable with child expressions is resized smaller, all of its children will be resized also when the parent starts approaching the size of the children. After a limit, it is not possible to resize to a smaller size.

7.10 Painting And The Properties Window

7.10.1 Painting The Canvas

A Graphics visitor performs painting in Java. The Graphics object traverses the hierarchy of visual components, using each component’s paint method to visit, obtain the necessary data and draw on screen the component. The Graphics object is entirely responsible for painting and components that wish to be drawn specify, using the Graphic object’s draw methods, lines and other shapes they need drawn for their component to be visualised. The actual intricacies of drawing on screen are encapsulated in the Graphics object. This Java painting method is made use of by CanvasPanel, which overrides the default paint method of JPanel, which CanvasPanel descends from.

1. CanvasPanel first checks the size of the WorkflowService expression on screen, which all other Drawables will be within, and if this is bigger than the size of the CanvasPanel, the size of CavasPanel is resized appropriately, with some padding. This has the effect of changing the size of the viewport of the JScrollPane CanvasPanel resides in, rather than resizes the window automatically.

2. JPanel’s paint method is called to draw the CanvasPanel, via super.

3. CanvasPanel then obtains the VisualExpr decorator of the WorkflowService it holds. The VisualExpr’s paint method is invoked, passing the Graphics object. VisualExpr’s paint method then passes itself, the VisualExpr, to the VisualExprPainter singleton, which draws the VisualExpr’s stored graphic details using the Graphics object – completing the visitation process.

4. WorkflowService’s VisualExpr’s paint method then iterates through the ImportParameters and ExportParameters it contains to see if any have Link references. If any do, the source of the Link reference is obtained and the Graphics
object used to draw a line from the source’s VisualExpr’s centre to sink’s VisualExpr’s centre, using the default line graphic data in VisualLinkPainter.

5. WorkflowService’s VisualExpr then invokes the paint method on all its children Drawable’s VisualExpr. All the Drawables under WorkflowService are consequently recursively drawn, as well as any necessary Links.

Link lines are not actually maintained in a data structure, but are calculated during painting, as shown in step 4, so one neatly unified data structure, the workflow schema, and any necessary decorations are all that are required.

7.10.2 The Properties Table

Data in JTables can be defined either by setting them explicitly with data, such as from a Vector, or setting them with an object that implements the TableModel interface, which defines methods for setting and returning values at specific cells in a JTables. VisualExprTableModel extends the default implementation of TableModel, AbstractTableModel, and is the TableModel set with the JTable in the Properties Window.

VisualExprTableModel is a singleton that has aggregated with a VisualExpr. Normally the exact VisualExpr it holds a reference to is determined by one of the CanvasPanelStates as mouse interaction takes place.

Rather than defining another data structure, VisualExprTableModel projects the necessary information required by the Properties JTable, through the TableModel methods that indirectly manipulates the data in the VisualExpr. In this way the single unified workflow schema data structure is persevered.

7.11 Monitoring

7.11.1 Aim

- Provide a simple way to observe the enactment of the workflow, and its results.

7.11.2 Derivation

To provide for greatest association of the workflow created and its consequent enactment, the existing view of the workflow should be used to show the enactment. In the same way London Underground’s central monitoring system, or any similar monitoring system, presents the view of the elements in the structure, and lights up the active parts, a similar view is preferable for the workflow. The output of the workflow itself could be HTML web content, as there are many web services that provide material in this form. Rather than present a table of result values, this could be presented in a form similar to a Web Browser, and the rest of the Monitor JInternalFrame fitted around this idea.
7.11.3 Overview

The MonitorManager initialises the Monitor JInternalFrame with the appropriate composition for graphical components for display. The Monitor JInternalFrame presents four functions, each invoked by the appropriate button in the display window.

- **Deploy** invokes the WorkflowEngine, deploying the current workflow schema on screen. A Boolean value is returned from the WorkflowEngine as to the success of this operation, which is translated to a message displayed on the Monitor's output window.

- **Undeploy** removes the current workflow from the WorkflowEngine. The returned Boolean is translated to a message on the Monitor's output window.

- **Invoke** causes the WorkflowEngine to enact the workflow, and consequently receives continuous information on the progress of the enactment.

- **Query** returns a Boolean value, translated as appropriate the output window, indicating whether the workflow is deployed.

Figure 61: The Passed Parameters display of the Monitor Browser

<table>
<thead>
<tr>
<th>Passed Parameters</th>
<th>Returned Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Type</td>
</tr>
<tr>
<td>data</td>
<td>string</td>
</tr>
</tbody>
</table>

Add Parameter | Remove Parameter

Figure 62: The Return Value output display of the Monitor Browser

<table>
<thead>
<tr>
<th>Passed Parameters</th>
<th>Returned Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bonjour monde</td>
<td></td>
</tr>
</tbody>
</table>

Add Parameter | Remove Parameter
7.11.4 Invocation And Updating Process

The Parameters used to invoke a given workflow are entered in JTable that shares a JTabbedPane with the output window. Once the workflow has been invoked with the given Parameters in the JTable, the output is consequently displayed in the output window.

The Parameters JTable is implemented with a TableModel, MonitorParamTableModel, which holds a vector of EParameters. When either the ‘Add Parameter’ or ‘Remove Parameter’ buttons are invoked, a new EParameter is added or removed. Each cell item equates to a field of a particular EParameter. When the workflow invoked, the Vector of EParameters is fetched from the MonitorParamTableModel, and an EMethod object is created, which is sent to the WorkflowEngine asynchronously to invoke the workflow.

Consequent to this, empty EMethods with just the identity of the workflow are sent to the update method of the WorkflowEngine, which returns a list of active expression. This list is passed to the VisualExprPainter, and a special monitor repaint called on the CanvasPanel, so VisualExprPainter only repaints expressions that transcend to and from active states.

The final update call is indicated to the User Application by a special tag, and the result of the workflow is given instead, which is displayed in the output window.
8 Evaluation

8.1 Chapter Summary

- The workflow language is evaluated against existing languages.
- Design and implementation of other parts of the system are evaluated in terms of functionality, design elegance, flexibility, usability and other qualitative metrics.

8.2 The Language

8.2.1 Workflow Languages

There are many workflow languages in existence, and there is no consensus to one single widely preferred language. Within the context of distributed systems, OTSArjuna, METEOR, RainMan and WSFL are the primary candidates for benchmarking against.

8.2.2 The SafeFlow Dialect

*Programming problem versus graph problem.* SafeFlow's attention is in that it takes a programmatic approach to designing workflow. Whereas all the above-mentioned workflow languages approach workflow as a graph problem, SafeFlow considers an approach motivated by encapsulation and tight control over the enactment of the workflow. This is due largely to the language's origins from the Interpreter Pattern, which is an approach not considered in the approaches to designing other languages. Although programming may be a disguised graph problem, it offers constructs and structures that reduce complexity and improve manageability.

*Control graphs as goto statements.* WSFL and OTSArjuna are the strongest related to graph-problems. The former lays control out as a graph and overlays a data flow graph above it that respects the control graph. OTSArjuna, METEOR and RainMan mix both control and data flow in the same graph. The argument against graph based approaches is that of maintainability and problem structuring – a graph of control lines is analogous to ‘goto’ statements. In contrast, the arguments for control to be represented as a graph is that of high flexibility and expressiveness. OTSArjuna considers intermediate Composite Tasks as a structuring mechanism, creating sub-graphs that can be represented as nodes themselves in other graphs. However, the underlying approach is still that of a graph problem. WSFL allows for deployed workflow to be reused as Web Services in another workflow schema, but there is no structure presented within the schema.

*Control as encapsulation.* SafeFlow does not use graphs, but instead recursively encapsulated control boxes, specifically Sequence and Concurrent, which dictate the control within the box. Data flow is represented as Link lines between these boxes. The significant advantage of such an approach is that any encapsulating box can be easily removed and replaced without effect to the rest of the workflow.
The cost of SafeFlow. The disadvantage of the approach is that of restrictions in flexibility and expressiveness. In particular, the following graph cannot be represented in SafeFlow:

![Figure 64: A graph-based workflow that cannot immediately be described in SafeFlow](image)

The cause of the problem is due to the synchronisation of the Concurrent Expression, as described in the above figure. If two expressions, B and D, are waiting for a Concurrent Expression that contains A and C, but D only requires C to finish, while B requires both A and C to finish, then both the waiting expressions must wait for both their predecessors to finish. In Java, the lack of Multiple Inheritances is presented as a 'feature' for better programming. While a similar case could be arguable here, the above scenario is too problematic in causing restriction. CircusFlow overcomes this restriction, and in a similar way to that in which Java presents Interfaces as a work-around for Multiple Inheritance, a solution to the problem combining CircusFlow and SafeFlow is given in the next section.

Scalability Versus Bureaucracy. SafeFlow allows for high scalability, due to the natural presence of recursive encapsulation. Thousands of nodes of Web Services, or other distributed component, could be organised effectively using SafeFlow. Subsections within the workflow can also be effectively reused, or the entire workflow itself. However, this also causes bureaucracy, as programming intricate subsections within subsections increases time and effort required to create the workflow. This is a problem that can be overcome through the use of workflow creation tools that reduce the effort of creating workflow definitions. An example task that may take some time is creating links between different EParameters, traversing different Expression; the User Application allows for this visually through simple drawing of a line.

Top down workflow creation, B Method style. The process of creating workflow with the language is not one that has been much thought by other workflow language creators, though the popularity of many programming languages is due how the language is used to perform the desired function. For SafeFlow, a top-down methodology is proposed, similar to the way in which B is used, though the proof of correctness aspect of B is not entirely respected. A process of refinement of the workflow from a high level allows for a more structured approach to creating workflow, with the workflow treated more as a specification of agreement between Web Services. Bottom level refinements are implementations, Web Services, and an implementation can be an existing workflow. The problem with using this design approach is that creating workflow quickly may be difficult – this is an approach for designing more stable workflow. RainMan considers the quick creation of small-scale workflow for the Internet, and in this respect has contrasting aims and approach to creating workflow. However, because SafeFlow was designed in terms of hierarchical decomposition enabling outsourcing of parts of the workflow to Web Services representing other workflow, which is possible in an Internet setting, creating large workflow will be faster with SafeFlow. CircusFlow has no hierarchical decomposition, so will be able to match RainMan for creating small-scale workflow.
8.2.3 The CircusFlow Dialect

A graph with no control. CircusFlow is a contrast to SafeFlow, in having minimal explicit control, concentrating on the data flow. CircusFlow is also graph based like the other languages considered. It has greatest similarity to OTSArjuna and Dennis data flow [WAN99], because its waits for data to be available before enacting the next task. The main difference of interest is that CircusFlow restricts data through synchronisation to achieve some level of ordering of expressions, while others will use some form of control flow. The differences between CircusFlow and existing languages are not significant; CircusFlow has the same strength and weaknesses of graph based approaches described earlier in the previous section.

High Parallelism. CircusFlow allows for high parallelism by ignoring control flow entirely – if a node can execute, and has sufficient information to do so, it will do; the workflow is enacted faster. The drawback is that in some cases control is desirable.

Combining CircusFlow and DataFlow. Continuing the Java analogy of Multiple Inheritance, to overcome the synchronisation problem identified in SafeFlow, it is possible to contain the part of the workflow that holds the nodes where one expression is waiting for a sub set of another’s nodes in a CircusFlow workflow. This workflow can then be deployed as a Web Service and included in a SafeFlow workflow. This solution enables the control and encapsulation aspects of SafeFlow to be preserved, while containing the uncontrolled part of the workflow within a Web Service. CircusFlow is not itself an expression in SafeFlow because then it would allow for having one top-level CircusFlow, within which an entirely unstructured workflow could be created, defeating the motivation for SafeFlow.

However, just as in Java, Interfaces are not a complete solution to Multiple Inheritance, overcoming the synchronisation problem with this technique is not a complete answer. Synchronisation will need to occur at some point, at the end of the perimeter of CircusFlow workflow, before execution can return to the structured SafeFlow. This will only be a significant problem if one of the expressions at the end will take a significant amount time. In this case, it is better to not wait for a return value, idling, and to accept the value from that Web Service at a later time, perhaps as a single call outside of any workflow environment.

8.3 The System And Its Design

8.3.1 General Architecture

Low Coupling Versus Practicality. The strength of the architecture is that of creating easily reusable workflow, as they are deployed as easily accessible method calls. This means the workflow will be backwardly compatible, in that it can also be used with all future and new workflow languages. This has benefits in saving time and effort in otherwise porting existing functionality to another language. Creating new value-added Web Services from existing Web Services is also a significant benefit. The drawback is that of high initial overhead. A server is needed to be running with the WorkflowEngine installed in order to enact any of the workflow, and Internet trends are not yet at the stage where servers are easily available for deploying servlets on to. This also makes using the User Application to create workflow immediately upon receiving the software difficult. Although this is a problem, this is due to the design aim of the Workflow Management System being to create new Web Services, which intrinsically require a server. Making the modification to allow the User Application itself to enact
Workflow is not difficult, as the WorkflowEngine could be accessed directly locally rather than over SOAP. WSFL is the only other workflow language that allows for encapsulation of a workflow as a Web Service, though as it is only a discussion paper at the moment, with no implementation, it is not certain if the same technique of encapsulating as method calls will be used, or another alternative, like creating an entirely new Web Service for the workflow. The former is lower overhead and is used in this system.

**Language: Structured And Extensible.** The structure of the language in terms of Beans descending from an AbstractExpression was thought out carefully, and the result is an extensible and flexible implementation. Some of the expressions in the language were added after the initial implementation. This was possible due to the backward compatibility of the language, in simply being able to add new Beans for new Expressions and modifying the Interpreter(s) appropriately; entire new language dialects can be implemented with new Interpreters. Incremental evolution of the language in this way is possible because the impact is minimal. Alternative implementations may have included the functionality of expressions in the Beans, in which case all affected Beans would need to be modified. The XML-Bean conversion process, which is independent of the Beans and XML, further improves the ease of adding extensions. The only difficulty would be in making changes to the language to such an extent that some of the common features encapsulated in AbstractExpression are not desirable of all expressions. However, since AbstractExpression defines only the fields and function that are basic properties of the language, the designer may desire a fundamentally different language.

**Double-dispatch decorating for trees.** The additional information required by the User Application for visualisation of the schema is achieved through double dispatching, a technique largely used by visitations of a Visitor. In this scenario, each expression Bean implements the Drawable interface, which enables a field to be set in the expression Bean for a class that descends from VisualExpr, and VisualExpr itself maintains a field referencing the expression Bean.

In this way, the VisualExpr can add more data to the expression Bean, without adding fields to the Bean itself, and it is possible to access the children of the expression Bean through the VisualExpr decorator, which is not possible with black box decorating. In contrast to the strength of being able to add extra data to a data structure without changing it or creating a redundant duplicate with the extra information, essentially maintaining only one data structure, the drawback is that of added complexity of the data structure. However, the use of one unified data structure with many possible views is the central thesis of databases.
8.3.2 The WorkflowEngine

Concurrent Visitors – An Advantageous Implementation. The strength of the Concurrent Visitor is in being able to evaluate several nodes concurrently, rather than traversing a data structure linearly. This enables for more complex processing of information, while maintaining [GHJV]'s aim for the pattern in adding functionality to a class without modifying it. Concurrent Visitors takes an approach that floods parts of the data structure with Visitors, achieving co-ordination between them using a Publisher-Subscriber protocol. Despite the benefits of separating functionality and data to improve maintainability, and achieving more complex computation, adding concurrency to Visitors increases the barrier to understanding the program for someone unaware of this implementation. However, the implementation is should be comprehensible to anyone who understands the Visitor pattern, as the increased complexity over the standard implementation is slight. As with any Concurrent set of processes, Concurrent Visitors also makes tracing the execution of the program difficult, and the implementation needs to carefully consider avoidance of and overcoming possible deadlock; this implementation does.

Monitoring Mechanism And Scaleability. The mechanism used for monitoring does not scale, as the MonitorPost entity that Interpreters register with as they enter and leave expressions is a singleton. However, the design case considered is that the WorkflowEngine typically represents one Web Service that encapsulates workflow, so in the normal case should only ever need to be monitored by one or two User Applications; even for two User Applications to manage the WorkflowEngine could cause problems if they are no co-ordinated. If more WorkflowEngine Web Services are required, they can be deployed at different URLs, and managed separately; each will have its own MonitorPost.

However, for the simple use of only one or two User Applications enacting workflow, this implementation is sufficient.

8.3.3 Probabilistic Method Calls

Automation of heuristic choices. Probabilistic Method Calls are useful for applications that are largely not critical, as intrinsically there is likely to be some level of error involved, even if very small. The idea is most useful where a complex heuristic choice needs to be made. Examples of this could include delegation of processing to one of several entities, and analysis of the most appropriate entity based on measures of the entities’ suitability. Probabilistic determination of this kind is useful in workflow because it may remove heuristic decision-making that may otherwise require human intervention. A weakness of Probabilistic Method Calls is where the Bayesian model may change over time, which this implementation does not account for, because accounting for learning within the WFMS system would be significant undertaking in itself.

Implementation Issues. There is some contention as to whether a Naïve Bayesian Network is sufficient for all purposes, and it is possible to implement a more advanced system with more complex multi-level networks, which may improve accuracy. More thought needs to be given in representing the information at the user-interface side, as at the moment the user simply types in a string to represent all the conditional probabilities and other information. It is also necessary to calculate the conditional probabilities before hand, which may not be convenient for some users. The implementation is basic, but allows for any number of child inputs, a way to quantise the input, and any number of parent possible methods, which are the critical components to any implementation, and which are sufficiently presented here.
8.3.4 The User Application

The user interfaces used to create workflow in other workflow systems were not easily available.

Visually Impressive. Most who have used the User Application comment on its positive visual impression, which is possibly due to the JInternalFrames used with the application, which is not often used in Java applets or applications. With the properties table, and a canvas, the effect does give something of the feel of a cut down IDE, which is the desired effect, as it gives the user a more intuitive way of manipulating the workflow than otherwise.

Prior Knowledge Required. The User Application still requires, however, that the user have some knowledge of the visual semantics of the language. The user will need to know data cannot be passed between Concurrent Expressions, for example, and will otherwise become frustrated. To some extent this is inevitable, as all programming-style languages have their intricacies that users need to become adjusted to. An improvement would be better visual cues and online help.

Debugging difficult. The nature of the workflow problem makes debugging workflow difficult. When enacting a workflow and the result is not as desired, it is difficult to know how or what is the problem, since all the main functionality is encapsulated and distributed across different Web Services. Adding the necessary support to make debugging and error handling a pleasurable experience is itself a significant task, and was not considered beyond any basic detail.

Proven Design That Works. Much of the internal design of the User Application is based on well-documented uses of the State and other patterns, and the design does not overly deviate from these, as the result is maintainable and extensible, with little redundant code. One possible criticism could be that there are too many singletons, but these are justified in that they represent parts of the system that there should only ever be one instance of, such as a Menu bar. If necessary, it is possible to extend some of these singletons by replacing them with publishers and subscribers.

8.4 Other Issues

Testing the Bayesian function is the only real test issue, and is given in the Appendix. Incremental testing was used during development of the system. Because of the use of patterns, like the Visitor pattern for Interpreting, and the minimisation of redundant code, there is less code to test than might otherwise be the case. The WorkflowEngine was developed first. As it encapsulates workflow functionality of the system, and can be used independently of the User Application, being passed XML workflow definitions in string as appropriate, it was incrementally tested as it was built. Since the WorkflowEngine is less than 1000 lines of code, with a significant portion being code for Beans with no actual functionality, problems could be resolved quickly and at a single point – the Interpreters. Some of the User Application code was generated by an IDE, which produced correct code, and most other code was trivial to test, such as ensuring when a button is toggled, all other buttons are become untoggled. VisualExprLogic, for ensuring schema is built according to language rules, and VisualExprFormatter, for ensuring visual manipulation is restricted as required, such as EParameters sticking to VisualExpr edges, were the most difficult to test. However, both of these segments of code functionality were encapsulated within their own classes, reducing the effort for ensuring correctness.
Performance measures are difficult with this work due to its nature. It is possible to measure the time taken to complete a workflow enactment, but this would simply be the sum of the network response time to accessing several Web Services, and is more related to the network and server performance than that of the Workflow Management System. The overhead time of the WFMS is very small to that of the overhead of the Web Services used in a workflow; the choice of Web Services used is determined by the user.

Single Point Of Failure. Due to Web Services using HTTP, it is not possible for callbacks, and it is necessary for the User Application to act as an intermediate between all Web Services, in that it invokes a call, receives the output and then send this output to another Web Service, rather than the Web Services communicating directly. Irrespectively, an Active Workflow Engine of this type is necessary if there is to be monitoring of the progress of the workflow made available to the User Application.

Detecting deadlock and starvation in the workflow is difficult in some cases, such as SafeFlow, because of encapsulation. Two Web Services may be waiting for each other, but at a high level it is not possible to observe this, as the deadlock could be hidden in one or more encapsulations.

Program Analysis techniques such as control flow and data flow analysis may provide more insight into workflow languages, but time and lack of experience with this area prevented this.
9 Conclusion

9.1 Summary Conclusions

\textit{Workflow encapsulation is a powerful structuring mechanism}. Encapsulation of control in a workflow provides a way of structuring complexity that would otherwise be difficult with standard graph approaches. Because each constituent part in a workflow can be closed as a black box, either using a Non-Terminal expression or by outsourcing to another Web Service that encapsulates workflow, it is possible to break up a large complex problem with top down refinement into relevant sub components. As in programming, if these sub-components are highly cohesive in terms of the functionality they perform, during maintenance they can be more easily modified. This is a problem with current approaches, which produce large graphs, with lines that are analogous to goto statements; changing a node without affecting others would be difficult.

\textit{Building Web Services from workflow over Web Services - speed of development versus flexibility}. The advantages and disadvantages of creating Web Services from a workflow of other Web Services are similar to that of any Component Based Development method. Development is faster because whole parts of the workflow can be outsourced to other Web Services that themselves may be encapsulated workflow. However, if Web Services cannot be found that meet all required requirements, it will be necessary to create proprietary Web Services to overcome difficulties. The interesting difference between traditional Component Based Development and workflow over Web Services is that the former has been problematic because of the lack of availability of components, while the Internet setting of Web Services overcomes this. There are already directory listings of freely available Web Services, such as \url{http://www.xmethods.com/}.

\textit{Concurrent Visitors – complex data manipulation}. The standard Visitor pattern implementation is suited for singular traversal of a data structure. However, there are situations where it is necessary to compare or perform a calculation in two different parts of a data structure before continuing traversal from one. In such a case, the use of one Visitor that moves back and forth from the two points of interest can make the code for traversal unnecessarily complex. Once the implementation is understood, in situations such as this, or where Concurrency is needed, the use of multiple co-ordinated Visitors can be a significant aid to an otherwise complex computation.

\textit{Probabilistic Method Calls add heuristic choices}. Rather than invoking a method through a systematic mechanism where the exact method call cannot be determined precisely, because probability needs to be factored in, Probabilistic Method Calls can be used. A systematic choice mechanism that uses if-then-style selection statements cannot model conditional probabilities for an invocation, such as there being 60\% probability for a particular method invocation, with the probability changing in quantity by factoring in other data, not all of which affect the probability to the same proportion. Probabilistic Method Calls can automate heuristic decisions that would otherwise be difficult to determine, particularly through systematic methods.
9.2 Insights And Possibilities

9.2.1 Insights

**Good design as efficiency constraint.** In the past, programmers were faced with limited memory and disk size. In this environment, techniques were used that would be deplored in modern times, though they would have been held in wonder in the past – fitting large amounts of functionality within these tight constraints. Upon completing this project, it is clear now that in the modern environment, with ample disk and memory space, as opposed to having more freedom of design and implementation, the efficiency constraint now is good, maintainable design. There are techniques developed to this endeavour, such as Refactoring and Design Patterns, and these do not require any less skill or experience to apply then those of old. The analysis of problems and identification of objects and responsibilities in such a way that their integrity is preserved through the software’s life cycle, to improve the software development efficiency in maintenance, is the goal of current programming.

**New problems, old solutions.** The Problem Frames [MJ] idea of analogising problems to other problems with well-defined solutions is taken as an approach to software development, but can equally be applied to research. Such an approach shows reoccurring basic problems that have good solutions in other parts of computing, such that attempting a new solution from scratch is not necessary. This project, through use of a structured programming approach to workflow, gives an example of this.

9.2.2 Possibilities

**Probabilistic Distributed Computing** could hold interesting potential, in its approach to solving problems probabilistically, rather than through a computationally intensive procedure which itself may not result in the correct answer. An example might be to consider a distributed component where the precise method calls are not known, but the meaning of the component, as a Pizza Retail, for example, are. In such a case, attempting to invoke a non-existent function ‘order’ with a parameter ‘Pizza’ could be matched to the real function ‘orderPizza’, because the system has learnt this pattern from previous method call attempts, and has probabilistically determined the correct call. The issue of the threshold of allowable errors will always be a problem.

**Workflow As Specification** could further extend the B Method analogy and look at creating proofs for workflow and consequent refinements. Issues will include proving the correctness of components that are imported across a network, especially if quality of service is involved. A certification or trust model is possible whereby some Web Service components are taken as more correct, with respect to what they assure, than others.

**Minor Extensions** include investigation into providing better debug support, as well as allowing the User Application to enact workflow. Certain aspects of the User Interface can also be fine tuned, and features added, such as a copy and paste of visual expressions.

9.3 Pseudo-Structured Workflow – Further Work

9.3.1 Aim

- To combine aspects of both SafeFlow and CircusFlow.
9.3.2 The Workflow Model

In this workflow model, Non-Terminal control expressions do not synchronise data as it enters and leaves the Non-Terminal expression boundaries. In a Concurrent Expression, any Web Service expression that has sufficient data to enact will do so, and for any other inner Non-Terminal Expressions data will be passed straight through the boundaries of that expression to any Web Services that need them. Concurrent Expressions are similar to CircusFlow, but the boundaries for intermediate composites like Concurrent are not synchronising. This creates a workflow model that is like a graph of data, but where sub graphs can be perimtered into sections. The flexibility of the sub-graph does not change, though this allows the sub-graph to be encapsulated, and replaced with the same endpoints of the sub-graph.

Sequence expressions force ordering on the constituent expressions, in that if the next constituent expression has sufficient data to execute, it will do so, and if it does not, blocking will occur until more data passes into the Sequence. Any output data from constituent expressions are passed straight through. Bayesian Expressions are not immediately possible, as the probability of the constituent expression to be invoked will change as new data becomes available to the expression.

9.3.3 Assessment Of This Model

At first glance, the encapsulation of the sub-graph or constituent expression may not seem to be truly black box, as changing a constituent expression’s internal behaviour may change the order and way in which other parts of the graph behave. However, this is also true of programs, as changing an object that implements an interface with that of another may result in the new object returning new results. However, in the case of this workflow model, since there is not synchronisation of data at the boundaries, ordering of output could also change, though the affected components will not. Non-synchronising borders of intermediate expressions are not found in any available workflow languages.

9.4 Bullets Over Broadband – Final Word

“There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.”

‘No Silver Bullet – Essence And Accident In Software’, F. P. Brooks [FPB95]

9.4.1 Components

Brad Cox, who in 1986 invented the idea “software integrated circuits” [BC86], is credited with the notion of inventing saleable ‘circuits’ – components – that can be reused and plugged into new pieces of software. Cox argues that component based encapsulation and reuse promises an order of magnitude improvement in software development [BC90], and in a later paper, also featured in [FPB95], Brooks sights that this is a good attack on the conceptual essence of the problem.

Web Services are components distributed over the Internet for which there is growing industrial support among opinion formers, such as Microsoft, IBM, HP and Sun. Web Services may potentially become the de facto component model for distributed computing, and even if this is not the case, there is some consensus that an open standards based similar model will be necessary.
How these Web Services components are accessed and combined for use in other Web Services is an inevitable issue. Current popular mainstream programming languages may be deficient, because they are inherently linear. Although the creation of new threads is possible, they do not provide a simple way to traverse several Web Services in parallel – each invocation would be in sequence and returned values handled in sequence before sequentially reinvoking other Web Services for parallel computation. To this problem, there are two possible immediate solutions. The first is languages based on the Actors Model, which have had limited success as a basis for reasoning about the properties of programs [ON]. A second solution is workflow, which offers a solution because of the concurrent nature of workflow languages.

However, because workflow has descended from an organisational management background, the automation of which has resulted in a niche interest in parts of distributed computing, workflow languages lack the structures and models that programming languages have evolved to. In consequence, in some senses some workflow languages have analogies to incorporating goto statements. This project has investigated adding structure and approaching workflow as a programming problem; the SafeFlow language described in this thesis can be reasoned for workflow properties.

9.4.2 Engineering Web Services From Workflow

If the Web Services that are to be created from a workflow over other Web Services results in a stable system, that will not undergo change, then current unstructured approaches are satisfactory. However, from Software Engineering it is taken that this will not be the case. Unless the changes are minor, such as replacement of a single Web Service, unstructured approaches are fine. However, if significant changes need to be made, because the business logic has changed due to a merger, for example, then it is necessary to corner the affected area and rework as necessary, hoping not to cause changes elsewhere. If the workflow has been designed in a way that encapsulates related responsibilities and functions, then this will be an easier task, as is the case with well-structured traditional programs.

However, it is clear that creating workflow is not easily a problem that is resolved by traditional functional, imperative, declarative or object-orientated approaches. However, the chosen technique may take inspiration from these, as has been the case from this work. The reason existing approaches cannot be taken as a whole is due to the concurrent nature of workflow, which requires the handling of several operations concurrently. In this endeavour, the approaches used to highly parallel supercomputers [WAN99], starting from Dennis data flow, as well as the Actors Model and process calculi may offer hints of a method.

9.4.3 Applications That Are Web Service Workflow

In engineering Web Services from workflow, there will inevitably be many of the same problems inherent in COTS based development. It is unlikely that all of the required functionality of the application can be found by simply gluing together Web Services to create a new Web Service, and there are cases where a Web Service almost performs the required function. To overcome this scenario Web Service development environments will need to allow for the creation of local components that can act as intermediates between two Web Services. In this respect, development of GUIs using IDEs like Visual Forte or Borland JBuilder already gives a glimpse of visual programming using well-defined components, with extra code to enable their interaction. It is not difficult to see the
emergence of standard Web Services offering credit card handling, banking and delivery, which can be used to create an online retail Web Service, and there is no reason why these standard Web Services cannot be visualised graphically and composed to build the required application.

Some evangelists propose that all devices will at some point be accessible as a Web Service [COOL]. If this were to be the case it is possible that CD players and other devices could be invoked with probabilistic determination of the exact parameters, particularly where some small level of randomness is not critical.

The greatest problem from Web Services created through workflow is that of wait times. An application made of several Web Services distributed across the world, which themselves may be composed of other Web Services, may cause long wait times for the user of the front-end application, due to the sum of the delays on all the connections. In this respect, one answer may be the increasing availability of, and improvement in, broadband technology. In a world where connections between devices are increasing in bandwidth and speed, wait times can be mitigated.

### 9.4.4 In Summary
High level programming of components promises an order of magnitude improvement in Software Engineering processes. Workflow over Web Services may fulfil that promise.

### 9.4.5 Prediction
*Within a decade*

- Programming languages will, instead of importing locally packaged classes and functions, refer to functions in components distributed across the Internet, such as Web Services.

- These languages will feature a more inherent level of concurrency than those of the current time, similar to that present in workflow languages, while keeping familiar organising constructs.

- IDEs supporting these languages allow for the visual manipulation of these components.

- Probabilistic determination and invocation of inconsequential functions will be incorporated in applications.
X1 Bibliography

X1.1 Sources

All sources are legitimate, coming from papers from universities and companies working on the technology referred to technical journals.


COOL The Cool Town Project at Hewlett-Packard http://cooltown.hp.com/

DFG,L2 Duncan Gillies, Intelligent Data and Probabilistic Inference Lecture 2 (October 2000) Imperial College http://www.doc.ic.ac.uk/~dfg/ProbabilisticInference/IDAPILecture02.PDF

DFG,TUT1 Duncan Gillies, Intelligent Data and Probabilistic Inference Tutorial 2, Q3 (October 2000) Imperial College http://www.doc.ic.ac.uk/~dfg/ProbabilisticInference/IDAPITutorial01.pdf http://www.doc.ic.ac.uk/~dfg/ProbabilisticInference/IDAPITutorialSolution01.pdf


DWAD00 David Webber and Anthony Dutton, ‘Understanding ebXML, UDDI and XML/edi.’,
XML.org

Addison-Wesley Publishing Company

GFC Graph Foundation Classes for Java (July 2000), Alphaworks, IBM
http://www.alphaworks.ibm.com/tech/gfc

GG00I Graham Glass, ‘The Web services (r)evolution: Applying Web Services To Applications’ (Nov 2000), IBM


GHJV Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, ‘Design Patterns’ 1994, Addison Wesley Longman Publishing

HP00H-1 ‘E-Speak Service Framework Specification’, Hewlett-Packard

HP00H-2 ‘E-Speak Service Engine’, Hewlett-Packard

HP00H-3 ‘E-Speak Fundamentals’, Hewlett-Packard
http://www.e-speak.net/features/module2/slide1.html

IBM00I The IBM Web Services Architecture Team, ‘Web Services Architecture Overview’ (Sept 2000), IBM

IBMWFA ‘e-business Process Automation’, IBM MQSeries Workflow

JEX ‘Providing Object Serialization for Your Classes’, The Java Tutorial

JMIBM John Mann, ‘Workflow and Enterprise Application Integration’, eBiz
http://eai.ebizq.net/workflow/approaches_to_EAI_2.html

KT96B Kelly Trammell, ‘Work Flow Without Fear’ (1996), Byte
http://www.byte.com/art/9604/sec8/art1.htm#046f1a2

MET METEOR2 Project at University of Georgia
http://lsdis.cs.uga.edu/proj/meteor/meteor.html

Addison-Wesley Publishing Company

Addison Wesley Professional

MK00M Mary Kirtland, ‘The Programmable Web: Web Services Provides Building Blocks for the Microsoft .NET Framework’ (Sept 2000)

MS00M  ‘Web Services Security’ (Nov 2000), Microsoft
http://msdn.microsoft.com/vstudio/nextgen/technology/security.asp

http://www.jeffsutherland.org/oopsla98/mts.html

http://micro-workflow.com/PhDThesis/phdthesis.pdf

OJG  Jesus M. Salvo Jr., ‘OpenJGraph - Java Graph and Graph Drawing Project’
(March 2001) http://openigraph.sourceforge.net/

OMGWM  ‘Workflow Management Specification’, Object Management Group
http://www.omg.org/technology/documents/formal/workflow_management.htm

ON  Prof. Oscar Nierstrasz, ‘Composing Active Objects’ (2001)
University of Berne
http://www.iam.unibe.ch/~scg/Archive/NFS/cao.html#STATE

ONE01  ‘Open Net Environment (ONE) Software Architecture’ (2001)
Sun Microsystems
http://www.sun.com/software/sunone/wp-arch/$sessionid$FW1ROZYAACEJDAMTA1FU45Q#openarch

PL00M  Pete Loshin (Dec 1999), ‘Web Services and the Simple Object Access Protocol’, Microsoft
http://msdn.microsoft.com/xml/general/soap_webserv.asp

PM00  Paul Marson, ‘Design of the User Interface (UI)’ (2000), Oxford University
http://users.ox.ac.uk/~kebl0885/3rd_report/Report%20Chapter%203%20-%20UI%20design.htm

PPCI  Santanu Paul, Edwin Park, and Jarir Chaar, ‘RainMan: A Workflow System for the Internet’, IBM T. J. Watson Research Center

http://www.systemsguild.com/GuildSite/Robs/Template.html

John Wiley

SB00I  Steve Burbeck, ‘The Tao Of E-Business Services’ (Oct 2000), IBM

SBSG00I  Steve Burbeck and Steve Graham, ‘Creating target-rich environments in a service-oriented architecture’ (Dec 2000), IBM

SOAP11  ‘Simple Object Access Protocol (SOAP) 1.1’, W3C Note 08 May 2000
http://www.w3.org/TR/SOAP/

Core Java Platform Group Java Software, Sun Microsystems, Inc.
http://java.sun.com/xml/jaxp-docs-1.0.1/docs/bind.pdf

UDDIEW ‘UDDI Technical White Paper’ (Sept 2000), Universal Description, Discovery and Integration

UO01 Uche Ogbi, ‘Using WSDL In SOAP Applications’ (Nov 2000), IBM

W1SRC Frédéric Ranno, Santosh K. Shrivastava, Stuart M. Wheater, ‘A Language for Specifying the Composition of Reliable Distributed Applications’, University of Newcastle upon Tyne
http://arjuna.ncl.ac.uk/group/papers/p070.pdf

W2SRC J. J. Halliday, S. K. Shrivastava and S. M. Wheater, ‘Implementing Support for Work Activity Coordination within a Distributed Workflow System’, University of Newcastle upon Tyne / EDOC'99,
http://arjuna.ncl.ac.uk/group/papers/p089.pdf

WAN99 ‘Advances in the dataflow computational model’ (1999)
Walid A. Najjar, Department of Computer Science, Colorado State University, FT Collins, CO 80623, USA
Edward A. Lee, Department of Electrical Engineering and Computer Science, University of California, Berkeley, USA
Guang R. Gao, Department of Electrical Engineering, University of Delaware, USA

WFCM-G ‘Workflow Management Coalition Terminology & Glossary’ (Feb 1999), The Workflow Management Coalition


WFNATO C. Mohan, ‘Recent Trends In Workflow Management Products, Standards and Research’, IBM Almaden Research Centre
http://www.almaden.ibm.com/cs/exotica/wfnato97.ps

WF P Wil van der Aalst, Bartek Kiepuszewski, Arthur ter Hofstede and Alistair Barros, ‘Workflow Patterns’, Minicom / Centre of Cooperative Information Systems (CIS) at Queensland University of Technology
http://www.mincom.com/mtrspirt/workflow/

http://www.aiim.org/wfmc/standards/docs/tc003v11.pdf

WREF2 ‘Workflow Reference Model’, The Workflow Management Coalition
http://www.aiim.org/wfmc/standards/model2.htm

WREF3 The Workflow Management Coalition
http://www.aiim.org/wfmc/mainframe.htm

Oxford University Press

WSDL1 ‘Web Services Description Language (WSDL) 1.0’ (Sept 2000), Microsoft
http://msdn.microsoft.com/xml/general/wsd1.asp

WSFL01 Prof. Dr. Frank Leymann, ‘Web Services Flow Language’ (May 2001)
Member IBM Academy of Technology, IBM Software Group

WSRC
http://arjuna.ncl.ac.uk/group/papers/p076.pdf

WXG
Weixiong Xu, ‘Neowork: A Reliable, Corba-Based Workflow Enactment System For METEOR2’ (1994), University of Georgia,

XJ
Brett McLaughlin, ‘Objects, objects everywhere’ (July 2000), IBM
Table Of Figures

X2.1 Table Of Figures

Figure 1: The corporate travel program directly collaborates with other relevant Web Services, propagating data more effectively [AR001, figure 3]...............11

Figure 2: [AR001, figure 4] describes the role of the three standards. The XML Repository and the Application Server is one unit Web Service, and the Business Registry is another Web Service. The lines here do not describe communication but pointers to information held by the descendent of the line. .................................................................17

Figure 3: [HP00H-3, slide 13] gives the tiered architecture of E-Speak.........................18

Figure 4: The architecture for Microsoft’s .NET platform [MK00M, figure2].................19

Figure 5: The Sun ONE architecture [ONE01, figure 1].............................................20

Figure 6: Correlating the typical scenario to the project. ‘Modeling and definition tools’ is the Specification Tool. The ‘Process Definition’ is the workflow schema description. The ‘Workflow Enactment Service’ is the Execution Environment. The ‘Applications & IT Tools’ is the Web Services. There are no humans in the workflow with regard to Web Service workflow. [WREF, figure 1] .................................................................21

Figure 7: The Reference Model, as seen here [WREF2]..............................................22

Figure 8: Using a common Workflow Management System, it is possible to co-ordinate all the different forms of enterprise information and data flow [IBMWFA, figure 4].............25

Figure 9: Workflow is co-ordinated over clients. This shows how tasks 1 to 9 are completed in a flow that includes interaction with staff and software. [KT96B, figure 1] 25

Figure 10: Task t1 is a compound task containing tasks t2 and t3. When all the inputs for an input set are complete, the task will process these and send output from an appropriate output set to tasks dependent on it. [WSRC, figure4]...............................................27

Figure 11: This diagram shows some simple tasks, which can be transactional (ACID supported) and non-transactional (ACID not supported). [WXG, figure 3.1].............28

Figure 12: A Source and Performer, here implemented by respective agents, can interoperate with several other Sources and Performers. [PPCI, figure 7]..................29

Figure 13: WSFL describes a graph of control between nodes [WSFL01, section 3.1.1.9]29

Figure 14: Note that WfActivity can have a WfProcess that can be a different sub-workflow [MTS, figure 3].................................................................................................30

Figure 15: Biztalk Orchestration splits the view into two parts, the left representing the abstract workflow, and the right representing how it maps to the actual real components [BIZ, figure 2].................................................................................................31

Figure 16: [IBM001, figure 1] shows how the three different types of Web Service interoperate..................................................................................................................33
Figure 17: The two parts of the system, the User Application and WorkflowEngine Web Service, and how they interact. This diagram gives a general overview; the specific more complex versions add other details that are explained in their respective chapter sections.

Figure 18: System Map without frames

Figure 19: Frame numbers correlate to Chapter Sections numbers

Figure 20: The language and related Beans

Figure 21: The AbstractExpression and its related field Beans. The Beans implement Drawable to allow them to be decorated. This facility is used in the User Application and described there.

Figure 22: The SafeFlowService and related Non-Terminal Beans

Figure 23: A, B, and C are parameters passed to the SafeFlow Service, which returns X and Y. In this scenario, the SafeFlow Service has a Sequence as its top-level node, so can itself be assumed to be a Sequence.

Figure 24: Within the expression, data is routed appropriately

Figure 25: This Expression shows how data can be statically defined and filtered within an Expression as appropriate.

Figure 26: This sample XML schema shows a SafeFlow Service with a Sequence expression that has two Static Web Services. The second highlighted import parameter refers to the parameter imported from its outer expression, using the reserved '#import', and the name of the parameter, 'sourcedata'. The first highlighted import parameter follows a similar convention except 'data' refers to the name of the parameter passed in at runtime. Export parameters use a different referencing convention. The export parameter for 'someproxy' refers to the parameter called 'return' returned from the Web Service. This return value is accessed using the reserved '#details'. The import parameter for 'someproxy2' refers to the export parameter from 'someproxy'. The export parameters for the Sequence and Workflow Service refer to export parameters from their constituent expressions, as is the convention with export parameters. The export parameters for the SafeFlow Service refer to what is returned at the end.

Figure 27: The visual version from the User Application has a direct one-to-one mapping. The left hand small circles are ImportParameters, while those on the right are ExportParameters.

Figure 28: A graph-based workflow that cannot immediately be described in SafeFlow

Figure 29: The CircusFlow Bean

Figure 30: A CircusFlow schema, with arrows indicating data flow dependencies

Figure 31: WebServices 1 and 3 are enacting

Figure 32: WebService 4 starts enacting immediately after 3 finishes

Figure 33: WebService 2 starts enacting when ready

Figure 34: The Sync synchronises the data

Figure 35: WebService 5 enacts

Figure 36: Above: CircusFlow
Figure 37: Above: OTSArjuna. Because of the graph-based approach, the difference between CircusFlow and existing workflow languages is not as significant as the difference to SafeFlow .................................................................57

Figure 38: The Terminal Beans ..................................................................................58

Figure 39: Left: The Bayesian Bean; Right: The naïve Bayesian network.....................60

Figure 40: SchemaConverter acts a wrapper to SchemaBean2XML and SchemaXML2Bean, as well as holding common functions required by both. ..............................................62

Figure 41: The simple mapping used in Data Binding. Notice that the Address field it itself a Bean; during XML->Bean, an instance of the Address Bean must be created before SampleBean can be set its Address field. .................................................................62

Figure 42: The WorkflowEngine Web Service can deal with several workflows.............66

Figure 43: The WorkflowEngine Bean ..........................................................................66

Figure 44: How the modified Visitor pattern works .......................................................70

Figure 45: Left: the WorkflowService’s enact method creates a new instance of an Interpreter, and passes the Interpreter instance to WorkflowService’s own accept method. The accept method calls the Interpreter’s visit function passing the ‘this’ – the WorkflowService bean. Right: The Interpreter’s visit function takes the passed WorkflowServiceBean as the AbstractExpression associated with it, and calls its own start function. The Interpreter forks as a new thread and the original execution path returns to the WorkflowServiceBean’s accept method, and then the enact method, where it sleeps until the new thread finishes. ..................................................................................70

Figure 46: Continuing from the previous figure, except generalising to an AbstractExpression – any Expression Bean. If the Expression Bean has several constituent expressions, a new Interpreter instance is created for each .........................................................71

Figure 47: Left: The Child Expression has its accept function called, which is passed the new child Interpreter instance. In a normal Visitor Pattern implementation, the parent Interpreter would have passed ‘this’ – itself. Right: This child Expression calls the child Interpreter’s visit function passing itself to it; the child Interpreter’s visit function takes the child Expression as its associated Expression and calls start(), forking execution The original execution path returns to the child Expression’s accept and then the parent Interpreter’s interpret function. In the case of a Sequence interpret function, the parent Interpreter will sleep until the child Interpret finishes, and then start the visitation process for the next child expression. In the case of a Concurrent, the visitation process next child expression begins immediately. In the case of a CircusFlow, if any of the child expressions has sufficient data to start, the visitation process starts for it, and the child expressions are rechecked each time a child Interpreter finishes and produces more data. ...............................................................................................................71

Figure 48: Left: the Interpreter publishes to its children by calling the method they ‘listen’ on. Right: A child publishes to its parent by calling the method it ‘listens’ on. Note: An Interpreter can be both a child and a parent to other Interpreters – two lists are maintained to publish to. .........................................................................................72

Figure 49: The Interpreter and descendents ..................................................................73

Figure 50: In this example, the main delimiters and delimited content are as follows, and given in the format content(delimiter)content: variable information#parent posterior probabilities, variablename | input range | conditional probabilities. ........................................74

Figure 51: Left: properties window, right: canvas, bottom: monitor window .................77
Figure 52: The hierarchy of visual components in the User Application. The reader need only observe the leftmost / highest level components to obtain an understanding of the User Application’s visuals. It is worth noting that this is the tree that the Java Graphics Visitor will traverse during painting. ........................................................................................ 80

Figure 53: GUI, the main class in the User Application, delegates all functionality to the AbstractManagers ........................................................................................................ 81

Figure 54: The menu ........................................................................................................ 81

Figure 55: How the different toolbar selections affect the state of the CanvasPanel; the Drop State is used in combination with the Flyweight Pattern. This configuration of design in user interfaces is well known. ...................................................................................... 84

Figure 56: In terms of the State Pattern, Canvas Panel is the context, with CanvasPanelState the abstract state .................................................................................................................. 85

Figure 57: Summary class diagram of classes involved in visual manipulation on the canvas. Most of the methods’ functionality in VisualExpr are delegated to one of the classes it uses.............................................................................................................................. 87

Figure 58: The VisualExpr and AbstractExpression (which implements Drawable) keep references to each other. This enables the VisualExpr to decorate the AbstractExpression, as is typical of the Decorator pattern (Left). Additionally, if the AbstractExpression it is decorating has child Expressions which VisualExpr needs the (child) VisualExpr of, the VisualExpr can obtain the child VisualExpr from the child AbstractExpression (Right). ......................................................................................... 89

Figure 59: Each link case and its visual form. Each EParameter resides in the AbstractExpression of the shape it is on the edge of. Hence the EParameter that is furthest left has Untitled 4 as parent, and for the next two EParameters to its right have Untitled 4 as grand parent............................................................................................ 91

Figure 60: View with EParameters (small circles) sticking to AbstractExpression edges. 91

Figure 61: The Passed Parameters display of the Monitor Browser ................................. 94

Figure 62: The Return Value output display of the Monitor Browser ................................. 94

Figure 63: The MonitorManager uses a new thread, Updater, to update and liaise with the WorkflowEngine, so that the User Interface does not freeze while this process occurs. 95

Figure 64: A graph-based workflow that cannot immediately be described in SafeFlow... 97

Figure 65: The VisualExpr and AbstractExpression (which implements Drawable) keep references to each other. This enables the VisualExpr to decorate the AbstractExpression, as is typical of the Decorator pattern (Left). Additionally, if the AbstractExpression it is decorating has child Expressions which VisualExpr needs the (child) VisualExpr of, the VisualExpr can obtain the child VisualExpr from the child AbstractExpression (Right). ......................................................................................... 99
X3 Bayesian Testing

X3.1 Testing

The Bayesian function that is used, as described in section 6.5, was separated out and unit tested separately against the models given below and tests.

Random Testing – Model 1 [DFG.TUT1Q3]

<table>
<thead>
<tr>
<th>S Value</th>
<th>D Value</th>
<th>Correct Parent Distribution</th>
<th>Bayesian Function Result Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0.1089</td>
<td>0.0196</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.1089</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0.1600</td>
<td>0.1122</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Indicator for which variable (input) this information is for – variable S

Ranges of each variable S’s states, for quantisation. MIN, MAX are + and – infinity.

Conditional Probabilities of variable S states (input states) against parent states (method states)

Variable D information

Posterior probability of parent states (method states)
Comprehensive Testing – Model 2 [RN90, fig 4.13]

\{(C \mid \text{MIN}, 1.5; 1.5, \text{MAX})
0.5, 0, 1;
0.5, 1, 0\}
\{(D \mid \text{MIN}, 1.5; 1.5, \text{MAX})
0.5, 0, 1;
0.5, 1, 0\}
0.33, 0.33, 0.33

Test Values

<table>
<thead>
<tr>
<th>C Value</th>
<th>D Value</th>
<th>Correct Parent Distribution</th>
<th>Bayesian Function Result Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.0825</td>
<td>0.0825</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0.0825</td>
<td>0.0825</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.0825</td>
<td>0.0825</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.0825</td>
<td>0.0825</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Comprehensive Testing – Model 3 [RN90, fig 4.14]

\{(C \mid \text{MIN}, 1.5; 1.5, 2.5; 2.5, \text{MAX})
0.5, 0.5, 0;
0.5, 0.5, 0.5;
0.5, 0, 0.5;
0.5, 0, 0.5\}
0.33, 0.33, 0.33

Test Values

<table>
<thead>
<tr>
<th>C Value</th>
<th>Correct Parent Distribution</th>
<th>Bayesian Function Result Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.165</td>
<td>0.165</td>
</tr>
<tr>
<td></td>
<td>0.165</td>
<td>0.165</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.165</td>
<td>0.165</td>
</tr>
<tr>
<td></td>
<td>0.165</td>
<td>0.165</td>
</tr>
<tr>
<td>3</td>
<td>0.165</td>
<td>0.165</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.165</td>
<td>0.165</td>
</tr>
</tbody>
</table>
Random Testing – Model 4

(C) \( \min, 1.5; 1.5, \max \)
| 0.5, 0, 1, 0.25; 0.5, 1, 0, 0.75 |

(D) \( \min, 1.5; 1.5, \max \)
| 0.5, 0, 1, 0.75; 0.5, 1, 0, 0.25 |

(E) \( \min, 1.5; 1.5, \max \)
| 0.5, 0, 1, 0.25; 0.5, 1, 0, 0.75 |

(F) \( \min, 1.5; 1.5, \max \)
| 0.5, 0, 1, 0.75; 0.5, 1, 0, 0.25 |

*0.2, 0.2, 0.2, 0.4*

Test Values

<table>
<thead>
<tr>
<th>C Value</th>
<th>D Value</th>
<th>E Value</th>
<th>F Value</th>
<th>Correct Parent Distribution</th>
<th>Bayesian Function Result Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.0125 0.0 0.2 0.0140625</td>
<td>0.0125 0.0 0.2 0.0140625</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0.0125 0.0 0.0 0.0140625</td>
<td>0.0125 0.0 0.0 0.0140625</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0.0125 0.0 0.0 0.0140625</td>
<td>0.0125 0.0 0.0 0.0140625</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0.0125 0.0 0.0 0.0046875</td>
<td>0.0125 0.0 0.0 0.004687500000000001</td>
</tr>
</tbody>
</table>

- *a should default to 1*
- *b should default to 1*
- *Dd should default to 1*
- *if error value entered for variable, use default value for variable – the first state.*
X4 User Guide

X4.1 Installation

The following packages, provided with the distribution, are required to be on the class path:

xerces.jar; jdom.jar; soap.jar; parser.jar; dg197.jar

To set up the WorkflowEngine web service, the user should download Apache Tomcat and Apache SOAP from http://www.apache.org/ and install them as per the instructions provided with those packages; the latest distribution of SOAP may be substituted for the jar file described in the block quote above. The WorkflowEngine can be deployed like any other Java Bean, via the SOAP Administration screen, specifying dg197.arctic.WorkflowEngine as the Bean to deploy.

The User Application can be run from the command line, specifying the URL of the WorkflowEngine as a command line parameter; dg197.decibel.GUI is the class to run is.

X4.2 Options

Menu Options

The Menu Options are self-explanatory. The Mode menu switches between workflow creation / editing mode and workflow deployment / editing mode.

Canvas Bar Options

Described from left to right.

LEFT HAND OPTION SET: Select an expression (for resizing/moving); Delete an expression; Create a link between two ImportParameters / ExportParameters

RIGHT HAND OPTION SET – Add to the canvas: A Web Service; ImportParameter; ExportParameter; Sequence; Concurrent; FailSequence; Bayesian; Sync

Note: Sequence, Concurrent, FailSequence are only available in SafeFlow; Sync is only available in CircusFlow
X4.3 Using the User Application – A Walk Through

(1) A new blank SafeFlow canvas is presented on starting

(2) Select the blue Sequence Expression and click on screen to add it

(3) Select the grey WebService Expressions and add them to the Sequence; they are ordered in for execution the sequence they are added in

(4) Add ImportParameters to each Expression as required – they are small red circles that stick to Expression boundaries.

(5) Add ExportParameters – small blue circles.

(6) Using the Select canvas button, select an expression using the mouse. Define its properties using the properties table on the left.
(7) Link up relevant ImportParameters and ExportParameters necessary for data communication.

(8) From the Menu, select Mode>Monitor to open the Monitor Browser. Select 'Deploy Current WF' on the Monitor Browser.