2002

CBSE: an implementation case study

Adrian James Collins
University of Wollongong

A thesis submitted in fulfilment of the requirements for the degree of

Master of Science (Honours)

From

University of Wollongong

By

Adrian James Collins (BcompSci)

School Of Information and Computer Science

2002

STATEMENT OF ORIGINALITY

I hereby declare that I am the sole author of this thesis and it has not been submitted to any other University.

I also declare that the material presented within is my own work, except where duly acknowledged, and that I am not aware of any similar work either prior to this thesis, or currently being pursued.

Adrian James Collins

ABSTRACT

Over the last couple of years, the shift towards component based software engineering (CBSE) methods has become a cost effective way to get an application to implementation stage much earlier.

Adoption of Component Based Development methods acknowledges the use of third party components wherever possible to reduce the cost of software development, shorten the development phase and provide a richer set of processing options for the end user. The use of these tools is particularly relevant in Web based applications, where commercial off the shelf (COTS) products are so prevalent.

However, there are a number of risks associated with the use of component based development methods. This thesis investigates these risks within the context of a software engineering project and attempts to provide a means to minimise and or at least manage the risk potential when using component based development methods

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Leone Dunn, for her support, guidance and encouragment during the period of development of the thesis. I would also like to thank Professor John Norrish and Professor John Fulcher for their time in carrying out reviews, and providing advice which help to make this work a reality. I would also like to thank Dr. Dunn and Professor Norrish for the opportunity to complete the project via the scholarship financed by the Strategic Partnerships with Industry – Research and Training (SPIRT) Scheme. They were also kind enough to extend the scholarship period by three months to allow me to complete the software implementation process.

On a personal note, I would like to dedicate this work to Jimmy, Estelle, Dale and Sarah... I love you.

Table Of Contents

List Of Figures

1 INTRODUCTION

1.1 BACKGROUND

Until 1994, distributed computer processing facilities had only been implemented by large, enterprise level organisations. Since that time, the distributed computing model generally, and the client-server model specifically, has been rapidly accepted and adopted by governments and businesses at all levels. Pressure from the business community has seen investment in architecture and technology related to the World Wide Web (Web) expand to the point where the impact is being compared to the steam engine's effect on the industrial revolution [DRUCKER99]. Even though "dot-com" organisations and businesses in general were handed a serious reality check in the late 1990's, investment in research and development (R & D) of client-server architectures has continued unabated, and steadily matured over the last 8 – 10 year period. Much of that research effort has focused on "middleware", which can be defined as the management and co-ordination of data being passed between the distributed computing facilities (hosts) involved in the client-server session. Research by third party vendors into middleware and the convergence of older transactional management technologies such as IBM's CICS and TUXEDO has added significant weight to the use of third party middleware components within the software development process. Maturity and hence developer acceptance of third party components has led to a software engineering variant which has been labelled component based software engineering (CBSE). CBSE promotes the use of components to provide autonomous service objects that can be integrated into larger, more complex components, leading to complete applications. A component is typically independent of platform or language, and may be a single object or multiple objects operating in synergy to form a complete application.

The growing use of components is fueled by the Object Oriented Analysis and Design (OOAD) principle of re-using code components, and this is having an effect on the way systems are developed. Currently, CBSE methods are predominantly targeted at Web based transactional applications using a mixture of prebuilt components such as Web Servers, third party Browsers etc. and specifically written custom code. The shift towards CBSE and "componentware" [VOAS98] has been promoted as a cost effective way to get an application to implementation stage much

earlier. It has been suggested that CBSE reduces the complexity, code size and maintenance overhead of the completed application, and demonstrates that splitting the components into one or more interface layers reduces the impact of change and later maintenance costs [BOURRET00].

1.2 MOTIVATION FOR RESEARCH

The focus of the research described in this thesis is the investigation and evaluation of CBSE methods to determine the effectiveness, or otherwise, of using such methods in the development of software applications. There is no doubt that Web based technologies have materialised at a rapid pace, and coupled with this is a comparable rise in third party vendor offerings which has necessitated a change in the way software engineers approach application development. Object Oriented software engineering design methodologies definitely promote the use of components and code re-use practices, however, current application development literature does describe some failed CBSE projects, making the adoption strategy less clear. Most research advocates the use of components, but advises caution, and suggests that risk management practices be incorporated into the development lifecycle [GANESAN01].

Motivation for the research has been initiated by the Welding Technology Institute of Australia (WTIA), whose requirements include a Web based data services framework to manage the data being generated by a national "Pipeline project". The term 'framework' will be given a more formal definition later, for now we can refer to it as a collection of modules which faciliate data transfer within the pipeline project domain. The project is being developed by the WTIA and the Australian Pipeline Industry Association (APIA), and is a collaborative effort between industry and academic research facilities. The strategic objective is to provide Australia-wide natural gas facilities using a mix of engineering and computer technologies. The operational objective is to minimise field site installation and maintenance costs. The Framework is one of a number of processing units making up the Pipeline project, which when completed and fully integrated will deliver the following field site functionality:

i) *Post weld quality monitoring*, the goal is to provide software which allows data produced by the field site welding process to be remotely administered and monitored for production quality.

ii) *Video surveillance,* of field site operations in real time.

iii) *Wireless data entry* of bar coded and Personal Data Assistant (PDA) data relating to plant and pipe inventory.

iv) *Global Positioning System (GPS) co-ordinate recording of weld position.* This is extremely important for later maintenance when pipe sections may need to be 'exhumed'.

v) *Web based data services Framework* to provide:

- Distributed data management, ie. backup, archival storage, deployment of data to and from field site welding stations.
- Administration of distributed post-weld quality data.
- A browser interface for query and analysis of process control data.

Development of the Framework (figure 1.1) is the focus of this research, and provides an opportunity to observe both benefits, as well as potential risks when using CBSE methods.

1.3 PIPELINE PROJECT BACKGROUND

The WTIA is developing commercial applications which support current and planned natural gas pipeline projects within Australia and South East Asia. Engineering research has developed an automated welding system, coupled with a real time data acquisition module which improves on the currently used manual welding procedures, by allowing potential defects to be identified during the welding process. The post-weld quality monitoring process augments the use of non destructive testing (NDT) methods by providing more specific information on the location of a fault. Current commercial projects use a post-weld x-ray method for fault determination, which is costly and time consuming. The mission critical requirement for the post-weld quality monitoring process is to ensure that pipe lengths are joined (welded) with zero tolerance for defect. The data acquisition equipment generates a log of data points as the weld is conducted, with tolerance levels set prior to the commencement of the weld. An alarm is triggered if data points fall outside the preset tolerance levels, indicating that an analysis of the weld statistics are required to determine the reason for the fault. The welding process uses Gas Metal Arc Welding (GMAW), coupled with a monitoring system which:

- Monitors the control of weld bead geometry

- Monitors the control of metal transfer
- Assesses welding arc stability
- Monitors the potential for undercut
- Facilitates trending analysis on aggregated weld data.

Data gathered at each remote weld site is used to facilitate quality control of the welding operation. Work on pattern recognition systems[OGUNBIYI95, FERNANDES99] has demonstrated how the potential defects may be identified, data point trends may indicate changes in weld quality. The primary concern of the engineering personnel at each field site will be the quality of a specific weld, and ensuring that the production quota is met. However, 'early warning' of creep defects in calibration of equipment can also determined from aggregated trending data gathered at each site. Each remote field site will use a PC host to store welding, operational and logistical data supplied directly from the welder via a National Instruments Data Acquisition card, as well as via the other forms of input mentioned earlier. Each field site is connected (via wireless Internet connection) to an administration host which oversees the welding process and has access to the post-weld data. Remote data management services are dependent on access to the Internet, and it should be noted that there may be times when no service is available, so data must be stored locally until a service becomes available.

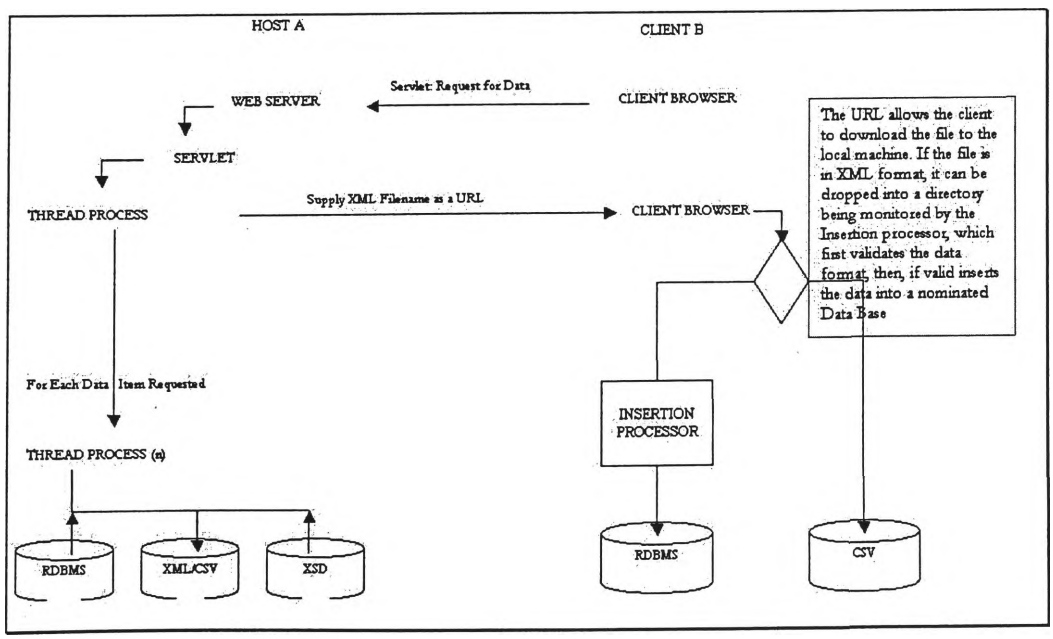

Figure 1.1 Architectural view of the Framework

1.3.1 FRAMEWORK BACKGROUND

A framework can be defined as a domain specific collection of reusable components [CAPRETZ01]. The emphasis here is on 'domain', as a framework should be most useful or most relevant to a specific domain or sphere of reference, with components being used and re-used within that domain. Components making up a framework can be sourced from third parties, or developed in house, the origin of the component is irrelevant. The main issue is whether the components within the application domain are generic enough to facilitate re-use. We cover the specifics of following CBSE principles and methods in the Literature Review, but in order to determine if there are indeed practical advantages in using CBSE methods we will develop the Framework using a combination of common off the shelf (COTS) components and specifically developed code. The development method will include the construction of a protoype to allow a much more detailed, bottom up analysis of the functional requirements, facilitating the identification of potential components for re-use.

1.4 RESEARCH ISSUES

In terms of CBSE principles, the Framework will be component based by definition [SOO01], because the majority of the processing is expected to be achieved using Web, or more generally, Internet technology components. While the detailed design has not yet been completed, major components of the Framework will definitely be sourced from COTS components. The Human Computer Interface (HCI) will be browser based, with the extraction process launched via a Web Server passing the request to a third party Servlet engine. The physical data extraction process will also be achieved using Open Data Base Connectivty (ODBC) components.

There are a number of well documented risks [BRERETON00, WANG01, KOTONYA01] and associated quality of service (QoS) [DSONLINE01] issues associated with using a component based approach. The thesis attempts to use the construction of the Framework to highlight potential risks and identify areas of vulnerability when using CBSE methods. Another major research priority is to investigate the success, and risks associated with integrating a new technology such as XML across a broad component spectrum. Once the risks can be quantified, it is hoped that suitable analysis and design techniques can be presented which minimise

risk and facilitate successful implementation of software developed using CBSE methodologies and related Object Oriented design principles.

1.5 SUMMARY

The aim of this chapter has been to provide a context for the research, and develop the readers understanding of the requirements of the data transport Framework and underlying Pipeline project.

Chapter 2 presents the literature review, which documents risks associated with CBSE, as well as a number of strategic business concerns, such as security, system efficiency and the use of components sourced from third party vendors. The chapter examines the emergence of Web technologies in considerable technical detail, documenting the investigation and research issues carried out as a prelude to the architectural design of the Framework software. The chapter includes documentation relating to the investigation and review process, and can be considered part of the analysis of the problem domain in relation to the Framework requirements. It is assumed the audience has an appropriate technical understanding of related client/server technologies and middleware concepts. However, in order to provide a background for the discussion on XML, Appendix D contains a detailed explanation of the XML language, schema concepts and associated technology uses.

Chapter 3 provides a discussion on the Framework architecture and focuses on the design and build issues. The chapter also contains a synopsis of the benefits to the CBSE method when a prototype is used in the development cycle.

Chapter 4 documents the results of the implementation, and discusses both usage and research issues which relate the Framework and hence the Pipeline project.

In additon to a detailed explanation of related XML technology, appendices are also provided which contain a User and Installation Manual for the Framework, a list of definitions, source code relating to the application, and results of related research.

2 LITERATURE REVIEW

2.1 INTRODUCTION

The activities of speaking and writing are elevated above those of making and doing [SAYER92]. This statement is very poignant, and has far reaching implications for the current developer, none more so than attempting to document the thought processes which led to a particular design or architectural innovation. As developers, we see a problem and attempt to come up with a practical, physical solution, becoming totally focused on the 'making and doing'. We do not naturally keep account of the inspirational flashes which come out of researching the alternatives of how we might solve the problem, only documenting the solution itself. We are trained to document for the user, or the other members of the development team who need to understand and interface with the components which we are creating. We strive to build systems that are useful and work [BOOCH98], but once the system is working, the design issues which were so important in the construction phase, become suddenly irrelevant. Once the system is implemented and stable, change becomes a maintenance issue, and often, the maintenance team are different personnel from the development team. In fact, twelve months after implementation, it is often hard to find a data model which exactly represents the current production database. These attitudes are a reflection of the stable state of software engineering technologies and methodologies prior to the World Wide Web. But post Web, this type of attitude can no longer be supported. Software project teams are constantly dealing with change in all stages of the project, and from all areas, change management is no longer only triggered by requirements creep. These days a project is faced with creep from many areas; technology, architecture, globalisation, and most importantly, 'data creep' (the need to make data more available, to more users, by more interfaces). Fortunately, change management is now recognised as a factor to be managed, like time and cost, and new ideas are being put forward to help project teams incorporate change into the design model. One of these ideas is component based software engineering (CBSE), which recognises that software applications will always be in a state of evolutionary change. Having recognised this, there is a growing trend to develop software as a set of autonomous but co-operating service oriented components which can be used as building blocks in the construction of software applications. The most obvious reason for the adoption of

this layered approach is the rapid rate of technology change, allowing us to 'plug in' new technology modules as they become available. Computer and communication technology advances affect the lifespan of all software applications, but none more so than Web and Interent based software applications [WHYTE01]. Adoption of Object Oriented modeling tools, coupled with a desire to reduce software costs by re-using software and integrating cheaper third party components into the application means that developers are spending more time speaking and writing about the components they are developing. While the Internet has caused an upheaval in the IT industry, it has also provided the industry with enormous benefits by making information and 'knowledge' much more available to the user [WILLIAMS00]. This increasing access to information, coupled with business related profitabilily issues, is driving an upward spiral in the number and scope of Internet based distributed applications. This upward spiral is in turn fostering a great deal of research into technologies and methods used to build distributed applications, particularly eCommerce applications. The application developed for the Pipeline project uses a number of technologies and development methods which reflect the changing attitudes to software development.

2.2 RESEARCH

2.2.1 APPLICATION OBJECTIVES

Motivation for the research came from a desire to develop a Framework for the Pipeline project which was easy to use, scalable and took advantage of off the shelf components (COTS) wherever possible. In fact, as the research progressed, it became clear that one of the possible future goals could be to replace all custom code components with off the shelf components. Initially, the intention was to manage the entire transport process under program control, however, analysis highlighted the following potential problems with this approach:

- If the data model changed, the extraction/insertion modules on each host would need to be modified to reflect the new table layout.
- If the tables did not exist on the receiving host, a create script would need to be sourced and executed prior to data being transported to the remote host.
- The send and receive modules required manual starting, so if the machine was unreachable or required rebooting, there was potential for the session to

require some administration for reconnection. This is a genuine concern and has enormous implications, since the data gathering Hosts are installed on vehicles in the field, in what can only be considered primitive conditions. It is expected that these Hosts will have access to the Internet via wireless technology, with availability subject to atmospheric and climatic conditions, as well as geographical location.

- Security was not a genuine concern, given the application, however, data corruption during the transport process was a potential risk [GREENSTEIN01]. Data integrity is considered a mandatory requirement, and standard client/server 'socket' architecture provides no real means of ensuring the data would be received in exactly the same form as it was sent. Refer to appendix D.2 for a detailed explanation of the Socket abstraction.

- One of the project objectives was to provide an extraction method which ported the data to a distributed Host in Comma Separated Variable format. This facility allows the data to be used for other analytical purposes or in third party applications such as Excel, Lotus Approach etc.

The benefit of using a classic client/server based 'socket' connection scenario is reasonable data throughput, but the overhead and tedious necessity of connecting the client to the server under program control is a negative from an operational perspective, so finding a suitable 'middleware' to automate this process is an imperative. Fortunately, use of third party Web Server technology is now widely accepted within the IT industry as a means of managing the client/server session, so it seemed a natural extension to focus on the use of this architecture as a transport medium. A number of component based middleware technologies [CAMPBELL99, ORFALI96] were investigated to determine the benefits and feasibility of use for the Framework. The main objectives were ease of use for operation, maintenance, deployment, and of course an efficient transport process. There are a number of middleware technologies which have potential uses within the Framework application domain. Of course, COM/ActiveX was a mandatory selection for querying the database, since Microsoft was the Operating System specified by the stakeholders, but the Framework data transport process required that a number of other middleware options be investigated, these were:

- Java (Peer/Peer) using Remote Method Invocation

- Java (Client/Server) using Enterprise Java Beans
- Java (Client/Server) using Servlets
- Common Object Request Broker (CORBA)

The key requirements for use of these technologies are openness, ease of configuration and interoperability. This can include mixing and integrating components which can be sourced from various areas, and using connection specific code to integrate components purchased from third party vendors (COTS), who may specialise in the field and have developed a 'best of breed' technology to manage a specific processing area.

2.2.2 RESEARCH OBJECTIVES

From a research perspective, the objectives are as follows:

- Determine the effectiveness of using component based software engineering (CBSE) methods in the development of software applications. In addition, to gauge the impact of CBSE on post implementation maintenance to determine whether these methods make change easier and less costly, or otherwise.
- Use the construction of the transport Framework as a case study to highlight potential risks and identify areas of vulnerability when using CBSE methods. The major software development risks associated with CBSE have been identified and presented in the following pages. It is hoped that the outcome of the research will determine if previously published doubts about overall benefits and associated risks are still valid, given the maturity of Web based technology, or whether the issues are losing relevance in the face of that maturity.
- Investigate both the benefits as well the risks associated with integrating a new middleware technology such as XML across the range of COTS Web components. Again, it is hoped the research can determine if new risks are emerging, which are a consequence of newer technology or resulting from complacency over the acceptance of the more 'mature' methods of development.
- Review the Object Oriented principles used in the development to determine if this added value to CBSE method, particularly in the areas of interface, re-use of code and ease of assembly and integration of the components. The project

makes also heavy use of a prototype to facilitate 'bottom up' analysis and identify re-use candidates.

The following sections describe the concept of component based software engineering, associated risks and related technology issues.

2.2.3 Component Based Development

One of the most significant recent developments in software engineering is Component Based Software Engineering, which promotes the use and re-use of self contained service objects. Using this development paradigm, applications are 'assembled' by gluing together components that may be supplied from existing source libraries or in binary form from third party vendors. This type of development has been strongly influenced by Sun's JavaBeans and by Microsoft's Component Object Model (COM) and ActiveX technologies. Any software which is developed, acquired, or deployed where the primary design goal is reuse, is considered to be 'component based', and can be:

- Commercial off the shelf (COTS)
- Public domain
- Freeware/Shareware
- An in-house developed service-based component
- A mixture of both COTS and in house eg. Enterprise Java Bean's

The basis of Component Based Software Engineering (CBSE) is that components provide services that can be integrated into larger, more complex components, leading to complete applications. A component is typically independent of platform or language, and may be a single object or multiple objects operating in synergy to form a complete application. The growing use of components is further fueled by Object Oriented Design principles and is changing the way systems are developed.

2.2.4 CBSE Risk Factors

While there are considerable benefits associated with the use of component based development, there are also a number of significant risks that have been well documented [GANESAN01]:

- The blackbox nature of the software when using COTS components

- Lack of software quality information
- Hidden Costs associated with post implementation maintenance
- Lack of a suitable 'bottom up' design information
- Lack of accompanying documentation/information regarding the software
- Potential for a longer and more costly development lifecycle

The blackbox nature of COTS software, ie. usage is restricted to the Application Program Interface (API). In order to maximise the benefits, developers would like to be able to review the design criteria of the component [KOTONYA01]. This is particularly relevant for components being integrated into a distributed or Web based application with associated issues such as security, scalability, performance etc. [BRERETON00].

Lack of software quality information and the longer term issues for third party components relating to product direction [KOTONYA01]. Clearly the objective for a vendor is to develop a solution which has profit potential, and this objective may cause future support and functional issues for a project making current use of the vendors technology.

Hidden Costs of post implementation maintenance [WANG01]. The two issues mentioned above may mean that when application requirements change due to user or market forces, the developer may not have the necessary component design information to determine the effect of the change on the component. Having to guess the effect of the change, or build a test bed to determine the effects may increase the cost of maintenance and so reduce the overall benefits of using pre-built components.

Lack of a suitable bottom up design strategy [CAPRETZ01]. This is a significant issue for the current, results-centric project team when a component based design method is being used and the project is large and or complex. Use of a 'bottom up' strategy could add significant time to the design phase, thus increasing the cost of development and delaying the implementation date. Ideally, during each design and

build iteration, as more components are identified along the design path, re-evaluation of the complete set of existing components would be necessary. Candidate components can be identified more quickly if a strategy is in place which models at an atomic, cohesive level. The designer is then able to map new functional requirements against the repository of granular pre-existing components.

Lack of documentation/information regarding the software. This may relate to configuration, operation, API specifications for 'wrapper' development, or error handling etc. Lack of current or accurate documentation is a problem in any software engineering project, but is particularly relevant in a CBSE environment, where third party components are being employed.

Risk of allowing a longer development period to reduce or nullify the expected cost savings from using off the shelf and re-usable components. Even when the project stakeholders proactively accept the potential benefits of using component based methods, there is still a concern over the time taken to produce tangible results. Analysis and Design can (and should) take up to 60% of the overall project timeframe, more when COTS software is being integrated with custom code. When component based software engineering methods are being used, domain analysis alone may take up to 25% of the project [CAPRETZ01]. Domain analysis is a study of the entire problem domain, which includes both functional and non functional issues and requirements and may be completed a number of times during the project. Domain analysis specifically is used to review the build phases and identify code components which may be candidates for re-use.

Risks associated with CBSE
The blackbox nature of the software when using COTS components
Lack of software quality information
Hidden Costs associated with post implementation maintenance
Lack of a suitable 'bottom up' design information
Lack of accompanying documentation/information regarding the software

Potential for a longer and more costly development lifecycle

Figure 2.1 Risk Summary

2.2.5 COLLATERAL RISKS

There are also a number of less obvious risks which can be equally costly to a software engineering project, unless action is taken to minimise or at least manage the risk. When components are derived from a number of sources, there is often a difference in the vision shared by each component stakeholder. The independent, third party component vendor may have an entirely different development and functional improvement focus from the user base. The developer must never lose site of the fact that components are third party derived, and software maintenance strategies may need to be completely redeveloped to accommodate potential future problems [BRERETON00]. This is especially true in distributed applications where maintenance becomes much more difficult and complex, because component source code is either partially or completely invisible. There is also the issue of 'frozen functionality' [VOAS98] in which the vendor has disappeared by the time maintenance becomes an issue. Incompatible upgrades (added features or bug fixes that, while *independently* reliable, are incompatible with the host system) are also an issue. Potentially, third party components may also contain viruses, trojan horses etc. and must be thoroughly tested prior to release.

A more emotional issue for the developer comes in the form of loss of creative scope. Developers of the future will be concerned with the integration of software components, potentially only writing code to provide 'black box' wrappers for the COTS components. IBM and Gartner are two organisations who foresee a move to pattern based development, where a finite set of appropriate approaches (Patterns) for representation, storage, and retrieval of reusable components is emerging [FLURRY01]. This trend is now even extending into the early project areas of requirements gathering and analysis. While this issue cannot be considered a risk in the tangible sense, when the design scope is constrained to fit into a predetermined pattern, the design may become too 'generalised'. The designer is then pressured into 'fitting' the requirement solution into the design or architecture pattern.

Mismatch can also occur when components fail to meet the architectural constraints. For example, components which operate under Windows NT may fail when used under Windows 2000. Functional deficiencies arise when components do not satisfy

all the functional requirements. Quality and maintenance issues must be addressed when developers integrate several disparate, third party components into a common application. Component adaptation may involve the use of special purpose wrappers to achieve interoperability, increasing subsequent maintenance costs for integrated systems [KOTONYA01] because of a lack of interoperability standards provided by the component vendor. Developers must readapt or re-glue components as they evolve, or potentially restrict the component types used for a particular integrated system. Developers must address how the integrated system inherits properties associated with product parts, and when integrating a combination of high quality and low/medium quality components, with the resultant system being assessed to the lowest common denominator.

As well as being a component integrator, the 'new age' developer must be skilled in risk analysis and be able to determine current and future component cost/benefits, for example, use of components may increase the cost of later maintenance. Implementing distributed components can become a human resource issue, significantly increasing the time taken for distributed deployment, maintenance tasks for testing, evaluation and acceptance. Integrators must develop suitable component testing practices, forcing potential vendors to declare and provide documentation on issues like dependencies and constraints. Operational and interoperability issues covering platform, architecture, Web Server, Application Server, Middleware etc. cannot be genuinely proven until the components are deployed in an environment in which they will run. Responsibility ultimately rests with the developer/integrator, who must play a significant role in risk assessment, and assessing the benefits versus drawbacks of using certificated components. As with any off the shelf package or component, there will be a trade off between implied quality through certification and the need to get the product to market.

These are genuine issues which must be addressed when developing applications using a component based methodology. Some of the claims relating to development risk were made when Internet technology and middleware tools were less mature. Component based development is now more wide spread and has become an accepted means of reducing the application code base. However, doubts over the net gain in productivity and benefit are still being raised in literature relating to CBSE. In order to determine and select an appropriate architecture for the Framework, a review of current technologies must first carried out.

2.2.6 RMI (REMOTE METHOD INVOCATION)

Java technology was a prime candidate for use in the Framework, and Remote Method Invocation (RMI) specifically, presented with a lot of advantages. It is Java Centric, so well accepted and used within Software Engineering applications. The prototype, which is documented in the following architecture chapter was constructed using Java, so potentially some of the code could be retained, and the RMI implementation of JDK 1.2 had most of the Common Object Request Broker Architecture (CORBA) recommendations in place, particularly RMI over Internet Inter Orb Protocol (IIOP is a super set of TCP) which would:

- Allow the file or sql selection to be processed on the target host as a Transaction (ie. fully committed or fully rolled back).
- Provide support for persistent Object references, so once the host is located, services can be requested on an ad-hoc basis without the need to continually broker the request.
- Provide support for remote Object activation which overcomes the manual restarting of the Objects when the machine is re-started.

Internet Inter Orb Protocol (IIOP) sits on top of TCP/IP and value adds the CORBA message exchanges prior to passing the information to and from the application. In this way, details of service location, and transactional boundaries are abstracted from the user application [ORFALI98]. RMI clients do not interact directly with distributed Objects, but interface via a published interface, as do CORBA clients. Arguments are marshalled via the Java Serialisation service (java.io.ObjectOutputStream/java.io.ObjectInputStream) and passed to the distributed Object via the relevant stream. The distributed Object is bound to the Java Naming Service and is then accessible via the abstracted proxy stub on the client. For example, a class GetWeld, on the local host, can call a local function getWeld(). This method is called via the proxy getWeld(), which uses the Naming.lookup service to locate a remote instance of getWeld(), execute the function, then return the results to the local proxy.

NOTE: Notation in the code examples which follow use standard Java syntax.

```
public interface GetWeld extends Remote
{
    String getWeld() throws RemoteException;
}
```

On the remote Host, a set of RemoteObject classes is sub classed to create the required objects

```
public class GetWeldImpl extends UnicastRemoteObject implements GetWeld
{
    public GetWeldImpl() throws RemoteException {    super(); }
    public String getWeld() {    return  "GMAW 301" ; }
}
```

In the main method of the distributed Object, the object is instantiated, then bound to the RMI Naming Service registry, as per:

```
GetWeldImpl MyObj = new GetWeldImpl ();
Naming.rebind("//host:port/name", MyObj);
```

In addition to abstracting the underlying 'socket' layer, RMI also manages the security aspects of the application, automatically registering the distributed Objects with the security service running on the distributed Host. The range of options is extensive and allows the developer to set these options in a persisted (physically stored) security file which is accessed and implemented by the Java Virtual Machine running on the distributed Host. There are a number of genuine benefits over raw 'sockets', the first was the remote activation feature which would allow the module to be called by the client when necessary, and the other was the ease with which the RMI client could persist (store) the data once it was received. In order to make use of the activation feature, the java.rmi.activation package is included in remote Object. In the previous example; the class declaration would be modified to

```
public class GetWeldImpl extends Activatable implements GetWeld
```

instead of

```
public class GetWeldImpl extends UnicastRemoteObject implements GetWeld
```

Activatable and UnicastRemoteObject are both sub classes of RemoteObject, and the Activatable implementation informs the RMI registry running on the distributed Host to load the GetWeldImpl if not already instantiated. With regard to the persistence facility, the receiving client would then only need to instantiate a class which extended the abstract class of RandomAccessFile in order to persist the rows to an ASCII file, eg. public class ANSIFileStream extends RandomAccessFile.

In analysing the usefulness of RMI as a transport solution, despite the range of options and flexibility associated with RMI, it remains a language specific middleware. Unless designed and implemented using a component based model, or at least a layered approach, there is a real possibility that the potential for later

integration with non java centric components[CAMPBELL99] will not be possible. In addition, at the end of the day, it was still an abstracted 'socket' model, and required a considerable amount of custom code to manage the client/server session

2.2.7 ENTERPRISE JAVA BEANS (EJB)

The distributed nature of EJB is facilitated by the further abstraction of Java's Remote Method Invocation (RMI) methodology. Enterprise Java Beans also provides an abstraction for Component Transaction Monitors (CTM). Component Transaction Monitors represent the convergence of two technologies; traditional transaction processing monitors such as IBMS's CICS and TUXEDO; and distributed object services such as CORBA (Common Object Request Broker Architecture) and Java RMI [HAEFEL00]. The CTM is implemented as an Application Server (resident on the same host as the data source), and is responsible for monitoring the state and behaviour of the client/server session under it's control. The major benefit of using this technology is that complex issues such as data security, concurrency, persistence and transactional integrity are all abstracted by the CTM. This frees the developer to focus on the business issues rather than becoming bogged down in the technical detail of transaction management. Enterprise Java Bean technology is extremely relevant to distributed processing, and the Framework. A considerable amount of time was spent setting up a test environment to investigate the internals of the technology and evaluate it's usefulness. The issues of transaction management and data concurrency management need to be addressed from the Framework perspective when data is being inserted back into the requesting clients database.

After a considerable amount of research effort and time was spent in testing and evaluating the technology, it was decided that the benefits did not justify the cost in terms of outlay, product support or specialised knowledge for later maintenance. Research of other technologies identified far more efficient and cost effective solutions for managing the processing requirements. For instance, while it is necessary that the loading of an extracted file back into the clients local database needs to be completed as a transaction, that process is managed by the Object Data Base Connection driver (ODBC) natively, and ODBC is a free technology component. For this reason alone, the cost of a third party CTM Application Server was not justifiable, in addition, the expected data volumes produced by the field welding sites were too high to be efficiently managed by a CTM. In summary, the

CTM overheads and administration requirements inherent with EJB technology were not justifiable, given the licensing cost of the EJB application servers and complexity of implementation, configuration and management of the environment.

2.2.8 XML - (EXTENSIBLE MARKUP LANGUAGE)

With the acceptance and adoption of new Web based technologies, the Browser is only one of the diverse Human Computer Interface (HCI) alternatives being presented for access to a data repository. Telephone, Wireless Application Protocol (WAP), Personal Data Assistant (PDA) and Smart White goods are all areas which provide a user interface and need an underlying set of standards to structure and format data content for the interface. XML development was initiated by the XML Working Group, under the auspices of the World Wide Web Consortium (W3C) in 1996, as a solution for supplying data to these interfaces in a standard manner. Development of the language was also prompted by pressure from developers who had two valid issues with using Hyper Text Markup Language (HTML) as the data interface. Development of a robust data interface is limited by HTML and it's lack of structure and adherence to standards. Also, developers refused to accept Standardised General Markup Language (SGML), which was initially proposed by the W3C as a replacement for HTML, because of its complexity and verbose specification.

It is well documented that HTML is limited to a fixed set of markup tags, while XML allows developers to create their own tags, or use tags created by others, ie. XML facilitates reusability and extensibility. As with SGML, XML can be formatted and validated by a Document Type Definition (DTD), which allows the user to declare what constitutes markup with the XML page and also what the markup means within the page. Once the XML parser has validated the document, a document tree is created, based on the hierarchical structure declared in the DTD. This document tree may then be made available to the user, or accessed by processing applications.

There is no doubt that the process of transporting data over the Internet has become a trivial task in the mind of the Internet user. The irony is that the delivery of that task will become proportionally more difficult as data traffic increases and bandwidth usage becomes a more critical issue. XML provides a means to maximise the efficiency of transporting data over the Internet by allowing developers to

develop specific 'tag' naming conventions which minimise the size of the file being transported. Based on research into the use of XML, and market penetration, two clear content patterns are emerging for XML; data-centric and document-centric [BERTINO01]. To manage XML data-centric documents, a data base management system (DBMS) must support both data extraction and data formatting services. Data transfer software is either built into the system or available as third party middleware which supports the extraction process. Provision of these services by DBMS vendors acknowledges that XML is now fully accepted as an external data containment and data transport solution. Most commercial DBMS's have by now been extended to accommodate the packaging of query results into XML format, including Oracle, IBM (DB2) and Microsoft SQL Server 2000 [BERTINO01]. For example, both Sun and Microsoft (ASP.NET) are developing 'Rowset' Application Program Interfaces (API), with implementations that can serialise and persist (store in a physical form) the data, metadata and properties of an SQL query result set to XML. That way, the result set can be disconnected, transported across the network, and manipulated by a remote application [WILLIAMS00]. In addition, middleware technologies are also beginning to make much more use of XML [SOO01], and independent middleware vendors are actively researching the use of XML as a strategy to allow their products to communicate [CAMPBELL99].

There is potential for a considerable amount of Framework data to be shipped during a session, so XML presents with a number of potential benefits to the application. As analysis progresses and the full implications of the potential uses for XML becomes apparent, the design criteria is shifting from incorporating and integrating some off the shelf components with the custom code, to fully realising the goal of replacing all custom code with COTS components [CAPRETZ01]. The most significant issue is that XML provides a connectionless alternative to the typical 'socket' based solution. Once data is extracted from the database and repackaged into XML format, third party component based web utilities can be used to transport the data from host to host, via the Internet. For this reason alone it was decided to construct the Framework architecture using XML as an integral component, however, as analysis progresses, many other prospective benefits are becoming apparent via the use of XML. As well as providing a container to store and forward the extracted Pipeline data, it's use allows the Framework to optionally map the extracted data base data into other formats, such as comma separated variable (CSV), facilitating use by third party statistical processing applications such

as Excel™. This is achieved by using a schema, which provides an explanation of the layout or format of the data residing within a database table, and is available to describe the contents of an XML file to other software applications wishing to make use of the XML data.

Within the context of the Framework, the schema also provides another level of usefulness to XML, mainly because of the shortcomings of the document type definition (DTD), outlined below. While this is not a criticism of the DTD which is a legacy of SGML[WILLIAMS00], and is used to great benefit when the content of the document is to be used as a document. However, when the content of the document is data, and a receiving application has the responsibility of processing that data, moreover, posting it into a relational database, the DTD can be found wanting in a number of areas:

- DTD is not declared in XML syntax, meaning the creator must learn a new language, which is complex, verbose and tedious to write.
- There is no easy or extensible way of programmatically modifying the DTD once the document has been created. The schema (which itself is an XML document), can be manipulated using a Document Object Model (DOM).

Use of the schema means that each extraction request can be mapped to a single database table, making usage, administration and data management a simple matter for both statistical analysts and field engineers [SOO01]. The XML document is quite human readable in its natural form, and we can assume the engineers are extremely familiar with the internal table layout. Each field site produces considerable amounts of data relating to the pipe weld process, use of the schema also facilitates an extremely 'thin' tag wrapping, which considerably reduces the size of the extracted XML file. Once this data is captured, it will be inspected and analysed, either locally or at a central repository, collated and then moved to a long term repository. The schema allows the data to be efficiently packaged (as XML) and transported over the Internet via Browser download.

Benefit
Safe container to store and forward extracted Pipeline data
Facilitates mapping the extracted Pipeline data into other formats
XML document is quite human readable in its natural form

Facilitates an extremely 'thin' tag wrapping, reducing container file size
Facilitates the mapping of container data back into a database repository
Support is built into many third party Web products eg. Internet Explorer

Figure 2.2 XML Benefits Summary

2.2.9 SERVLETS

Another technology which was investigated as a possible candidate in the CBSE environment was Servlets, which are Java code modules that use a request/response paradigm to extend the facilities provided by a Web Server. The Servlet application program interface (API) can best be described as Common Gateway Interface (CGI) replacement technology, which allows arguments embedded in HTML to be captured under program control and passed to a co-operating Java program. It should be understood that Servlet technology is non proprietary and open in the same sense as Java. The Servlet engine which deals with the incoming HTML traffic is specific to the underlying Operating System and Web Server environment. So whilst the implementation and initialisation properties are Vendor specific, usage of the technology assumes nothing about how a servlet is loaded, the server environment in which the servlet runs, or the protocol used to transmit data to and from the user. This allows servlets to be embedded in many different web servers, and makes them an effective substitute for CGI scripts [JAVADOC01].

USES OF THE SERVLET API

- Dynamic Data Processing via HTTP
- Concurrent processing, by handling multiple requests concurrently and instantiating a thread for each request.
- Forwarding requests to other servers and servlets, facilitating load balancing, mirroring and archival processing scenarios.
- Being a community of active agents. A servlet writer could define active agents that share work among each other. Each agent would be a servlet, and the agents could pass data among themselves [JAVADOC01].

2.2.9.1 SERVLET ARCHITECTURE OVERVIEW

The Servlet API is provided free of charge from Sun, and uses the 'specific' HttpServlet interface to support a range of management methods that facilitate communication with a client application. When a servlet accepts a call from a client, it receives two objects, a ServletRequest and a ServletResponse.

SERVLETREQUEST: The users request arguments (if any), are encapsulated in the HTTP communication from the client (typically a Browser) to the server. This allows the servlet access to parameter information passed in (as arguments) by the client. If the method type is POST or PUT, the servletRequest object also provides the servlet with access to the data via an input stream (ServletInputStream) [JAVADOC01].

SERVLETRESPONSE: The servers response to the client are encapsulated in the programmatically generated HTML, from the servlet back to the client. In addition, the servlet provides an output stream (ServletOutputStream), and a Writer Object through which the servlet can send the reply data[JAVADOC01].

2.2.9.2 SERVLET LIFECYCLE

Servers load and run servlets, which then accept requests from clients and return data to them. When a server loads a servlet, it runs an initialisation method. Even though most servlets are run in multi-threaded servers, there are no concurrency issues during servlet initialisation. This is because the server calls the init method once, when it loads the servlet, and will not call it again unless it is reloading the servlet. The server can not reload a servlet until after it has removed the servlet by calling the destroy method. Initialisation is allowed to complete before client requests are handled (that is, before the service method is called) or the servlet is destroyed.

After the server loads and initialises the servlet, the servlet is able to handle client requests, and processes them in its SERVICE method. Each client's request has its call to the service method run in its own servlet thread. The threaded servlet model can run multiple service methods at a time.

Servlets run until they are removed from the service, for example, at the request of a system administrator. These design 'features', make development and testing extremely challenging, because the developer has no genuine way of ensuring that the latest compiled version of the Servlet has explicitly been taken up by the Servlet

engine. When the Web Server ISAPI server removes a servlet, it runs the servlet's destroy method. The method is run once; the server will not run it again until after it reloads and reinitialises the servlet. When the destroy method runs, however, other threads might be running service requests. [JAVADOC01].

2.2.9.3 WRITING THE SERVLET

Servlets implement the javax.servlet.Servlet interface, and effectively allow the developer to receive the arguments passed in via HTML as a set of calls to the servletRequest Object. The API supports two thread models, catering for more specific uses of the technology, by allowing the developer to nominate the type of model to be used as a compile time option. If the servlet is being used as an HTML receiver, then the multi-threaded model would be appropriate. However, if the Servlet is shielded from direct access by using a specific port number, the developer can nominate a single threaded model. For example:

```
public class myServlet extends HttpServlet
                implements SingleThreadModel {    /* typical code */ }
```

2.3 SUMMARY

This chapter has detailed the technology investigation and research carried out in preparation for finalising the architectural design of the transport Framework software. The analysis and investigation process had a number of objectives:

- To select an appropriate technology for development of the Framework.
- To select appropriate third party components which modeled the mix of COTS and custom developed code within a much larger project. This provides a basis for comparison and review of the risk/benefits associated with implementing applications using CBSE methods on a much larger commercial scale.
- Identify functional areas within the Framework requirements with the objective of developing code modules engineered for re-use. In order to maximise the value of the analysis and provide 'bottom up' design input to the Framework architecture, a functional prototype will be developed. The prototype will help to determine the scope of the data extraction and re-insertion requirements, and identify areas for re-use within the application domain. Review and iterative analysis of the Framework domain to identify

components that are candidates for re-use will also model the development method used in much larger CBSE projects.

The architectural design chapter (following) documents the design criteria of the project and looks in detail at the functional issues which needed to be addressed in the final design. In order to provide the reader with a context for the discussion of component based development issues, application design is addressed at class level, with the functionality of each class being discussed in some detail. The following chapter focuses on the areas of:

1. Extraction of the data and creation of the XML document (Server Side)
2. Creation of, and access to, the in-memory Document Object Model
3. Retrieval and insertion of the data into the target database

An understanding of XML and related XML Schema technology is required as a prerequisite for the method chapter which follows. To this end, Appendix D provides background and issues relating to the technologies investigated, particularly XML and it's value as an external data container.

3 ARCHITECTURAL METHOD

'Method' suggests a carefully considered way of approaching the world so that we may understand it better. To make judgements about method it helps considerably if we have some idea of the nature of the relationship between ourselves and that which we seek to understand.

Andrew Sayer 1992

3.1 BACKGROUND

This chapter describes the architecture of the Framework, documents the issues surrounding the design and construction of the application. The chapter also provides a detailed description of the Framework internals which relate to the use of the XML Schema and associated in-memory document object model which drives the Framework. In order to provide the reader with enough information to clearly evaluate the ease, or difficulty of integrating components into the design, it was felt that the various component models should be documented to the class level. This is particularly relevant when covering the sections relating to the polymorphic use of the XML Schema for data extraction, writing the data to file format (persistence) and re-insertion back into another database.

The most basic functional requirement of the Framework was to implement a generic, user driven interface for moving data from one database to another. As long as both sending and receiving hosts have access to the Internet, then data can be exchanged between hosts. It must also be remembered that in addition to achieving the functional objectives of the Framework, research is focused on the perceived risks/benefits of using CBSE methods. As stated in the introduction, one of the objectives of the research is to use the construction of the transport Framework as a case study to determine if previously published doubts about overall benefits and associated risks of using CBSE methods are still valid, given the maturity of Web based technology.

The Re-use of code is one of the major initiatives of the CBSE philosophy, and an integral part of the Object Oriented development lifecycle. In order to identify potential processing areas where re-use of code can be achieved, design must be

completed from the bottom up, that is, the functional processing requirements must be broken down to a granular or atomic level. Once each of the processing steps can be identified in isolation, other processing steps within the application domain with similar needs can be grouped together. To facilitate a bottom up analysis and better understand the issues relating to database extraction and re-insertion requirements, and identify areas for re-use, a prototype was developed. Construction of the prototype also provides useful information in analysing the problems associated with incorporating COTS components into a pre-existing application. Proponents state that following the CBSE methodology allows the designer to maximise system extensibility and minimise later redesign costs as a result of change [CAPRETZ00]. However, other research highlights the risks associated with post implementation maintenance, and functional enhancement of an existing application by including COTS components [VOAS98, BRERETON00]. There is general agreement that use of CBSE methods allow the application under development to be completed more quickly. CBSE literature suggests that iterative reviews of the problem domain during the construction phases improves the likelihood of success (when using CBSE methods). To be able to highlight, then validate or dispute the risk issues in any tangible way, it is important to provide a development scenario that models the risk potential faced within a commercial software engineering project, and this is our goal.

3.2 TERMINOLOGY

There are a number of terms used in the following chapter which may have different interpretations for different readers. This section is included to reduce confusion by providing a definition and explanation of those terms.

CLASS

A class is a representation of an entity which contains both data and a means to manipulate that data. The term 'class' is used within the design scope, and while data variables can be declared within the class, actual values cannot be assigned until memory is allocated to that class.

OBJECT

An object is also is a representation of an entity which contains both data and a means to manipulate that data. The term 'object' is used within the runtime scope,

and since memory has been allocated to the 'class', we refer to the runtime instance of the class as an 'object'.

INSTANTIATED

When a class has memory allocated to it, we refer to this process as 'instantiation'. Once memory has been allocated, we then refer to the class as an 'object', or an 'instance'.

THREAD

A thread is an associated child (thread of execution) of a parent process. The parent process, once instantiated, has memory allocated, and an address space which can be shared by any subordinate threads which have been spawned. Because threads share the parent processes address space, context switching between threads is much more efficient, thus maximising the degree of concurrent execution.

WRAPPER

This is a specific code module which is written to allow two dis-associated code modules to communicate. A wrapper facilitates a connection between two disparate components, and is particularly useful when COTS components need to be incorporated into an application, and communicate with other COTS components or custom code modules within that application.

3.3 PROTOTYPE

OVERVIEW

Functionally, the prototype managed the connection between the server and the requesting client host, and was constructed to transport the contents of a single database 'table' from the Server host and deposit the contents into a database resident on the requesting Client host (Figure 3.1). The architectural design of the prototype is classic client/server, with the Server process listening for incoming client requests, then initiating a thread to manage the interchange between the two databases. The prototype was premised solely on the transmission of data from one remote database, to another, using a dedicated Socket model to transmit the data. To this end, an Application Program Interface (API) is provided at each end of the socket, with an 'extractor' interface receiving configuration and SQL parameters

from the client. Once the SQL request is processed, the resulting data is shipped back to the client, an 'inserter' process accepts the socket stream and generates a series of SQL 'insert' statements, posting the data into the remote database. The process itself is efficient from a transport perspective, given that any method of distributed transport would require a marshalling process to 'flatten' the data into a text stream for transmission over the network [CAMPBELL99].

Priorities at this stage of the analysis were issues relating to data transport, and how to efficiently move data from one place to another. At this stage, a user interface which allowed the client to select data from the database for transport had not been built. The prototypes functional responsibility was to transport data which already existed in a Relational Data Base, via the Internet, to a remote Relational Data Base. Both server and client components within the prototype share a number of common classes, providing small re-use, polymorphic benefits, because the same components could be used for both extraction and insertion. This allows common 'container' properties of objects on both server and client to store in memory representations of the database data being processed. For example the DBConnection class (Figure 3.2 and 3.3) has a role on both the client and server, the 'runSQLSelect' method allows both input and output movement of data. These shared classes are included in both the server and client dialogue to show the differing functionality exercised.

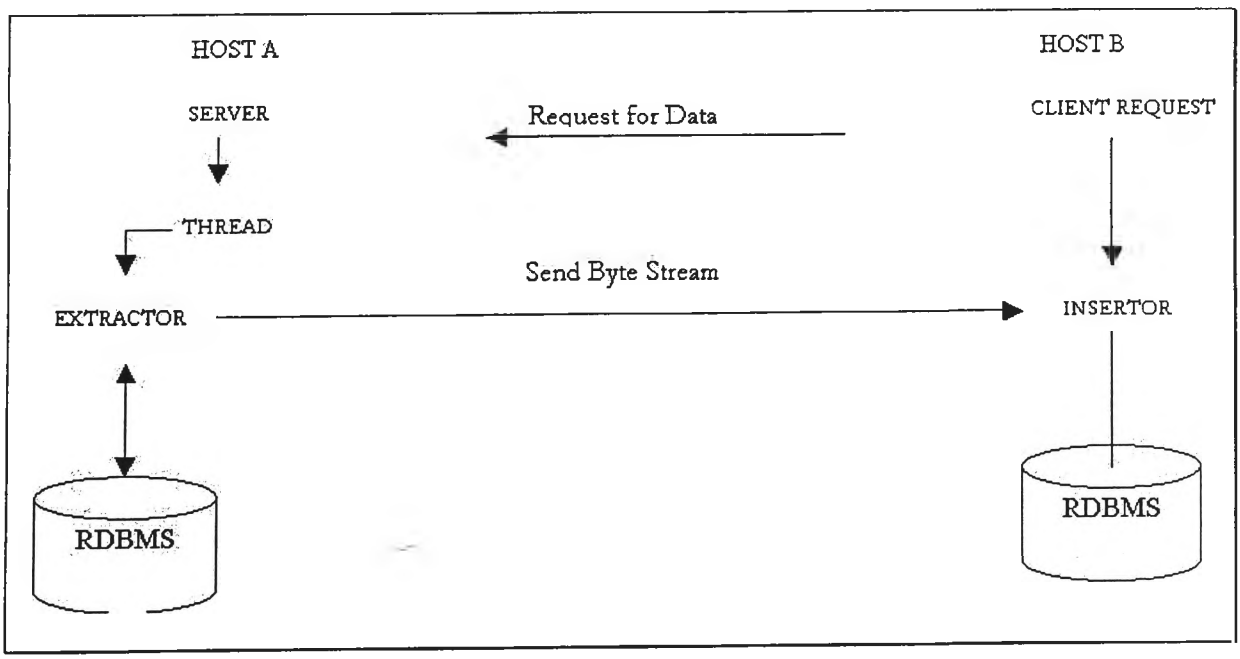

Figure 3.1 The prototypes 'socket based' architecture model

3.3.1 SERVER SIDE CLASS MODEL

At Server startup time, a UserId, Password and Open Data Base Connection (ODBC) Data Source Name (DSN) are passed in as runtime parameters. The Server process must be initiated manually using valid database login arguments, which allows the process to act as a gatekeeper for access to the Data Source Name nominated in the input argument. Figure 3.2 shows a class diagram of the processing and demonstrates the collaboration between objects.

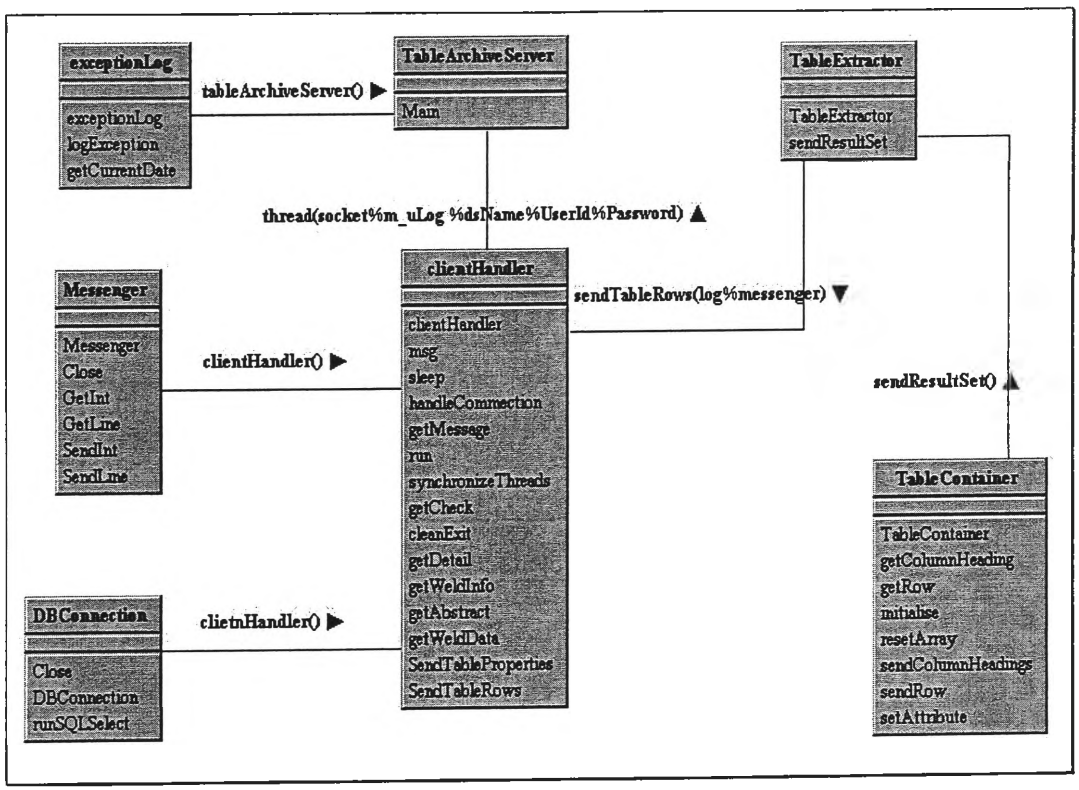

Figure 3.2 Server Class Diagram

PROCESSING

Once instantiated, the Server process listens for requests from potential clients which pass in the name of the table to be transmitted to the client site. Each client request is serviced separately by a 'clientHandler' thread, generated by the Server on receipt of the client request. At application startup time, a synchronised exceptionLog is generated by the Server process, and the address is passed into each thread, which then instantiates a number of other objects to manage the data extraction process

MESSENGER

This class provides a means of writing to, and reading from the synchronous datastreams associated with the socket. The class resides on both client and server, using keyword flags to:

- Signal to the other side what sort of data follows
- Send and receive the data
- Signal completion or error to the other side

For example, server sends a 'S_SEND' flag to the client indicating that data is about to be transmitted. Client uses 'C_QUIT' to inform the server that it wishes to terminate the session

DBCONNECTION

This class acts as a jdbc-odbc bridge for the table management process. Requests from the client, for data are received in the form of SQL statements. Once instantiated, the object executes those requests and stores the results as a Java RESULT SET. The result set is marshalled into a row of 'string' data and passed back via the messenger object to the client for processing.

TABLEEXTRACTOR

The class acts as a container for the table management process as per the following code snippet. tableExtractor class is instantiated to manage the packaging of the resultSet as a whole entity, allowing it to be returned from the called process.

```
ResultSet rs    = null;
rs =  m_DBConnection.runSQLSelect(script);
m_TableExtractor = new TableExtractor (m_uLog, m_Messenger);
m_TableExtractor.sendResultSet(rs);
```

TABLECONTAINER

The class tableContainer is instantiated, then encapsulated within tableExtractor and is used as a type-safe container for the data contained in a single row of a table. At instantiation time, tableContainer data is consigned to the server as an array of strings. The first element (0), contains the number of elements in the array (this number matches the number of columns in the row. The object is resident on both the server and client side. When instantiated on the client side the series of get() methods are used to manipulate and repackage the data into the variable's correct

data type. When instantiated on the server side, the series of set() methods are used to package the data into 'string' data for transport onto the socket datastream.

3.3.2 CLIENT SIDE CLASS MODEL

At client startup time it is assumed the user has decided to fetch some data from a remote host. The following session parameters may be passed in as part of the initialisation process:

Server address

Server port number

Clients (local) ODBC DSN name

Table name

Local database userid

Local database password

The client process instigates a socket connection (Figure 3.3) with the remote (server) site, and the nominated table name is sent to the server as a request argument.

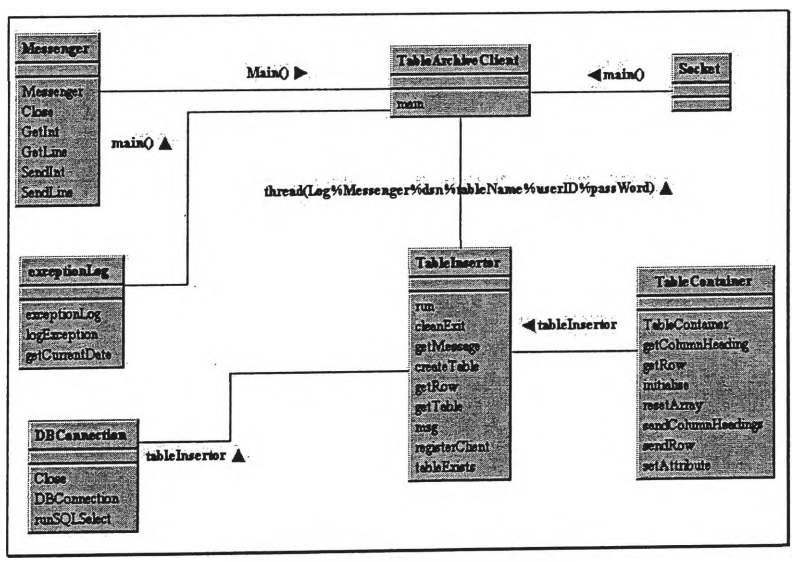

Figure 3.3 Client Class Diagram

MESSENGER

As per the Server model, this object provides the means of writing to and reading from the datastreams associated with the socket.

TABLEINSERTOR

This class manages the insertion of the data into the database. Once instantiated, the object uses a polymorphic instance of tableContainer to prepare each received row for insertion into the local database. The object first determines if the table already exists, if not, tableInsertor sends a message to the Server (clientHandler.SendTableProperties()) requesting the create script for the table. This script is resident on the server and can be shipped as an execution string when requested.

TABLECONTAINER

On the client side, the instance of tableContainer accepts each array of strings from the messenger object and prepares an SQL string which allows the data row to be inserted into the database. tableContainer is used by tableInsertor to house specific data, relevant to the table being processed. Functionally, the object continually calls the messenger object to get the next line and process it, until there is no more data.

DBCONNECTION

In this instantiation, the object is acting to insert the data into the client side database, and will be passed compliant SQL *INSERT* scripts for execution by tableInsertor.

3.3.3 Review Of The Prototype Architecture

The Client/Server communication process can best be described as one of managed 'waiting'. The socket architecture incorporates a blocking mechanism which effectively manages each process (requestor and server), sending it's data through the socket datastream, then monitoring the socket until data is received from the remote communication host. The cost of blocking is that the sending process sits idle much of the time, but by using newer technology components such as the Servlet engine feeding Java code modules, the Framework is able to offset the inefficiency by the use of threads which allow multiple concurrent transmissions. From a re-use perspective, there are a number of existing code modules in the prototype which carry out useful work. The extraction and insertion modules will be retained and remodeled as re-use components in the Framework design because they are already efficient service objects. The extraction and insertion process requirements were clearly identified in the initial analysis, so the code already written

to manage this process can remain, however, an iterative code re-use review (Domain Analysis) is undertaken to determine if there was more potential for re-use. When investigating the use of third party components, priority is given to those areas which can support the end to end transmission process. Initiating the process via a Servlet launch satisfied this desire, as did the use of XML for the external data container. The container classes documented earlier are then be able to be connected to the COTS components by writing 'wrapper' code to facilitate the connection.

The architectural design which resulted from the analysis and post prototype review left a number of issues still unresolved. Priority issues were the user interface for data selection as well as the method of transferring the data to the client host after the extraction had completed.

CREATING A NEW TABLE ON THE CLIENT

A method of creating the table entity within the target database was desirable for those situations where data being moved to the target did not already exist. This is not an issue for the Pipeline project, because the database Schema is static among all host nodes in the system. However, to make the system as generic as possible, a method of creating the table as part of the insertion process would be a requirement.

USER SELECTION OF DATA

This is noted as an issue for a number of reasons, not the least of which is the practical necessity of providing the Web user with a method of nominating the data to be extracted and transport to the desired site. In addition, the prototype only allowed the user to select an entire table for transport. The pipeline application generates large amounts of data for a single weld session. The requirement exists to allow the user to generate adhoc SQL queries for parts of a weld session, ie. A root pass only, or data lying between certain time periods of the weld session.

EXPORT FACILITIES

In order to facilitate data analysis of the weld sessions, analysts use a number of statistical process control applications. The socket version provided no conversion or reformatting facilities, this was seen as a desired deliverable for the Framework.

3.4 COMPONENT INFLUENCED FRAMEWORK

The following sections provide a treatment of design issues which influenced and led to the final Framework architecture. Issues include the user interface, data management via the use of the Document Object Model (DOM), as well as the various component models making up the architecture.

3.4.1 ARCHITECTURAL BENEFITS

One of the underlying design initiatives of the Object Oriented method, and more specifically CBSE, is to provide an integrated, but layered approach to the components [BOURRET00]. This layered approach may be implemented across large and complex service components such as a Servlet Application Server, or down to individual interfaces operating between two code modules. The benefit derived from this layered approach is to reduce the impact of change, because one 'consumed' component may be replaced by another with little or no impact on the co-operating components. Consumed in this case means no longer functional, the requirements have changed and a new component is required. As long as the new component includes the same interface specification, components may be upgraded or replaced with minimal impact on the service requestor of the component [CHEESMAN01]. The benefit of threads, mentioned earlier, is very important to a Web based application. We can assume that any service published via a Web server should, wherever possible, allow the underlying application to make good use of concurrent co-operating processes or threads. The adoption of threads of execution by the larger application components such as Web Servers and Application Servers mean that the developer no longer has this responsibility. Use of a Servlet engine allows the application to spawn a separate thread for each user request, extract the data and wrap it in an XML envelope.

At the client end, the browser based interface allows the user to both nominate selected data and initiate extraction processing on the remote host. The scope of functionality relating to the human interface requirements was expanded by associating an XML Schema with the physical data [ABITEBOUL00]. These increased scope benefits include porting the XML into a predefined Style Sheet, or using the Schema to access to the Data Object Model for alternative output formats. Also, the Schema model provides processing information on how the data is formatted and can be presented for both extraction and insertion processes.

Incorporating an XML Schema into the design greatly improved the potential for re-use of components within the Framework application [CAPRETZ01].

3.4.2 Risk Issues

There is of course some downside associated with any process which converts data from a packed binary form to string data for external containment [BOURRET00]. The use of XML as a container does mean that the size of the document can become quite large [BOUMPHREY98], however, this is offset by the wider range of usage options which XML provides. The World Wide Web Consortium is expected to release a data type standard for XML which will mean that data can be retained in a binary format while still being externally contained [WILLIAMS00], however, the current standard only supports a string (character) data format. Efficiency problems are compounded further when using component technology such as ODBC/JDBC to access the database, as data is made available by the ODBC interface (API) in string format only.

Functionally, the extraction process is achieved using SQL to query the database, with the data returned as a resultset (refer to 3.3.1). This was achieved in the prototype using the well supported **java.jdbc.DriverManager** object, ie. data must be converted to a series of strings for external XML storage. There is potential for more detailed analysis and research within the extraction/insertion process domain, but this would mean moving outside the scope of using CBSE methods to the development of a very specific database interface. All extraction/insertion methods investigated use ODBC components to interface to the database, and the Framework architecture utilises a design model which is similar to a number of generic data extraction/insertion models [JOO01], all of which facilitate:

- Creating of XML documents using data extracted from a database[BOURRET00, STONE02].
- Loading of data from XML documents into relational databases[BERTINO01].

The major distinction with the Framework is the use of an XML Schema to map the extracted data into either comma separated variable (CSV), which facilitates porting to other applications, or XML format.

3.4.3 FRAMEWORK INTERFACE

The browser model was selected for the Framework client because it allows the user to see and select the desired data (Figure 3.4 and Figure 3.5) in a mature and well understood user interface. Once the data has been selected, the interface can be used to provide input to the SQL arguments for data selection.

Copy	Abstract	Tolerances	Batch	Weld_Date	TOTAL_VOLT_F	TOTAL_CURR_F	TOTAL_WFR_F	TOTAL_HEAT_F	TOTAL_DEP_F
☐	Select	Select	910	14-May-00	0	0	0	0	0
☐	Select	Select	909	14-May-00	0	0	0	0	0
☐	Select	Select	910	14-May-00	0	0	0	0	0
☐	Select	Select	909	14-May-00	0	0	0	0	0
☐	Select	Select	910	14-May-00	0	0	0	0	0
☐	Select	Select	909	14-May-00	0	0	0	0	0
☐	Select	Select	909	14-May-00	0	0	0	0	0
☐	Select	Select	909	14-May-00	0	0	0	0	0
☐	Select	Select	909	14-May-00	0	0	0	0	0
☐	Select	Select	909	14-May-00	0	0	0	0	0
☐	Select	Select	909	14-May-00	0	0	0	0	0
☐	Select	Select	909	14-May-00	0	0	0	0	0
☐	Select	Select	909	14-May-00	0	0	0	0	0
☐	Select	Select	909	14-May-00	0	0	0	0	0
☐	Select	Select	909	14-May-00	0	0	0	0	0

Figure 3.4 Data Selection Screen

3.4.4 DATA ISSUES

An American National Standards Institute (ANSI) compliant Relational Data Base Management System (RDBMS) was a prerequisite for the application, and since the Operating System and related application infrastructure would be based on Microsoft Technology, it was decided that the base level RDBMS would be Microsoft SQL Server. This provided a lot more flexibility with regard to data manipulation, access to Stored Procedures, Triggers and if necessary, Cursors. The original prototype used Microsoft ACCESS as the source and target database, which necessitated having to wrap the data elements in specific delimiters before insertion into the target database. As a prelude to moving to a more robust Relational Data Base Management System (RDBMS), some of the data tables from the MS Access database were exported to SQL Server (Version 6.5) RDBMS (with no trouble). However, when the ODBC DSN was modified to point to a SQL Server DBMS, the escape codes for quotes surrounding numeric data types were not acceptable to SQL Server. SQL Server was also more stringent about validation of dates which was to be expected.

While these are typical third party connection issues, using a product that is ANSI compliant reduces the need to modify the insertion script to suit the database Vendor. There are a number of intangible benefits which also accrue from using a more industrial strength RDBMS, including:

- SQL Server comes with an administration module which caters for distributed management, which allows a remote user to create and implement database storage facilities, administer user access and remove aged data.
- Provide remote access to data logs
- The latest version of SQL Server facilitates the output of data in XML form, which means the Framework can be further enhanced to accommodate the use of components to replace the extraction process, if this proves to be an ongoing feature.

Overall, the concept of integrating 'best of breed' components should be promoted when those components are sourced from a third party Vendor. For this reason alone, it was decided to upgrade the database to a more robust product such as SQL Server. Both MS Access and SQL Server are both Microsoft products and are fully compliant with ODBC, so the choice to migrate was straight forward.

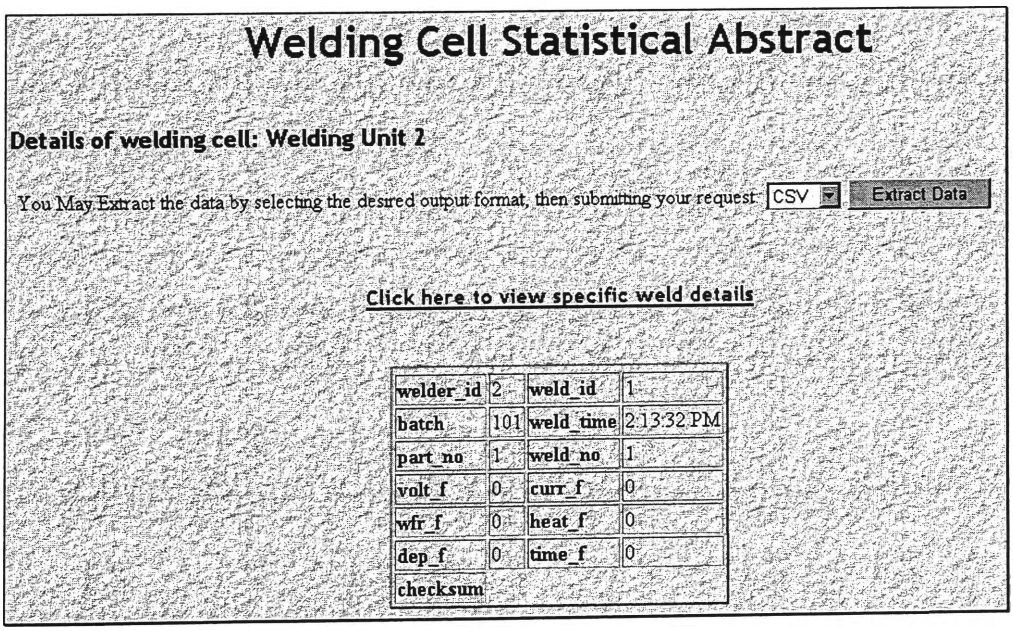

Figure 3.5 Specific File Selection

3.4.5 EXTRACTION INTERFACE

The Framework provides a common interface to accept the selection criteria for an extraction. A parameter object runParms, allows an extensible list of arguments to be

passed in, and made available to the SQL WHERE clause as shown in figure 3.6. Typically, access to the data is provided by a Web Server, with ASP/HTML used for the interface. Using the browser, the user may first view, then select the data as per figure 3.5. Data is presented to the user in the standard HTML *table* format using a generic method of extracting the selection criteria which allows the table name, key/index information to be passed into the SQL WHERE clause of the select statement, to be used by the Data Extraction module. In order to manage this process, a **runParms** parameter class was developed to allow the desired table and associated keys to be passed in and managed in a standard manner.

```
<input type=submit name="Button" value="Extract"></td>
<input type=hidden name="Table" value ="Weld_Description">
<input type=hidden name="KeyName 1" value = "welder_id">
<input type=hidden name="KeyValue 1" value = <%=WelderNumber%>>
<input type=hidden name="KeyName 2" value = "weld_id">
<input type=hidden name="KeyValue 2" value = <%=WeldNumber%>>
```

Figure 3.6 Data Selection script (HTML)

The parameter names are case sensitive and values may be supplied as constants, variables, or, supplied dynamically by accessing the Host Database for required values. This means data can be passed to the Application Server as HTML or as arguments in the Servlet URL. This is an extremely important point and is covered in more detail in the Servlet Design Section, but the importance relates to the matching of 'Name' and 'Value' pairs which are ultimately used to generate the SQL extraction request. The numeric increment attached to the matching name and value is required because the Servlet Model does **not** receive the arguments in any particular order, so a method is required to maintain the consistency between the name and value pairs.

This feature allows the Framework to extract and transport any nominated data without the need to extend the coding functionality, the name and value pairs are used to increase the granularity of the SQL WHERE clause (refer to the Implementation and Usage chapter), as per the example in figure 3.7.

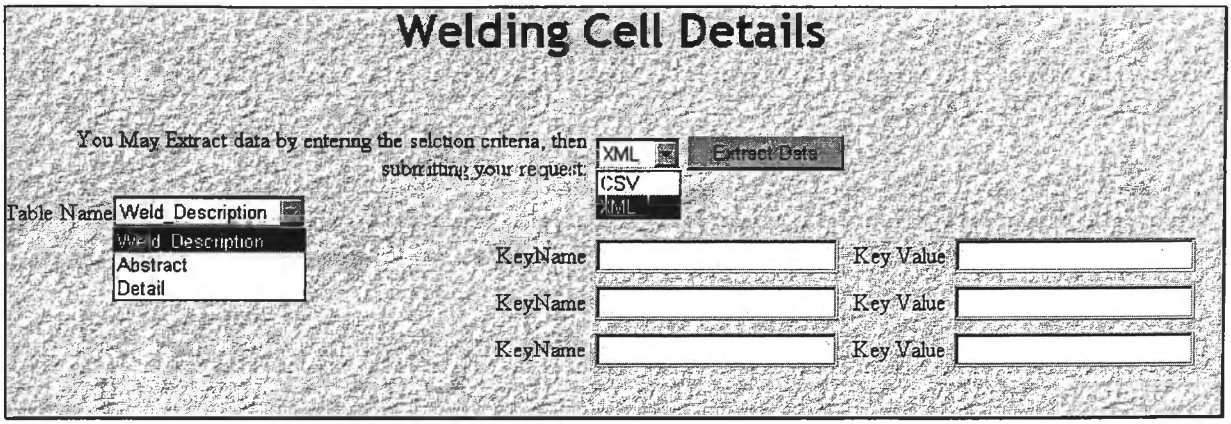

Figure 3.7 Granular Data Selection

Configuration information also needs to be passed into the Framework and loaded into the runParms object. The runParms interface requirements are fully documented in the User Manual located in appendix C.5.

3.4.6 SCHEMA MODEL

The Frameworks' main goal is to provide an efficient means of processing large volumes of data being transported from one host to another. Once the user selects data to be extracted from the database, a number of processing options are available. All are facilitated by the Document Object Model (DOM) which facilitates the processing and formatting of data for the extraction, persistence, or insertion processing carried out by the Framework.

Extraction - The DOM provides data definition information to the extraction process, indicating how many data columns are in the selected table rows, and how those data items need to be processed in order to convert the data to string format.

Persistence – The DOM provides formatting information to the physical data output process, ie. When writing the data to file in either XML or CSV format.

Insertion – The DOM provides data definition information to the insertion process running on the client host, indicating how to build a row of table data for insertion. For example, string data types need to be bracketed with quotes in order to successfully be inserted into the database.

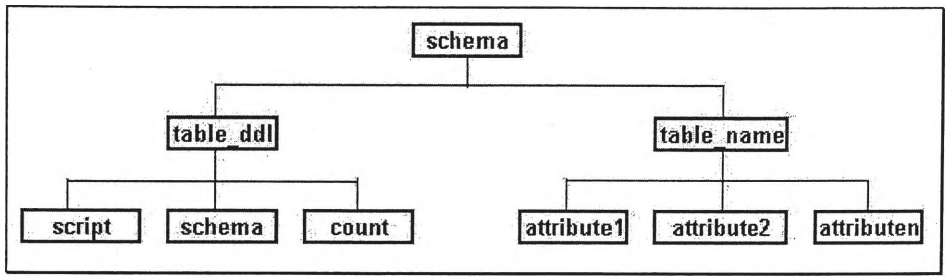

Figure 3.8 Document Object Model

It is the Schema which makes this information available to the in-memory DOM, and this is achieved by a component called the xmlManager (refer to the class diagram in figure 3.11). When the xmlManager class is instantiated, it is passed the name of the relevant database table Schema to load (refer to figure 3.9). The name of the Schema is usually provided as a Uniform Resource Locator URL, and is typically encapsulated within the XML document as per figure 3.10. In that example, the Schema abstract.xsd in the figure is located at name space http://192.168.1.254/xml.

The Schema URL is also stored within the extraction database, when xmlManager is instantiated it will be passed a URL, the calling component in this case will retrieve the Schema URL from the database. Once the xmlManager object locates the Schema, then loads the Document Object Model (DOM) from information provided, the DOM is then used to create a mapping for data to be extracted, persisted or inserted back into the remote users database. The relevant Framework service has access to the in memory tree and places service requests on xmlManager via relevant public methods. An application specific Schema has been developed for the Framework, and in the Framework Schema, there are two elements, one describing the table data definition language, and one describing the table data.

```
- <abstract>
    <schema>http://192.168.1.254/xml/abstract.xsd</schema>
  - <row>
      <C1>2</C1>
      <C2>1</C2>
      <C3>101</C3>
      <C4>1899-12-30 14:13:32.000</C4>
      <C5>1</C5>
      <C6>1</C6>
      <C7>0</C7>
      <C8>0</C8>
      <C9>0</C9>
      <C10>0</C10>
      <C11>0</C11>
      <C12>0</C12>
      <C13>null</C13>
    </row>
  </abstract>
```

Figure 3.9 Data-Centric XML

The Schema represented in figure 3.10 shows an example table abstract and indicates the Schema has two main elements, the first table_ddl, and the second refers to the table being transported, in this example table abstract. Once the Schema is located and successfully loaded, the Framework then produces a DOM tree as per figure 3.8. The xmlManager object instantiates the DOM and encapsulates the Element objects table_ddl and abstract, and subordinate child attributes for each of the elements.

ELEMENT 'TABLE_DDL'

This element has three child attributes which provide administration information available to the source or target application which is producing or processing the XML document.

Script: This attribute provides a data definition script to the target application if the table (in this case abstract) does not exist in the target database. When the session is established, the script attribute is used by the Framework to create a table in the database. The script is executed if the table does not already exist in the target database. It is assumed the tables are built on the target database as a means of analysing the data produced on the source machine, rather than securing the data, as such, there is no key or relational constraints included in the script, it is provided only as a means to provide a repository for the received data.

Schema: This attribute is used by the Source Application to 'bury' the contents of the attribute into the XML document which is being produced. Typically, the URL value relates to the Source Host, but could be a Framework wide URL which manages the terms of reference of the Framework.

Count: This attribute provides a validation check for the application. The numeric value of the count attribute must match the number of attributes which are identified by the abstract element. If not, a Parser error occurs during the extraction/insertion and processing stops. The Schema is prepared as a user function (refer to User Manual), so the count attribute is used by the Framework to match against the number of columns which are nominated in the extraction query. The example shown in figure 3.9 indicates a count value of 13, which must match the number of attributes listed under the abstract element. Moreover, when the Framework Source Host extracts the data from the database and begins to assemble the XML output structure, the column count in the query result set must be 13.

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema">
        <element name="table_ddl"/>
                <attribute name="script" value="create table abstract (welder_id int not null, weld_id integer
                not null, batch int not null, weld_time datetime not null, part_no int, weld_no int, volt_f
                smallint, curr_f smallint, wfr_f smallint, heat_f smallint, dep_f int, time_f int, checksum int);
                create unique index abs_idx on abstract (welder_id, weld_id, batch, weld_time);"/>
                <attribute name="schema" value="http://192.168.1.254/xml/abstract.xsd"/>
                <attribute name="count" value="13"/>
        <element name="abstract"/>
                <attribute name="welder_id" title="Welder Id" code="C1" type="Integer"/>
                <attribute name="weld_id" title="Weld Id" code="C2" type="Integer"/>
                <attribute name="batch" title="Batch" code="C3" type="Integer"/>
                <attribute name="weld_time" title="Weld Date" code="C4" type="Datetime"/>
                <attribute name="part_no" title="TS Set" code="C5" type="Integer"/>
                <attribute name="weld_no" title="weld_no" code="C6" type="Integer"/>
                <attribute name="volt_f" title="Volt" code="C7" type="Smallint"/>
                <attribute name="curr_f" title="Curr" code="C8" type="Smallint"/>
                <attribute name="wfr_f" title="WFR" code="C9" type="Smallint"/>
                <attribute name="heat_f" title="Heat" code="C10" type="Smallint"/>
                <attribute name="dep_f" title="Dep" code="C11" type="Integer"/>
                <attribute name="time_f" title="Time" code="C12" type="Integer"/>
                <attribute name="checksum" title="CHECK" code="C13" type="Integer"/>
</schema>
```

FIGURE 3.10 ASSOCIATED SCHEMA

ELEMENT 'ABSTRACT'

This element describes the underlying table which is to have the data queried and extracted. There is an attribute for each of the columns in the table being nominated with the following properties:

Name: Which must match the column identifier within the table. The Framework target application uses the name attribute to prepare the 'INSERT' statement for the Insertion process.

Title: The Framework target application uses the title attribute to prepare hard copy reports of the content being inserted into the database as a Column Heading.

Code: To facilitate efficient transportation of the data, a meaningful tag name for the column is replaced by a code. The Framework target application uses this attribute within the XML document to match the against the Schema, then used the corresponding name value to build the INSERT statement.

Type: Which must match the column identifier within the table. The Framework target application uses this attribute to prepare the 'INSERT' statement for the Insertion process.

3.4.7 DOCUMENT OBJECT MODEL

The xmlManager component class is an integral part of the Framework architecture and manages the instantiation of the document object model (DOM) specific to the database table being targeted by the user. Once instantiated, the xmlManager object encapsulates two subordinate objects, elementProperties and elementAttributes, which map explicitly to the elements and attributes located in the associated Schema. In effect, the Schema is a static representation of the in-memory DOM.

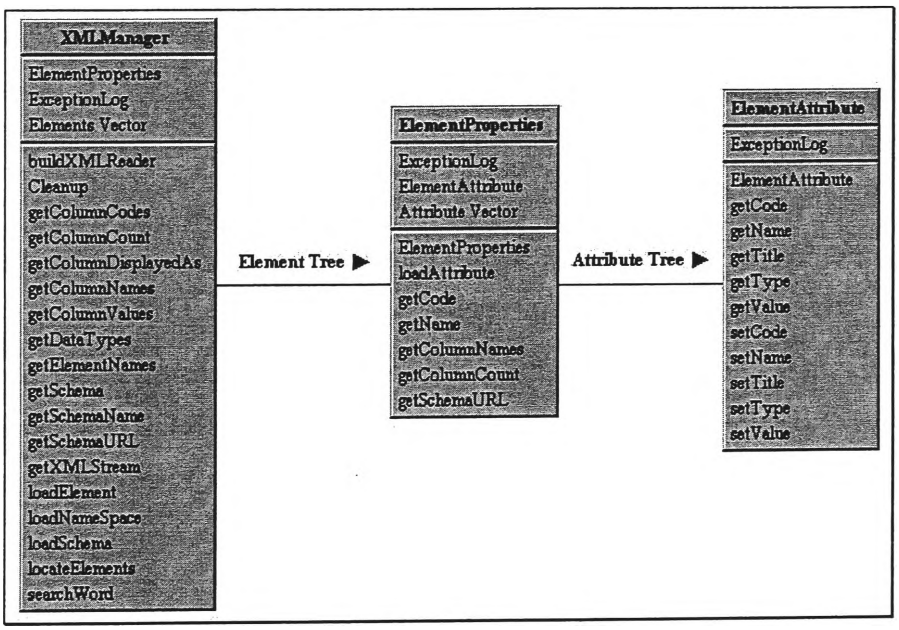

Figure 3.11 XMLManager Component

ELEMENTPROPERTIES

The design follows the standard of beginning the name of public methods with 'get', and each of the service request methods which return data from the DOM, receive that data as an array string. The Schema is loaded by instantiating two Java ElementProperties objects, one for the table_ddl and one for the table_properties (abstract in the example), access to these classes is provided by attaching the Element Object to a Vector (array), then nominating the element from which data is to be provided; as per the example

```
public int getColumnCount()
    {
      ElementProperties l_element;
      // the number of columns is stored in the 1st element vector
      l_element = (ElementProperties)v_elements.elementAt(0);
      return l_element.getColumnCount();
```

```
}
```

which indicates that the number of columns getColumnCount() is stored in the 1st element vector, and as such, the developer may refer to this explicitly as

```
l_element = (ElementProperties)v_elements.elementAt(0);
```

Using this method is much more efficient than using a standard binary tree method, because the developer knows exactly where the required data is, and does not have to 'walk' the tree, comparing values to the those required. The data is object based, and can be immediately accessed, by the public 'get' methods, as per:

```
public String [] getColumnNames ()
```

After xmlManager instantiates the elementProperties classes, each object is called upon to load the matching attributes (loadAttributes), for each of the elements. This is achieved by instantiating as many instances of child elementAttribute classes that are referenced in the Schema, and associated with each Element. As shown in figure 3.10, there are 13 attributes associated with the abstract element, these attributes, each have Name, Code, Title, Type and Value properties. Public access to the data is then made available by calling the appropriate 'get' method in the elementProperties object. Note, that direct access to the elementAttribute object is via public method calls to the elementProperties object. xmlManager also validates the XML document against the Schema by matching the boundary of the array against the count value stored when the Schema is loaded. The public methods are:

GetCode: Returns an array of each attribute code nominated as children of the element. (C1... C13 in the figure 3.10 abstract example).

GetColumnCount: Returns the published count of the number of attributes listed in the Schema under the table name element (13 in the figure 3.10 abstract example).

GetColumnNames: Returns an array of each attribute name nominated as children of the abstract element.

GetName: Returns the published name of the element. ('abstract' in the figure 3.10 abstract example). This name also matches the table name of the data being extracted from the database

GetSchemaURL: Returns the published URL of the Schema. The URL is placed into the XML document being produced by the extraction process. The URL is then accessed by the target application to build the DOM for processing the content of the document.

ELEMENTATTRIBUTES

While the methods for returning data are public, only the elementProperties object directly accesses the elementAttributes objects. The design instantiates each of the named attributes in the Schema, and provides access to the data via the parent ElementProperties object.

3.4.8 SERVLET MODEL

Using the Servlet model allows the designer to utilise the Internet transport facilities and abstract the entire end to end data transmission. The session is controlled by IIS (Microsoft's Internet Information Service) technology running on the Servlet host, managing the interaction between the Web Server and the Application Server (IIS Internet Service API, ISAPI). The extraction process is initiated as a Java thread of execution, the Schema is located by using the table name (which is passed in as an argument) to access a table containing the associated Schema URL. The user has the choice of extracting the data in either XML or CSV format. Moreover, if the user elects XML, this data can then be dropped into a directory on the Target Host which is being monitored by the 'background' insertion service.

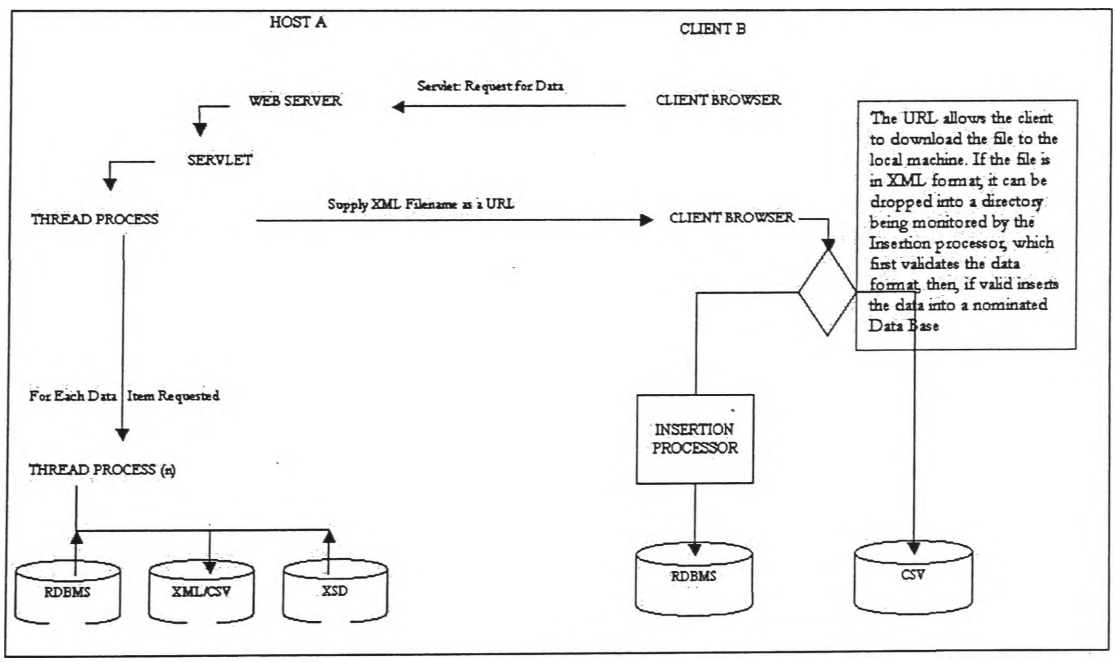

Figure 3.12 Architecture Diagram

Once extraction has been completed, the client is presented with a screen (figure 3.14) which contains a link to the XML file generated by the server side host. On clicking the link, the user may download the file to the local machine in the desired format. The system monitors a directory being monitored on the users platform,

and when the downloaded XML file is 'dropped' into the directory being monitored, a daemon process opens the file and attempts to process the contents, by inserting the data into the users local database. This is facilitated by having the daemon start up at system initialisation time, with runtime parameters indicating the location of the database.

3.4.9 SERVER SIDE CLASS MODEL

This model consists of a number of classes which are instantiated by the Servlet object (servletExtractor).

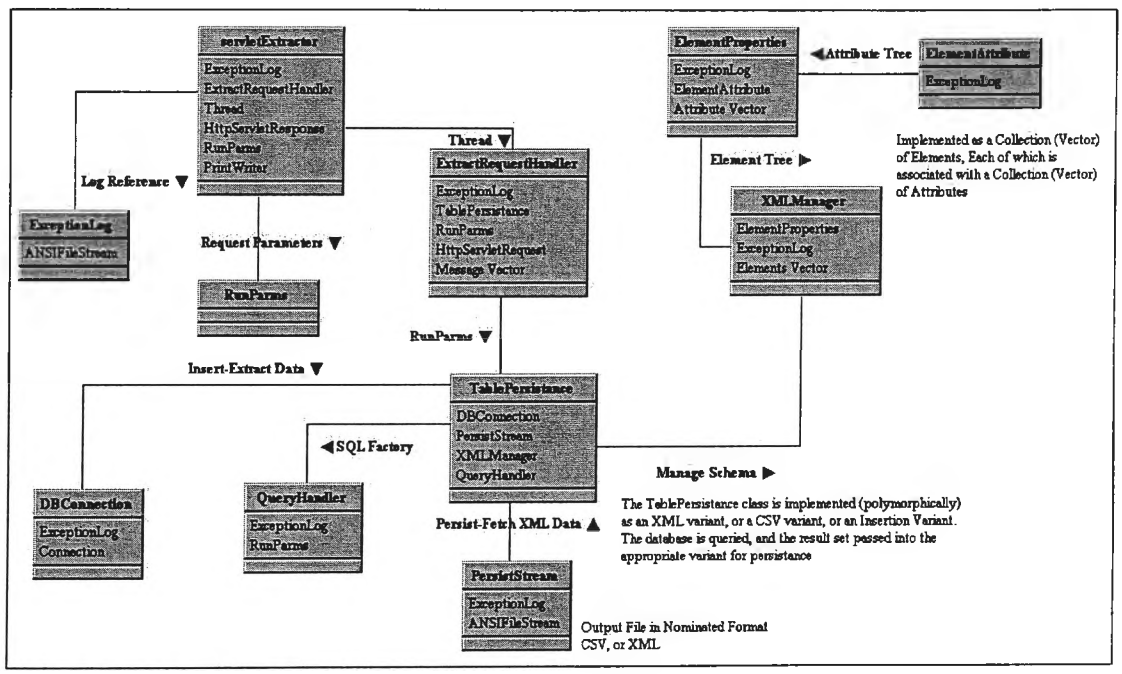

Figure 3.13 Extraction Class Diagram

RUNPARMS

As outlined earlier, the runParms class provides a container for the argument pairs to be stored and retrieved by the various processing classes which use the arguments. The class provides methods for:

- Adding parameters and related values
- Retrieving the value of a nominated parameter name
- Retrieving a list (Listiterator) of the parameters
- Retrieving a list (Listiterator) of the Values

Once instantiated, the object is requested to retrieve the value of a nominated parameter name

Eg. `s_table = m_runParms.getParameterValue("table");` the runParms object iterates through the current list of parameters looking for the keyword 'table'. If found, the object returns the matching value, in this case, the name of the table being extracted. To maintain a generic interface, all arguments are stored as Strings, and the requesting object is responsible for casting the result to the appropriate data type.

EXCEPTIONLOG

On both the client and the server side, the log is created at initialisation of the processes. On the server side, a reference to the log is then passed to each Thread of execution. The exceptionLog class has a setLog() method which accepts a Boolean value. When set to *False*, the log is only incremented in the case of a Terminal error. When set to *True*, the log acts as a Debugging tool, outputting code variables set by the developer or system maintenance personnel, eg:

```
m_uLog.LogMessage(kTHIS," Could not determine the Host file location, mandatory data - ends");
```

Depending on the developers needs, the exception can be used as a debugging tool, or true exception handler, which can either terminate processing, or pass the exception back up the chain to be handled by a higher level calling module.

At runtime, the object is public and 'synchronized', and is called by passing the exception and an accompanying message eg:

```
LogException(String excpt, String msg);
```

SERVLETEXTRACTOR

This class is a standard Servlet class which receives the HttpServletRequest and HttpServletResponse from the Servlet engine, instantiated and managed by the Application Server Host. The Application Server instantiates a servletExtractor object thread each time a request is received by the Web Server. User data passed in from the Servlet engines' httpServletRequest are encapsulated within a runParm object (refer to figure 3.5), the data is extracted from the associated database and persisted in either CSV or XML format for transport to the users nominated site.

Operationally, servletExtractor instantiates a manager Thread (extractRequestHandler) to deal with the users request. The Thread communicates with the parent (servletExtractor) and passes status information back to the users Browser (refer to figure 3.14). This is achieved using a Vector which is instantiated by the Parent (servletExtractor) and passed into the spawned Thread by reference. The parent then monitors the Vector until the response is prepared. The response is supplied once the parameters needed to service the request have been successfully loaded, and a unique file name has been allocated. This allows the user to be informed if the httpServletRequest parameters cannot be loaded in the correct name/value pair sequence.

EXTRACTREQUESTHANDLER

This class is instantiated to determine the type of request (using the runParms object). Once the type is determined, the object then calls the appropriate processing modules. Options are either CSV or XML, and in both cases the extraction request is persisted to the local hosts file system. A reference to runParms is passed in to allow the type of request to be determined, ie. CSV or XML, and an appropriate persistence object is spawned and processed. The data file generated is deposited into an externally accessible directory, which is subordinate to the Web Server directory ie. http://host/ServletData. A URL is presented to the user which points to persisted data on the host Web Site eg.

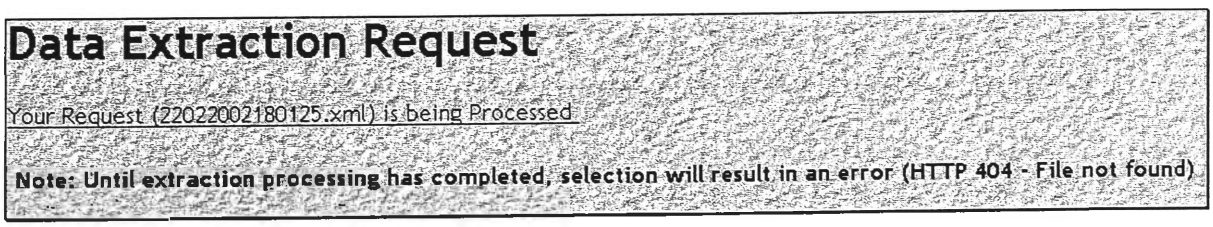

Figure 3.14 Data Extraction Process

A warning is presented to the user indicating that the file may not be immediately ready for extraction, this message is posted to ensure that if the file is large, the user cannot access the file until extraction is complete. The method used to safeguard the data is to pre-pend a temp_ on the filename, which is renamed to the correct filename on completion of the extraction. Processing first determines a unique file name for the data file, the user is then notified of the File Name, and processing continues to persist the data (Refer to the servletExtractor class documentation for a reference to the notification method).

TABLEPERSISTANCE

This class is an abstract class, implemented as either persistanceXML or persistanceCSV. The generalised (parent) object contains methods which perform the following processing requirements:

- Determine the unique file name for the specified file type
- Locate and make available the URL of the nominated extraction table from the Framework database
- Locate and make available the XML Schema associated with the SQL request made by the user
- Locate and make available the destination namespace of the file generated as a result of the extraction process
- Execute the SQL request and make available the resultSet data stream to the requesting object
- Launch the extraction process

These methods are then called, as needed by the specialised child object.

XMLMANAGER

The component class is instantiated on both client and server, and plays a different role when used by each requestor. In both cases, the object has to carry out an administrative load of the Schema (XSD) to create an in memory DOM (3.3.4), then provide an information service to client requestors. When instantiated on the server side, xmlManager is encapsulated within the tablePersistance variant, and the objects main task is to build faceplates[MOHR02] to map the resultSet stream to the physical file format selected by the user. Once instantiated, the object locates the Schema namespace from the 'schema_control' table within the host database, then loads the Schema (XSD) file using a URL openConnection method for example:

```
m_xsd = new URL(getSchemaSource());
m_xsdConn = m_xsd.openConnection();
```

The Schema is used to build a set of in memory collections containing the properties of the table selected by the user. From a service perspective, the object provides a public interface to requestors of information who are building the respective physical file formats.

PERSISTANCECSV

In object oriented terms, this class is implemented from the abstract tablePersistance class, and works in co-operation with the xmlManager object to generate the physical data file in CSV format. At constructor time, the object creates a file, uniquely named with the csv extension. The next task is to generate a row with the column headings, derived from the Schema. Once this is completed, the object then streams in the resultSet data, and generates the output row by inserting a 'comma' in between each column value, then output the row the persistStream object.

PERSISTANCEXML

This class is also implemented from the abstract tablePersistance class, and works in co-operation with the xmlManager object to generate the physical data file in XML format. At construction time, the object creates a file, uniquely named with the XML extension. The object outputs the mandatory XML standard tags, then wraps the output from the resultSet data in the appropriate data tags and outputs the row via the persistStream object.

PERSISTSTREAM

The class is inherited from the ansiFileStream (refer to Source Code), and simply persists the data passed to it. The ansiFileStream class is itself inherited from the standard java randomAccessFile class which provides an interface for streaming data to a physical file format.

QUERYHANDLER

The class is used to prepare request parameters for execution by DBConnection. The components needed to format the SQL request are located from the runParms object, and passed in at construction time. The object is necessary because the number of statements making up the SQL WHERE clause is arbitrary. The user may pass in numerous key value pairs, which need to be built into the appropriate WHERE clause. This object co-operates with the runParms object to manipulate the key value pairs and prepare the correct SQL string for the DBConnection object to process.

3.4.10 CLIENT SIDE CLASS MODEL

This model consists of a number of classes which are instantiated at system startup, and run as a stand alone deamon process, continually polling a nominated directory on the client host. The directory name is maintained in an application control table, set up in the system inialisation process.

Once the Insertion process has commenced, the target host generates an entry into the target database providing information on the status of the insertion process in relation to the data being inserted into the target database. The entry contains information such as time/date, file name, source host ip address, table name and the number of records in the transmission. Once the Insertion process has completed, the status is set to completed. If a failure occurs, the data is rolled back, and the log is updated to reflect the rollback status.

TABLEARCHIVECLIENT

This process continually monitors a nominated file directory until a file is deposited into the directory with an XML file extension. Once the processor sees a valid file, it attempts to process the data using the local (host) Web Server to facilitate a URL connection to read the XML file. Based on information received from within the XML file, ie. the name of the XSD Schema, and the name of the table to be actioned, the Schema Source Data is fetched as a stream via a URLConnection.

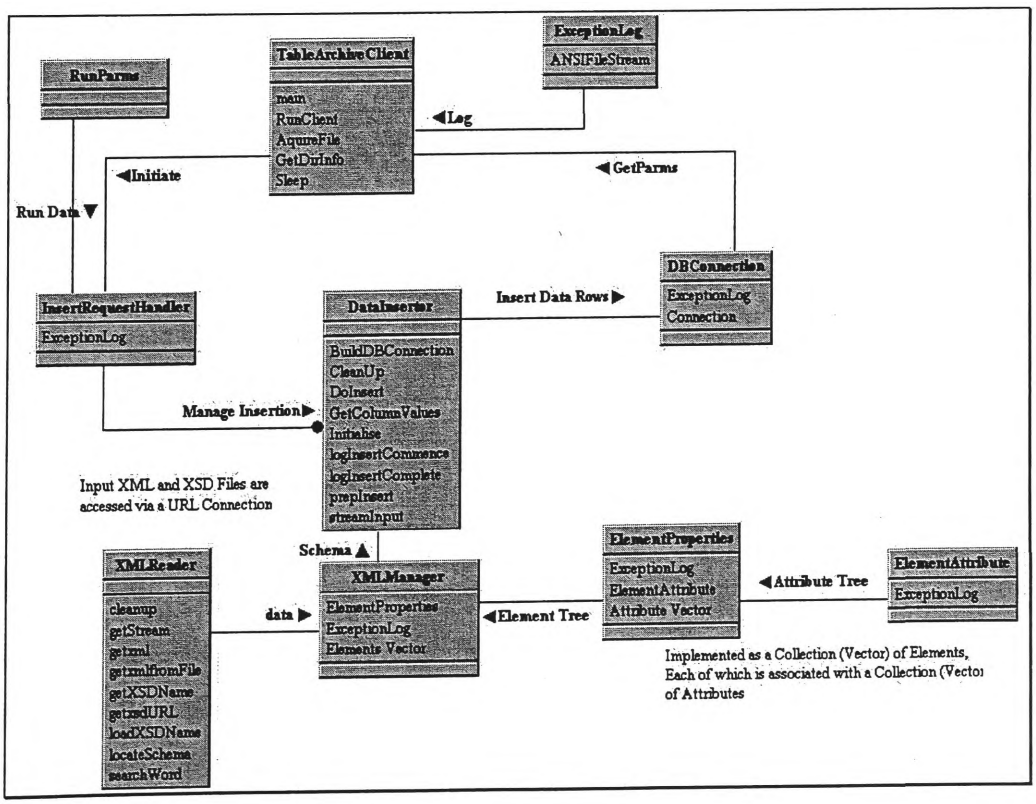

Figure 3.15 Client Class Diagram

The process instantiates an exception log at startup to log events over the coming session. The TableArchiveClient object is responsible for initiating the InsertRequestHandler, which validates and ensures the resources necessary to insert the data into the RDBMS are available.

INSERTREQUESTHANDLER

This class, whose main task is to gather the parameters for the processing task, then act as a wrapper for the dataInsertor object. This class simply sets up the appropriate environment required by the dataInsertor object.

XMLMANAGER

When instantiated on the Client side, the xmlManager object still needs to locate the Schema associated with the table nominated by the requesting user. The method of locating the Schema is different on the client, the object must first open the XML document, then locate the schema within the document before using the extracted URL. In this case, the object co-operates the xmlReader object to facilitate the extraction of the Schema data from the XML file. Once the DOM is created, the data is read in from the file, and mapped to the database using the dataInsertor object

XMLREADER

The class is used as a helper for xmlManager, and carries out the physical preparation of opening the data file and passing the stream to xmlManager. Overall, the class has the following responsibilities:

- Locate and make available the XML Schema associated with the SQL request made by the user
- Locate and make available contents of the XML data components

DATAINSERTOR

This class is a variant of the tablePersistance class but in this instantiation is used to manage the insertion of data into the requestor's database. The commonality with the tablePersistance class lies in the usage of xmlManager and DBConnection classes. At runtime, the objects co-operate to retrieve and read the XML Schema from the

local host, and manage the insertion of data into the database. Once the target XML file is opened, the Schema tag

"<schema>http://192.168.1.254/xml/welddesc.xsd</schema>"

provides the URL of the Schema XSD file which is used to unpack the XML data. Once the XML stream is accessed:

- The object determines the existence of the table in the target database
- Executes a build table routine (if necessary)
- Builds an INSERT statement using the attribute names provided by the Schema
- For each XML row, marries the matching data into the INSERT statement, ie. VALUES (…..).

Each completed row is passed to the DBConnection object for insertion into the database. Any errors which may occur, are dealt with by the object. The most typical error is a 'duplicate key', meaning the data has previously been inserted and is ignored by this insertion process. Another less likely scenario is that the table format has changed, but the Schema has not been updated to reflect the new layout of the table. In this situation, the transport and insertion process is terminated. A run log has been incorporated into the requestor's database, which is used to store details about the success/failure of transport sessions. At the beginning of the transport session. A container object is assigned to store status event messages from the session, on completion or termination of the insertion process, the container uses the DBConnection to insert logging information about the session into the database. The user requesting the transport can access this log via SQL query within the dqm_log table, location of the session details is provided by way of entering the physical name of the XML file being transported.

3.5 SUMMARY

The aim of this chapter has been to describe the architectural components of the Framework, and, while the concepts themselves are not complex, describing the mechanics of how those components interoperate was a challenge. The desired outcome of the chapter was to develop the reader's understanding of the architecture by documenting the mechanics of extracting/inserting data from/to a database when using a Schema driven document object model.

The following chapter also reviews the Framework from a post construction perspective but focuses on issues such as:

- Determine the effectiveness of using component based software engineering methods in the development of software applications.
- Gauge the adaptability of CBSE on post implementation maintenance to determine whether these methods make change easier and less costly, or otherwise.
- Use the construction of the transport Framework as a case study to highlight potential risks and identify areas of vulnerability when using CBSE methods.
- Investigate both the benefits as well the risks associated with integrating a new middleware technology such as XML across the range of COTS Web components.
- Review the Object Oriented principles used in the development to determine if this added value to CBSE method, particularly in the areas of interface, re-use of code and ease of assembly and integration of the components.
- Whether construction of the prototype actually improved the Domain Analysis, and facilitated the exposure of candidates for re-use.
- Whether adoption of the XML Schema model provides genuine extensibility to the Framework.

These issues are central to the thesis and hopefully allow us to determine both the merits and risks software engineers face when developing software using CBSE methods.

4 POST IMPLEMENTATION REVIEW

4.1 INTRODUCTION

Research indicates that there is general consensus regarding the benefits of integrating pre-built components into a software application in order to reduce development time. There is also agreement regarding the use of COTS components to solve a well identified processing requirement. Microsoft and IBM, just to name two organisations, would not invest hard earned profits in these areas if there was no market for their products. The real problem, and this is the focus of this thesis, is in attempting to highlight the more subtle issues which result from the adoption of CBSE methods. Implementation and usage of the Framework allows many of the research issues (refer to the summary at the completion of chapter 3) associated with the design, construction and usage to be addressed. In this chapter we review the Framework from a post construction perspective to try to highlight some of these issues, especially software quality and risk potential when change or maintenance is required to the original system. Specifically, software quality and risk are addressed from a number or perspectives:

- Integration of Common Off the Shelf Software (COTS) components [GANESAN01] with custom code
- The use of a formal review of the problem domain at the completion of each build iteration, to maximise the potential for re-use of custom code [CAPRETZ01].
- Clearly defined interfaces to facilitate the later replacement of custom code with third party COTS components.
- Using XML as a data container to further facilitate the level of COTS integration.
- Use of the Schema to deliver the Document Object Model (DOM).

In order to provide a context for discussion, this section begins with a walkthrough of the extraction and insertion processes. Focus then moves to research issues using the Framework as a basis for discussion of those issues. The final section in this chapter identifies areas of future work which may lead to improvements in the efficiency of the Framework.

4.2 WALKTHROUGH

As can be seen from the Framework architecture diagram in figure 4.1, a typical extraction scenario is for a remote Internet user on host B to:

- Place an extraction request on the Server host A
- Have the request serviced, ie. remove the data from the repository and convert it to XML
- Transport the XML file over the Internet to host B
- Re-insert the data into the user's local database, or another repository on host B

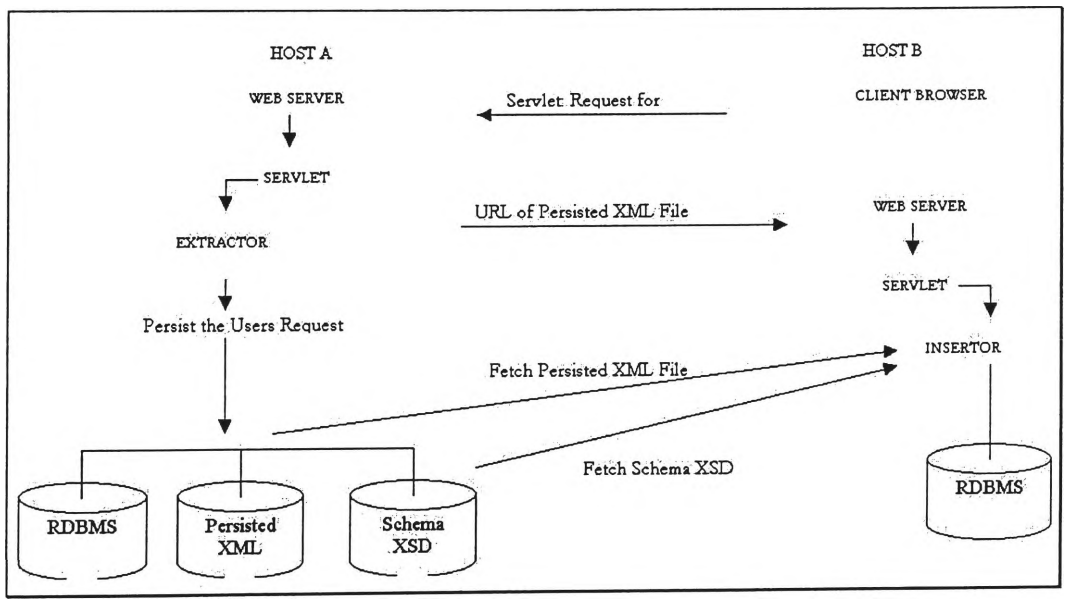

Figure 4.1 Architecture Diagram

The walkthrough first describes the user's runtime environment and data selection process which culminates in a data extraction request. Processing the extraction request is then discussed in some detail, and includes an explanation of how the extraction arguments and configuration data are passed into the extraction process. The XML file produced as a result of the extraction is then transported to the user's host for insertion into the user's local database. Data transmission time over the Internet is a product of bandwidth and latency. Outside the confines of the local area network, transmission time can be considered a relatively inconsistent variable, so we do not include the issues of bandwidth or latency within the walkthrough scenario. Transmission of the XML or CSV file over the Internet is initiated from

within the user's browser by selecting the link generated by the extraction process, as shown in figure 4.3. The re-insertion process is then presented with an explanation of the mechanics of that process.

4.2.1 RUNTIME ENVIRONMENT

The Framework provides a common interface for inputting the desired selection criteria for an extraction (figure 4.2). The browser method is used for the walkthrough, and has the task of passing in the nominated data for the extraction request. Functionally, the browser allows the user to view the data prior to extraction. Request parameters are passed into the Framework via HTML and all user access with the system is via the Web Browser. Data can be requested locally (subnet) or externally (Internet) using the data site's Web Service to initiate and manage the extraction process. If the remote user is intending to populate their local database with the results of the extraction, then the extraction format needs to be XML. In addition, all target machines need to have the Java Runtime (JVM) installed to facilitate the insertion of data into the local database (refer to the User Manual in appendix D for setup details). However, if the user intends to port the extracted data into a third party application such as Microsoft Excel or Harvard Graphics for statistical analysis, then no additional software is required. If this is the case, the user selects CSV format for the extraction request.

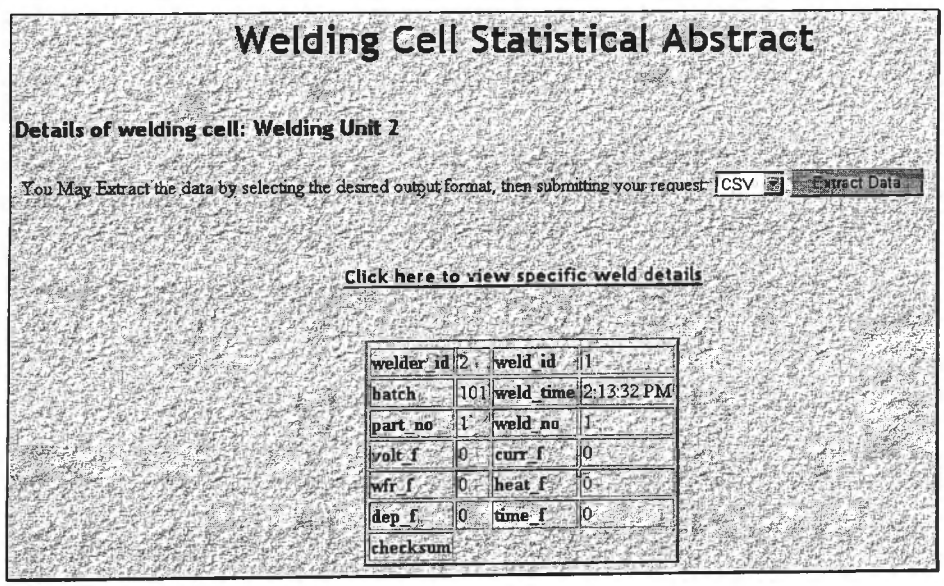

Figure 4.2 Data Extraction Request

4.2.2 DATA EXTRACTION REQUEST

Once the extraction request has been received by the Servlet engine, a processing thread is initiated by the Servlet engine to deal with the request. The request

arguments are typically presented as an HTML request and have the following format:

http://192.168.1.254/servlet/servletExtractor?Format=XML&Button=Extract+Data&Table=Weld_Description&KeyName+1=welder_id&KeyValue+1=2

The request can be broken down into the following sub components:

URL	http://192.168.1.254/servlet/servletExtractor
Format	XML
Table	Weld_Description
KeyName+1	welder_id
KeyValue+1	2

All requests to the Framework must be in this format, the complete list of allowable parameters is provided in the User Manual.

URL: Names the host and the extractor module launched (as a thread) by the Servlet engine.

Format: There are two formats currently supported by the Framework, CSV (Comma Separated Variables), and XML (Extensible Markup Language). The request must specify the output format to the extraction module.

Table: This parameter refers to the table being queried for data extraction

KeyName 1: This parameter, (note the numerically appended increment allowing set of matching pairs that are appended to the WHERE clause in the SQL statement

The system caters for a variable number of parameters to be passed in, making the extraction request as broad or granular as necessary. Once the request is received, the user is presented with a response screen which indicates that processing has commenced. Extraction, conversion and creation of the CSV or XML data can take some time, depending on the volume of data being extracted, and in both cases the extraction request is persisted (physical file created) to the local host's file system.

Figure 4.3 Data Extraction Request

A URL is presented to the user which links to the physical data file, and a warning, indicating that the file may not be immediately ready for extraction. This message is posted to ensure that if the file is large, the user cannot access the file until extraction is complete. Once complete, the user may download the file to their local machine. The download request is initiated by clicking the link, which causes the browser to fetch the file from the extraction server and request a file deposit location on the local host. Once the user has indicated a directory location for the file, transmission commences.

4.2.3 DATA INSERTION

The Framework enables data to be inserted into the local user's database by providing a monitor component which polls a nominated directory looking for incoming XML files. The user may either deposit the file into the directory being monitored for automatic insertion into the local database, or, if the file format selected is CSV, may nominate an alternative directory for the download. When the monitor sees a candidate file appear in the target directory, an insertion process is initiated. The insertion monitor component is configured into the Windows autostart process, and commences when the user's machine is powered on. The user must know the name and location of the directory to 'drop' the XML file into, but this is typically set up as a directory with a shortcut set on the user's desktop. The monitor continually polls a nominated file directory until a file is deposited into the directory with an XML file extension. When the insertion monitor process sees a valid file, it attempts to process the data using the local (host) Web Server to facilitate a stream connection to the XML file. Schema information within the XML file allows the process to build the in-memory Document Object Model for the associated table. The insertion monitor component of the Framework acquires the necessary configuration information from the user's local database. Information such as the physical polling directory, whether logging is set on or off, and if logging is set on, where the log data is to be deposited. Once the Insertion process has

commenced, an entry is logged which provides information on the status of the insertion process, time/date, table name, number of records in the transmission etc. Most importantly, a *TRANSACTION* is commenced to allow Commit or Rollback on the entire insertion process. On completion, the status is set to *COMPLETED*, or, if a failure occurs, the data is rolled back, and the log is updated to reflect the rollback status.

4.3 Issues for Discussion

This section reviews the research objectives and provides a context for discussion, by using the Framework architecture, relative to CBSE. The section also compares benefits accrued from using CBSE methods alongside the associated risks.

The stated research objectives are:

- Determine the effectiveness of using component based software engineering (CBSE) methods in the development of software applications, as well as post implementation maintenance.
- Highlight potential risks and identify areas of vulnerability when using CBSE methods.
- Investigate both the benefits as well the risks associated with integrating a new middleware technology such as XML across the range of COTS Web components.
- Review the Object Oriented principles used in the development to determine if this added value to CBSE method.

4.3.1 Component Based Development

A prerequisite to the adoption of CBSE is an understanding of what constitutes a component. Once the component is understood, a set of usage or deployment procedures can be developed to ensure that the benefits of that usage or deployment is maximised. Szyperski defines a software component as a functional module with contractually specified interfaces and explicit context dependencies, which can be deployed independently and is subject to composition by third parties.

The concept of contractually specified interfaces is mandatory for all Off The Shelf components. The interface facilitates the request and response communication between two components and must be explicitly and rigidly enforced. The contract

is binding between the requestor and the service component, with input parameters, processing constraints and any resulting data completely specified within the contract [VOAS98]. The Framework makes use of both Common Off The Shelf Software (COTS) and custom code. Figure 4.1 presents an architectural view of the Framework, which uses COTS components such as the Web Server, Servlet Engine, and ODBC Drivers. These components are operating in consultation with custom code components to complete the data extraction, physical persistence of the data files, and the re insertion of data back into the remote data base.

Explicit context dependencies help to specify the runtime environment of the component, this can only be determined by carrying out a thorough domain analysis. Whether the component is a COTS component or derived from a re-use repository, the context or scope within which the component will be used must be completely analysed and understood by the developer prior to deployment.

Included in the domain analysis should be some attempt to determine the possible future requirements or functional enhancements. Software engineers cannot predict the future but users often have a wish list of functional objectives. If the designer can gain some understanding of these future directions, stakeholders can be made aware of any disparity between the component's current, versus future functional dependencies. Presented as a business case, the stakeholders can then decide whether the short term benefits derived from use of the component is outweighed by future requirements which cannot be accommodated by the component in its current functional state. The issue here is that stakeholders are often sold on the cost benefits of using a COTS component, without being given an opportunity to amortise those costs over the entire life of the application.

The Framework components were derived from a mix of in-house and third party sources. Development made use of a functional prototype to maximise the information gathered for input to the design. This provided valuable data on the required interfaces and operational context dependencies needed to construct an integrated and interacting set of components. It can be suggested that the implementation was successful or at the very least effective, because the use of a prototype allowed the runtime context dependencies and interfaces to be more easily determined.

4.3.2 Domain Analysis

Capretz promotes an initial domain analysis be carried out when a component based development model is being used. Domain analysis yields a catalogue of potential reuse candidates, allows the developer to identify processing boundaries, interoperability constraints and areas where COTS components may be used. Our research confirms Capretz's opinion that the use of a prototype or functional model is a priority if useful domain analysis is to be achieved. Design improvements to the Framework derived from the socket based prototype cannot be overstated, providing a clearly defined scope of operation and highlighting technical and operational issues surrounding the transportation of data from one host to another. When an existing application is in place, the outcome of Object Oriented Analysis generally and Domain Analysis specifically is greatly improved. The application presents both top down and bottom up views for analysis [ABU-GHAZALEH99]. This is relevant because processing operations, data interaction and relationship issues can be observed from a number of levels. The outcome of this duality is that candidates for reuse can be identified at both the granular level of code modules, but also at a much higher level of operation. This higher level scoping is particularly important when extending the component based model to include COTS components and identifying larger more functional custom components.

The successful implementation of the Framework, and ease of integration of specifically written code with third party components confirms that the adoption and integration of third party technology components into the design has made the Framework more robust, more abstract and therefore, more extensible, for example:

- Using an http request (via a Servlet) to launch the necessary Extractor wrappers to manage the stream (managed by a java.net.URLConnection), means the process does not have to be manually initiated.
- Querying the database and preparing resultsets (using a java.jdbc.DriverManager class), simplifies the design and strengthens the robustness of the application.
- The externally managed Schema model provides processing information for both extraction and insertion processes on how the data is formatted and presented.
- Using XML means that the scope of the application is automatically broadened to include any third party process which accepts XML.

4.3.3 Benefits of Component Based Software Engineering

Implementation and Scalability

Intangible benefits are also accrued by using off the shelf components. Setting up and deploying additional field sites is now a very straightforward task for Pipeline support personnel. Standard PC system configuration will include most of the third party components, installation of the operating system (OS) will include the Web Server, SQL Server, and the Servlet engine. The browser-based system provides easy access to relevant data. I in normal use is coupled with a 'drill down' viewer, which allows the user to query and locate data using a visual location method, prior to downloading the data from the desired field location. Once the site is setup, scalability becomes an issue for the Web Server, and is abstracted from the application domain.

The benefit to the administrator is the ability to install and run the applications with minimal disruption to existing users. This is achieved because field site setup information is either provided using scripts, or manually entered via the SQL Server Enterprise Manager either locally or remotely.

4.3.4 Object Orient Software Engineering

Based on our experience with the Framework development, we stress that component based development methods must be embraced by all stakeholders and implemented in all phases of the development lifecycle. To this end there are a number of CBSE development methodologies which are be being promoted [GANESAN01]. All utilise an underlying Object Oriented methodology, specifically Jacobson's Object Oriented Software Engineering (OOSE) [JACOBSON97] method which intrinsically adds value to the process of analysis. Using the OOSE method, processing modules are categorised into three areas of responsibility; Interface, Control or Entity. These categories follow the three tiered architecture model in which Interface modules, or boundary objects provide an interface with a requesting object (Actor), such as facilitating the capture of information, or output of results. An entity object has a direct interface and mapping to the data layer. Control objects are then deemed to be 'what ever is left over', and are developed as necessary. Control objects can be simple wrappers to glue the other two layers together, or, can include complex processing instructions to validate, constrain, or transform data, to add value to the overall application process.

We found that this broad brush categorisation is extremely useful in the domain analysis phase of the CBSE method, and allows the developer to clearly separate the processing requirements, as it highlights areas where components can be interfaced for service and co-operation. Seminal work by Mehta et. al. [MEHTA00] is putting a special type of component forward, categorised as a Connector which addresses the issue of interfacing domain and generic non domain components using a connection specific object.

Our observations highlight the importance of an iterative approach to the analysis and design phases in any software engineering application, especially when component based methods are being employed. As Capretz points out, when the architecture is 'component-centric', designers need to ask "where are components that I can directly, or indirectly use or reuse, to solve this problem" [CAPRETZ01]. As components or potential components are identified, re-evaluation and subsequent analysis iterations are required, ie. domain analysis. Our experience confirms the recommendations made by Capretz, who maintains that a Re-Use Library should be actively referenced throughout the entire analysis, design and implementation phases of the project. The advantage is the conceptual continuity across all phases of the software development lifecycle. Continual re-evaluation does have the effect of increasing the length of the analysis and design phases, as stated previously. This issue needs to be addressed with the stakeholders prior to commencement of the project.

4.3.5 REUSE

The issue of re-use is now fundamental to all Object Oriented methodologies. Organisations like Microsoft (COM) and PowerSoft (PowerBuilder) have been promoting the use of foundation classes and a common architecture for more than a decade. When this methodology is adopted, re-use is promoted via inheritance, with specific process classes being inherited and interfaced from more generic foundation classes. More recently, a number of component based development methods are also being promoted which focus on the iterative review process to identify components for re-use much earlier in the development cycle. KobrA [ATKINSON01] and Catalysis [DSOUSA98] are both methods which use a UML notation method to identify components, but include a component identification (domain analysis) from the beginning of the analysis cycle. Both lean heavily to the Unified model, but mandate the identification of components much earlier, as

Atkinson says "the Rational Unified Process explicitly delegates components to the final implementation stages of development". We can confirm the benefits to the implemented Framework when re-use is factored into the total amount of code written for the Framework. The benefits are derived from use of the prototype, which allowed the insertion requirement to be modeled at a much more detailed level. Because the extraction process code had already been written for the prototype, it was much easier to design the objects necessary for the Framework extraction and insertion process. The implemented Framework uses the same core objects for both extraction and insertion. These objects are simply assembled differently when bundled for insertion or extraction.

Henderson-Sellers' fountain lifecycle model is another object-oriented approach to software development, using a highly iterative and re-use focused approach. The Fountain Software Development Lifecycle Model promotes reuse within the domain analysis/design phases. The model gets it's name from the analogy of a fountain, in which water rises up the middle and falls back, either to the re use 'pool' below or is re-entrained at an intermediate level. In addition, the model fully promotes the concept that analysis, through design to implementation is overlain with iterative cycles across two or more lifecycle phases.

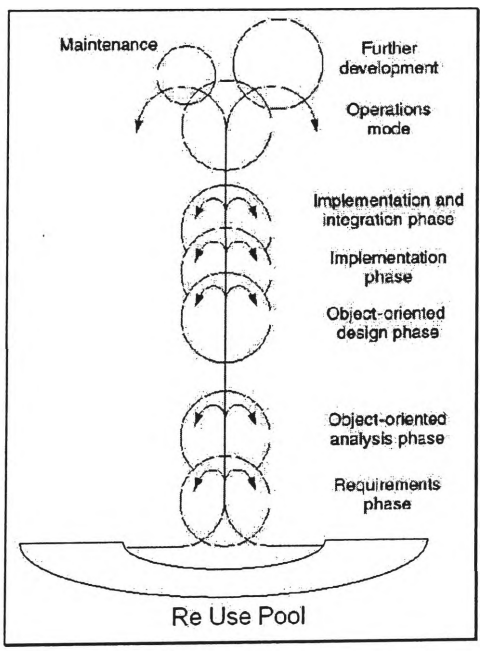

Figure 4.4 Fountain Lifecycle Development Model

Development of an object-oriented system is much more likely to lead us to focus on subsystems, which are made up of collections of classes which work closely

together. Adherence to the OOSE method facilitates the early identification of candidate classes or components.

4.3.6 Risk Management

We acknowledge there are significant risks associated with CBSE practices, however, there are a number of ways to minimise this risk, and at least manage the negative potential of components, should it be necessary. We review the risks associated with CBSE development methods which are listed below:

- The blackbox nature of the software when using COTS components
- Lack of software quality information
- Hidden Costs associated with post implementation maintenance
- Lack of a suitable 'bottom up' design information
- Lack of accompanying documentation/information regarding the software
- Potential for a longer and more costly development lifecycle

We acknowledge that the Framework architecture only makes use of standard COTS components, such as the Web Server, Servlet Engine and ODBC drivers. The Frameworks' use of in-house developed components being integrated with COTS components can be used as a basis for comparison against current CBSE literature.

With regard to COTS components being integrated into the application when they are derived from code libraries or have been sourced from specialised third party software houses:

- There may be problems associated with the functional use of black box software, ie. No source, or more importantly, no design criteris is available.
- The documentation may not fully, or explicitly explain all usage issues, or provide enough detail regarding the handling of conditions or events associated with the component.
- Post implementation modifications being made to custom software which is being integrated with other components may also be unsuccessful because of a lack of extensibility built into the components.

Brereton makes the point that when special purpose code is developed to facilitate integration or at least achieve interoperability with COTS components, an increase

in subsequent maintenance costs is likely because developers may have to modify or readapt wrappers or re-glue components as they evolve [BRERETON00]. However, when the components being integrated into the application are more autonomous and have been developed with a clearly defined application program interface (API), there is much less likelihood of such issues as incompatibility, unreliability, lack of scalability or extensibility.

The completed Framework allows the issue of hidden design criteria and functional unknowns, (blackbox) associated with nature of COTS software to be addressed. For example, ownership of the Servlet engine has changed a number of times, and it may be argued that the next vendor may not provide backward compatibility. However, there are a number of Servlet engines available on the market, all have been developed to accept input from a standard HTML page. As long as the interface between the components is well documented, with clearly defined usage rules, the worst that can happen is that the brand of Servlet engine may change. At the large and complex end of the COTS spectrum, the issue of hidden functionality of using black box components is being addressed from both top down, and bottom up. Large service and software development organisations such as IBM, Microsoft, SAP etc. provide excellent online documentation and support across the entire software development lifecycle. At the other end of the COTS cost spectrum, freeware technology such as Tomcat etc. also provide excellent online documentation, There are also specific component 'markets' which are well specified, with a finite number of vendors, for example, any design which promoted the in-house development of an application specific Web Server over the integration of a third party off the shelf Web Server, supporting numerous middleware gateways would likely be rejected.

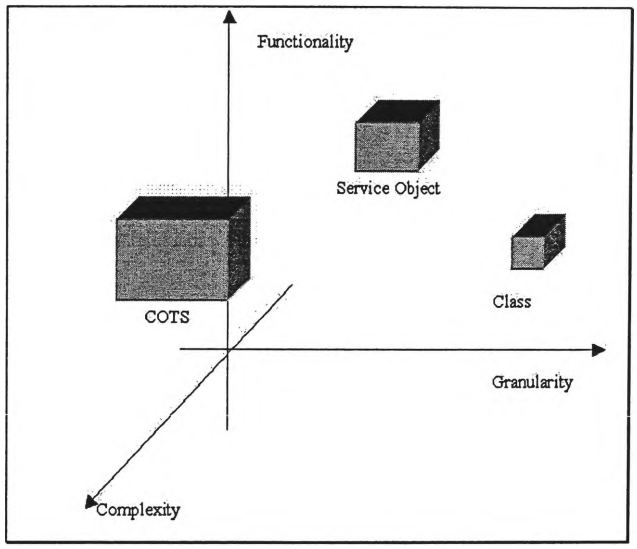

Figure 4.5 Domain Analysis

When integrating off the shelf components into an application, analysis models may need to include a weighting factor to signify the importance of access to the internals of third party components based on granularity, functionality and complexity (refer to figure 4.5). The diagram graphically indicates the position of COTS components relative to other components (service objects) and 'hand crafted' classes. While the COTS component offers a complete (complex) solution to a specific problem, there is a trade off. The designer does not have access to the granular detailed information needed to assess the component from a bottom up perspective for later re-use or maintenance. If the risk of not being able to determine the internal design criteria outweighs the advantage of using the product, based on the weighting, then alternative sources for the component may need to be found.

An example of weighting and determining design priorities based on that weighting is that there is no application level security in place within the Framework, given that there is no direct (external) access to the Source or Target Database. When a Domain Analysis phase has been incorporated into the development lifecycle model, the designer is able to use the analysis results to determine that custom built authentication and access are not a requirement. The Framework extraction module has write access to the directory containing the extracted files, but user access is read only to facilitate the data transport to the user's desktop, or to another database under program control. As long as analysis and requirements gathering has clearly identified the nature and scope of the application security needs, the correct weighting can be applied. In the context of the Framework, the main security issue

relates to the modification of incoming XML data (for malicious purposes) by a third party "man-in-the-middle" [GREENSTEIN01], who modifies the contents of the XML file. However, the designer has enough information to determine the priority and scope of the security requirements and can then determine whether off the shelf components, re-use repository or custom code is necessary to fulfill the requirement.

The potential for Hidden Costs relating to post implementation modifications [WANG01]cannot really be placed in context when using the Framework as a basis for argument. The software system designer or architect must fully investigate the source pool of components, and be diligent in researching the configuration, deployment, tuning, scalability etc. issues which may result in future changes. Even in a design in which current processing requirements are met, failure to fully investigate and comprehend context dependencies can mean that maintenance may be more complicated and difficult because complete access to the source code of the component is not always possible. The development cycle for the Framework used a series (3) of reviews, which took place after each build iteration. We confirm there is a significant case for the use of domain analysis techniques, and iterative reviews of objects and components with a view to re-use throughout all phases of the development project. Especially when the project is large and/or complex; as more components are identified along the design, re-evaluation of the complete set of existing components becomes necessary. Code re-use is one of the main principles of CBSE methods, and continuous review of the problem domain promotes the identification of re-use candidates, which has enormous benefit potential. Costs can be reduced both in the current development project, as well as later maintenance costs when change is required, or components can be used in other development projects. With regard to COTS components, lack of documentation and information available to potential users of the software components can be problematic. This issue is similar to the software components being assembled and deployed as a black-box. If the vendor does not provide the necessary quality of service, and this includes documentation, then other Vendors in the market must be approached. Another alternative may be developing the component in house. As long as the stakeholders are involved early enough, risk is minimised, or at least managed. The time spent in reviewing, or investigating alternative Vendor solutions, may well be a longer development period, however, these days, all systems must be developed with change being an accepted and

acknowledged part of future requirements. Development projects have less scope to project or quantify potential risk, when using COTS components (and this is a risk in itself).

COTS components are primarily factored into a cost/benefit analysis as an immediate, or short term benefit. This is because the cost of developing an in-house solution is much more expensive than acquiring an off the shelf solution, the short term benefit is a cost saving. Longer term benefits can only be generalised, ie. perceived benefits are based on organisational strategies which project future requirements, and how the COTS components would operate within those future requirements. Typically there is no source code available with a COTS component, so re-use of code at a later time is not generally an option, the major risk is that the organisational requirements will change, placing a requirement to meet that change on the software application which has adopted or integrated COTS components into the solution. In today's technology environment, software has to be extremely adaptive, and loosely coupled, there is considerable risk that COTS functionality may not be extended to cover the agile requirements of the 21st century organisation. One analogy is calling the purchase of a car, a risk because the purchaser will have to buy new tyres as part of future maintenance. This may be considered a hidden cost, so the purchaser must weigh up the value of buying better quality tyres now, or replacing a cheaper brand sooner. The Framework only makes use of the Web Server, and the Servlet engine as COTS components, these are not components which demonstrate any short term propensity to be overcome by functional change which cannot be accommodated by the current operational scope.

With regard to in-house developed components, such as the DOM, query handler physical persistance components etc. code re-use is probable, extending the functionality of existing code is also probable. This greatly reduces the cost of future maintenance, and lessens the risk of introducing errors when integrating completely new components into the production environment.

4.3.7 XML USAGE

Use of XML as a transport container, coupled with an XML Schema has benefited the Framework application in many ways. For example, the Schema allows for an extremely 'thin' tag wrapping, which reduces the application footprint overhead associated with the StoneBroom utility [STONEBROOM02], and provides a

common mapping for both the extraction and insertion processes. The XML document is quite human readable in its natural form, making the format useful for field site users who can easily recognise the weld specifics by viewing the file using Internet Explorer. This ease of access for the human user also demonstrates the intangible benefits associated with use of an exchange medium like XML. Microsoft is fully committed to the standard and has included an XML Parser within Internet Explorer (Version 6.0+), and this commitment is further evidence of the willingness of third party software and technology vendors to facilitate component based software engineering. Within the Pipeline system domain, field sites produce considerable amounts of data relating to the pipe weld process, site logistics and geographical location. Once this data is captured, it needs to be analysed, either locally or at a central repository, collated and then moved to a long term repository. The Schema allows the data to be efficiently packaged (as XML) and transported over the Internet via Browser download.

Bourret identifies a number of potentially negative issues associated with XML which are well documented. In the case of the Framework, the only genuinely negative issue is the amount of data being extracted by the Framework means the resultant XML file is also large in terms of data content. As outlined earlier, there are a number of issues which make the Schema method a good fit for the project under discussion. Most importantly is the volume of data being generated from a weld, which causes an overhead when the data is extracted from the database, shipped over the Web, then unpacked at the target end. Each of these steps takes time and increases the total wait time for an interactive user. It is important to note that use of an XML Schema provides a lot of scope for the developer to model the Schema so as to maximise the efficiency of the syntax to best suit the application under development. Appendix D provides more information on specific Schemas put forward for use by interested industry groups. This concept can be extended to allow an application specific Schema (rather than industry specific), and this is the case with the Framework Schema. There is no need to have the Schema registered if it is being used within a single domain, ie. as long as both sender and receiver have access to the Schema DOM. The Framework Schema has the following design criteria:

- There is no necessity to provide for human access to the XML, although the Schema is 'well formed'.

- The Schema must provide information to allow the extracted data to be persisted in Comma Separated Variable format (CSV).
- Because of the data volume, the Schema must facilitate the most efficient form of document storage.
- The Pipeline Project is supported by the Framework, which means that Schemas' will be developed for additional tables being transported by the Framework. In order to facilitate this process, the Schema model must provide a simple format which can be followed by non technical users who wish to use the Framework facility. The entire basis for the Framework is extensibility for the non technical user, therefore the Schema must be simple enough for a non technical (computer wise) user to generate, and deploy.
- The Schema can be used to build the XML document as well as read and 'unpack' the XML document.

4.3.8 Operational Issues

Operationally, the Framework has been rigorously tested and successfully deals with extremely large amounts of data. Use of the intrinsic download feature of the Web browser enables data to be transported from one host to another in an efficient and secure manner. Extensibility has also been demonstrated with other components of the Framework making use of the Framework to transport data within the system boundary.

Considerable load and volume testing of the Framework has been completed on the assumption that the typical user will be transporting high volumes of data over the Internet. In addition, one of the main objectives of testing is to determine the worst possible extraction time based on the ambient load, which will provide a guide to the most appropriate minimum configuration for the field data Server. In order to determine the configuration details, a number of differently rated Servers have had extraction and insertion requests placed on them, alongside a range of concurrent requests.

There are a number of interesting issues which relate to the overall efficiency of the extraction and insertion processes. Insertion of data into a Framework associated data base is four (4) times quicker than the extraction process, yet both processing

modules contain basically the same code. Analysis and testing of the modules suggests that the main issue which contributes to this disparity is the persistence or physical writing of the extracted output to a flat ASCII file format. There is also some overhead associated with ODBC, ie. general efficiency is affected because of the conversion of the all data types to string data for transport.

In extraction mode, the Framework uses an abstract Java base class 'RandomAccessFile' to perform the physical output of the data into XML or CSV format. Insertion of data back into the database does not use this class, and instead reads the XML file using a 'BufferedReader' class, so the data is passed in as a stream, and is clearly more efficient than the IO mapping carried out by the RandomAccessFile object. The CBSE design method acknowledges change and accommodates for change by allowing new components or improved components to be integrated into the application. This is clearly an issue for design review, as the physical I/O of the XML file should not present an efficiency bottleneck. Focusing on the extraction process in isolation; clearly, the processing speed of the machine carrying out the extraction also becomes an issue. On a PentiumIII 500 MHZ machine, the system processes data in the order of 1.1 megabytes of data per minute, which equates to approximately 3500 rows of welding data. Running the same extraction process on a PentiumIV 900MHZ, processing speed is improved by roughly 3 times. While the data extraction speeds must be considered slow on a Pentium III machine, the typical field machines will be Gigahertz rated, which will provide extraction speeds of the order of 3.5 megabytes (11,000 rows) per minute. The Framework will easily accommodate the integration of a replacement persistence module, as the persistStream object can be replaced by a more specific component to maximise the speed of physical I/O.

Kotonya cites the lack of software quality information and the longer term issues of product direction and development priorities. Moreover, the importance of distributed licensing may not be given the necessary priority when off the shelf components are integrated into an application. Licensing and related documentation of the third party components making up the Framework, or any application is an issue. The implications of ignoring or not giving the appropriate priority to investigation of the availability or lack of availability of documentation, licensing restrictions, support, education or training cannot be over stated. Developers are

used to taking responsibility for quality standards, specifications and operational documentation associated with components they develop themselves. The issue does point to the professionalism and experience level of the project stakeholders, but may be overlooked, say when the project team is incorporating COTS components for the first time. These issues confirm the importance of using a CBSE methodology when initiating a project which makes use of components and in particular off the shelf components.

4.4 Review of Design Issues

The Framework provides a number of extremely useful benefits, not just for the Pipeline project, but any application which must move data from one host to another in an efficient manner:

- The thin application footprint on both source and destination machine mean that memory usage and processing overheads have been kept to a minimum. This is possible because of the use of third party, component based technology running on each host machine, in effect simply exercising applications which are already resident in memory.
- Java's strengths in component development have provided enormous code saving benefits. Extensibility, facilitated by abstraction and the use of interfaces allows the Servlet model to pass data to custom written multi-threaded code modules, which again maximises the machine efficiency. The benefit to the user is that multiple extraction requests can be processed concurrently. The Java Servlet model manages the threads, which means the only programming requirement is to synchronize the appropriate code modules, but no thread specific code needs to be developed.
- The Framework presents as a mix of distributed components which provide a means of selecting, then extracting data for local use. The application demonstrates the concept of Web Services[EVJEN02], in which users may access remote services using loosely coupled distributed modules to reduce the duplication of functionality at the local site.

A separate, but nonetheless interesting design issue did not arise until the system was tested in a live environment. Problems associated with Internet latency caused the

process to be split into two separate components. Initially, the design method used to satisfy the insertion of the data into the user's local database used the extraction thread to contact a Servlet running on the client host. The XML file was passed to the client host as a 'background' insertion process which eliminated the necessity of the user manually downloading the XML file. This method worked well under laboratory conditions, where a Subnet serviced the machines in the System. However, when tested in a genuine Internet environment containing multiple switches, Proxy Servers and bandwidth latency, the Servlet model was not able to deal with that latency in a controlled manner. In many cases, the two hosts involved in the URLConnection process would time out, and the connection would be reset by the peer service. Also, if the file being transmitted was of a size greater than .5 of a megabyte, there is a considerable time delay between initiating the request, and completing it.

In summary, we found that risks associated with CBSE can be successfully managed as long as the analysis and design methodologies accommodate and accept the principles of Component Based Software Engineering. Acceptance of these principles mean that a rigorous Domain Analysis is carried out in order to:

- Identify potential areas where Off the Shelf Components (COTS) may be used within the application domain.
- Identify potential areas where generic service objects can be acquired from third parties, or custom built
- Identify potential areas where custom objects can be made more extensible by increasing or making changes to the existing design

Iterative review of the component pool must take place on a regular basis throughout the entire development cycle. Performing these tasks accurately and with enough granularity means that the amount of time required to carry out the analysis and design will be extended and must be commensurate with the complexity of the application, and the stakeholders desire to make use of CBSE methods.

Post implementation reviews of the Framework have demonstrated that change and enhancement can be accommodated with minimal disruption to existing code if a layered approach is adopted in the design.

We also promote the use of functional prototypes as a mandatory part of domain analysis, when the scope of the application under development is large, complex, or the stakeholders wish to investigate the use of new or untried technology.

Another issue which re-inforced benefits of CBSE and provided further confirmation that in order for Component Based Development (CBD) methods to deliver successful applications, Object Oriented analysis and design development principles must underpin the software engineering model adopted for the project, at all levels.

4.5 CONCLUSIONS

We conclude the thesis by reviewing the research objectives, and elaborate on the outcome of those objectives.

The Framework design, construction and implementation has been reviewed and accepted by the WTIA. Using CBSE methods in the development of the Framework to support data transport over the Internet has produced a robust and extensible data management application which provides a rich set of presentation choices for the end user. It is clear that the adoption of CBSE methods must be embraced by all stakeholders and implemented in all phases of the development lifecycle in order to be successful. The developer must fully recognise the potential for risk, and incorporate strategies into the project to minimise the risk and maximise the benefits of the use of components.

One of the primary research objectives was to review the Object Oriented principles used in the development to determine if this added value to CBSE method in terms of assembly and integration of the components. We found that constructing the Framework as a series of integrated components which communicate via clearly defined interfaces allowed those components to be uncoupled, and run as separate processes. Bourret makes the point that adopting CBSE design principles allows components to be split into one or more interface layers which reduces the impact of change.

Another objective of the research was to gauge the impact of CBSE on post implementation maintenance to determine whether the adoption of CBSE methods make change easier and less costly, or otherwise. This relates directly to the issues cited by Wang regarding the potential hidden cost of post implementation changes

and maintenance. The impact of post implementation change can be demonstrated by how easily the Framework can be remodeled to accommodate the problem of the Servlet model being unable to keep the user informed of the progress of the transmission. We observe a minimal cost, when the application modules have been designed as components, communicating via clearly defined interfaces. The solution to the problem was to have the user manually download the file from the Source Server. Since the XML or CSV file already exists on the source host, a link to the file is presented within the target user's browser, when the user selects the link, a download request is presented. Instead of the insertion component receiving the file from the remote source host, the user has to manually download the file, so the insertion component receives the file from the localhost. It is interesting to note that the target host sees the referenced Schema (XSD) file located on the Source Host, and processes this file using the URLConnection Interface, to great effect. This is because of the relatively minute size of the Schema XSD, associated connection problems only occur when the XML file is larger than half a megabyte.

In terms of potential risks, there are far more positive benefits for the developer who chooses to incorporate third party components and technologies into the application design, and takes the time to continually review those components which have been already identified across the entire range of granularity, functionality and complexity. The use of 'best of breed' components is now well recognised in the commercial and business environments, and a study of the domain should always be incorporated into the early design phase to ensure that a component wheel need not be re-invented.

With regard to the risks associated with integrating a new middleware technology such as XML across the range of COTS Web components, the COTS components selected for the Framework were too specific to provide enough scope to make a determination. The Web Server component has been developed to accept and accommodate XML, so a risk comparison was not demonstrable. We observe that XML is a language which is being widely accepted, and the speed with which associated technologies such as Web Services are being developed confirms that little risk is being identified.

We were also unable to explicitly determine if the adoption of Object Oriented development methods benefited the use of CBSE. OO promotes the re-use of code, which is a tenant of CBSE. What is clear from the Framework development is that

explicitly identified Interfaces facilitate the loose coupling of component and object interaction. We observe that when change is required at the component level, clearly defined interface boundaries, and a layered approach to design make the integration of components, whether in-house or third party, a much less complex task. More importantly, less modification to existing code is required to integrate new components, or modify the logic of existing components, when an OO development method is used.

4.6 FUTURE WORK

One of the more subtle objectives of the research was to maximise the use of off the shelf components in the processing functionality of the Framework. Microsoft have now deployed SQL Server 2000, which has built in support for XML, ie. SQL requests can be serviced with the output produced in XML format. There is also the issue of improving the efficiency of there persistence module, responsible for physical outputting of the XML data.

Reviewing the design, and carrying out a post implementation domain analysis to determine the functional 'fit' of replacement components, would provide more useful information relating to component based software engineering practices. The closer we move to the use of 100% off the shelf solutions, the more generic the application under review becomes, and ultimately this is the only tangible issue of using component based software engineering methodologies.

Connectivity between components has traditionally been dealt with using wrappers to facilitate data interchange. Current work places the component interactions and points of data interchange between connectors on a par with component functionality.

New 'reflective' middleware technology is also putting pressure on component integrators to provide methods of passing configuration, event management, data stream, broker information etc. The concept of a connector component, whose role is to manage the interface in a pro-active manner has a very high research focus at present. The Framework provides a useable test bed and research base for future investigation and comparison of the design criteria and resulting Interfaces used between components in the Framework, and the properties of interface specific 'Connectors' [MEHTA00].

References

[ABITEBOUL00] Abiteboul, Buneman and Suciu, *Data on the Web*, Morgan Kaufmann, 2000

[ABU-GHAZALEH99] N.B. Abu-Ghazaleh, M. Rangarajn, D.D. Dieckman; *Engineering of Component Based Systems*; Proceedings of the ECBS'99 IEEE Conference and Workshop, Nashville TN, USA, 1999

[ATKINSON00] C. Atkinson, J.Bayer, O. Laitenberger, J. Zettel, *Component Based Software Engineering: The KobrA Approach*, Proceedings of the 22nd International Conference Of Software Engineering, 2000

[BERTINO01] Bertino and Catania, *Integrating XML and Databases*, IEEE Internet Computing, Vol 5, No. 4, July - Aug 2001 pp. 84-88

[BOOCH98] Grady Booch; Object Oriented Analysis and Design(2nd Edition); Addison Wesley (1998)

[BOUMPHREY98] Boumphrey et al., *XML Applications*, Wrox Press 1998

[BOURRET00] R. Bourret et.al, *A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Tables*; Advanced Issues of eCommerce and Web-based Information systems, WECWIS (2000), 2nd International Workshop, Milpitas, CA, USA

[BRERETON00] Brereton et.al; *Component Based Systems, a classification of issues*; IEEE Computer Vol 33, No. 11; 2000 pp 56

[CAMPBELL99] A. Campbell , G. Coulson, M.E. Kounarvis; *Distributed Computing; Managing Complexity Middleware Explained*; IT Professional, Vol 1, Issue 5 Sept. 1999, pp. 22-28

[CAPRETZ01] L.F. Capretz; M.A.M. Capretz; Li Dahai, *Component-Based Software Development*, IECON '01. The 27th Annual Conference of the IEEE Industrial Electronics Society, 2001 pp. 1834 1837

[CHEES01] John Cheesman and John Daniels, *UML Components: A simple Process for Specifying Component Based Software*; Addison Wesley; 2001

[DSOUSA98] D. D'Sousa, A.Wills, *Catalysis: Objects, Frameworks and Components in UML*, Addison Wesley, 1998

[EVJEN02] W. Evjen, XML Web Services for ASP.NET, Wiley, 2002

[FLURRY01] G. Flurry, W. Vicknar; *IBM Systems Journal*, Vol 10, Number 1 2001, pp. 22

[GANESAN01] R. Ganesan; S. Sengupta, *O2BC: A Technique For The Design Of Component-Based Applications*, 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems, 2001, pp46

[GREENSTEIN01] M. Greenstein, M. Vasarhelyi, *Electronic Commerce; Security, Risk Management and Control, 2nd Edition*, McGraw Hill Irwin, 2002, pp317.

[HAEFEL00] Richard Monson-Haefel, *Enterprise Java Beans*, O'Reilly, 2000

[JACOBSON97] I. Jacobson, M. Christerson, P. Jonsson, *Object Oriented Software Engineering, a Use Case Driven Approach, 4th Edition*, Addison Wesley, 1997

[JAVADOC01] Java Doc, *Servlet Tutorial*, http://www.java.sun.com, 2001

[KOTONYA01] G. Kotonya, A. Rashid, *A Strategy For Managing Risk In Component-Based Software Development*, Proceedings of the 27th Euromicro Conference, 2001. pp 12 - 14

[MEHTA00] N. Mehta, N. Medvidovic, S. Phadke, *Towards a taxonomy of Software Connectors*, 22nd International Conference on Software Engineering, Limerick Ireland, 2000. pp. 178-187

[MOHR02] Stephen Mohr et. al, *Web Service Faceplates*, Wrox Press, 2002

[ORFALI96] Orfalli, Harkey and Edwards, *The Essential Distributed Objects Survival Guide*, John Wiley & Sons, 1996

[SAYER92] Andrew Sayer; Method in Social Science, a Realist Approach (2nd Edition); Routledge (1992)

[SOO01] Joo Kyung-Soo, *Design of Middleware components for the connection between XML and RDB*, Industrial Electronics, Vol. 3 June 2001, Proceedings, ISIE 2001, IEEE International Symposium, Pusan, South Korea pp. 1753-1756

[STONE02] Stonebroom Software; *ASP2XML*; www.stonebroom.com/asp2xml.htm (2002)

[SZYPERSKI97] C Szyperski, *Component Software: Beyond Object Oriented Programming*, ACM Press and Addison Wesley Publications, 1997, New York.

[VOAS98] J Voas; *Maintaining Component Based Systems*; IEEE; July/August, Vol 15, No.4; 1998

[WANG01] Guijun Wang, G. Cone, *A method to reduce risks in building distributed enterprise systems*,Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference, 2001.

[WHYTE01] W.S. Whyte; Enabling eBusiness, Integrating Technologies, Architectures and Applications, John Wiley and Sons, New York, 2001

[WILLIAMS00] Kevin Williams, *Professional XML Databases*, Wrox Press, 2000

APPENDIX A

A.1 SOURCE CODE

```
/**********************************************************************************************************************
 *
 *          class ANSIFileStream - Class provides a random access file stream to an ASCII/ANSI code set file.
 *          - Caters for conversion between UNICODE characters and ASCII/ANSI characters.
 *          - Extends the functionality of class java.io.RandomAccessFile
 *          - Provides Filtering for 'usable' characters, which include :
 *          Alphabetic characters, numeric characters,
 *          Punctuation marks, operator characters and special symbols
 *          ANSI Codes 120 to D70 -  which may make up words
 *
*
 *          - provides functionality to read/write and search for words from the given file
 *          A word is defined to be composed of the above listed characters.
 *          Any other character will be ignored and treated as a word separator
 *                  - this includes all whitespace characters
 *
*
 * Written by Adrian Collins
 *
 * RELEASE - October 2000
 *
 */
import java.io.*;

public class ANSIFileStream extends RandomAccessFile
{
        static final byte kFIRST_VALID_CHAR = (byte) '!';
        static final byte kLAST_VALID_CHAR = (byte) '}';
    protected String m_sFileName = null;
    protected String m_sReName = null;
        private File        m_File      = null;

  public void Delete() throws IOException
        {
          m_File.delete();
        }

/**
 *          Constructor - Creates a random access stream to a file.
 *                                  The Stream is a read/write stream to the associated file,
 *                                  Does not truncate the file.
 *
 *          Arguements  - fname - filename of the desired file
 *
 *          Throws              - IOException
 *
 *          Returns             - n/a
 */
  public ANSIFileStream(String fname) throws IOException
        {
        super(fname,"rw");
                m_File = new File(fname);
                m_sFileName = fname;
        }
```

```
/**
 *          Method              - Tests whether the opened file is a vlid open file
 *
 *          Arguements          - none
 *
 *  Throws                      - IOException
 *
 *          Returns             - true if file is valid, false otherwise
 */
  public boolean isValid() throws IOException
          {
                  boolean valid = getFD().valid();
                  return valid;
          }

/**
 *          method passes in the actual name the file will be named to when the user
 *     will be given control of the file
 *
 *          Arguments           - fname - filename of desired file
 *
 *          Throws              - IOException
 *
 *          Returns             - n/a
 */
          public void passInFileName(String as_rename)
          {
          m_sReName = as_rename;
          }
/**
 *          Method              - On completion of the data load, rename the file to allow
 *                  the User to select it for download
 *
 *          Arguments           - none
 *
 *          Throws              - none
 *
 *          Returns             - bool
 */
     protected boolean renameFile()
     {

        File nfile = new File(m_sReName);

        if ((m_File.exists()) && (m_File.renameTo(nfile)))
        {
           return true;
        }
             else
             {
                return false;
        }
           }
/**
 *          Method              - Appends a given line of text to end of file
 *
 *          Arguements          - line - The line of text to be appended
 *
 *          Throws              - IOException
 *
 *          Returns             - void
 */
  public void AppendLine(String line) throws IOException
          {
                  seek(length());
                  writeBytes(line);
                  writeByte('\n');
          }
```

```
/**
 *      Method          - Writes a line of text
 *
 *      Arguements      - line - The line of text to be written
 *
 *      Throws          - IOException
 *
 *      Returns         - void
 */
        public void WriteLine(String line) throws IOException
        {
                writeBytes(line);
                writeByte('\n');
        }

/**
 *      Method          - Writes a word without endline charachter
 *
 *      Arguements      - word - The string of text to be written
 *
 *      Throws          - IOException
 *
 *      Returns         - void
 */
        public void WriteWord(String word) throws IOException
        {
                writeBytes(word);
        }

/**
 *      Method          - Reads Line of text from the file
 *
 *      Arguements      - none
 *
 *      Throws          - IOException
 *
 *      Returns         - line - the actual read line from file, null if at EOF
 */
        public String ReadLine() throws IOException
        {
                String line = null;

                try
                {
                        line = readLine();
                }
                catch(EOFException eof)
                {
                        return line;
                }
                return line;
        }

/**
 *      Method          - Searches the file for a specific word, and repositions the file
 *                              pointer at the end of the found word, if the word was not found
 *                              the file pointer is placed at EOF.
 *                              The word must not contain any whitespace characters
 *
 *      Arguements      - word - the word to search for
 *
 *      Throws          - IOException
 *
 *      Returns - true if the requested string was found, otherwise false
 */
        public boolean SearchWord(String word) throws IOException
        {
                String nextWord = null;
                seek(0);

                try
                {
                        for(;;)
                        {
                                nextWord = ReadWord();
                                if(nextWord.equals(word))
```

```
                                {
                                        return true;
                                }
                                if(getFilePointer()==length()-1)
                                        return false;

                        }
                }
                catch(EOFException eof)
                {
                        return false;
                }
        }

/**
 *      Method          - Reads and returns the nth word from the file,
 *                              count starts from the beggining of file.
 *
 *      Arguments       - n - the nth word
 *
 *      Throws          - IOException
 *
 *      Returns         - nextWord - the actual nth word, null if eof file was reached
 */
        public String ReadWord(int n) throws IOException
        {
                String nextWord = null;

                try
                {
                        seek(0);
                        for(int i=0;i<n;i++)
                        {
                                nextWord = ReadWord();
                        }
                        return nextWord;
                }
                catch(EOFException eof)
                {
                        return null;
                }
        }

/**
 *      Method          - Reads the next whitespace separated word from file
 *                              A word is defined to consist of ANSI characters 120 to D70
 *                              any other character will be treated as word separator.
 *
 *      Arguments       - none
 *
 *      Throws          - IOException
 *
 *      Returns         - word - the next word, null if eof file was reached
 */
        public String ReadWord() throws IOException
        {
                String word = null;
                byte ch;
                long len = length();
                long pos = getFilePointer();

                try
                {
                // skipping all non-word characters
                        pos = SkipNonWord();

                // checking to see if at EOF
                        if(pos ==len)
                        {
                                return word;
                        }

                // re-positioning cursor
                        seek(pos);
                        ch = readByte();
                        pos++;
```

```
                    if(pos ==len)
                    {
                            return word;
                    }

                // checking to see if the charachter is valid
                // Word Charachter - any alpha-numetic charachter including
                // Symbols, and punctuation marks
                    if(ch>=kFIRST_VALID_CHAR && ch<=kLAST_VALID_CHAR)
                    {
                            word = String.valueOf((char)ch);
                    }
                    else
                            pos = len;

                    while(pos != len )
                    {
                            ch = readByte();
                            pos++;

                    // preceeding characters can be numbers or underscores
                            if(ch>=kFIRST_VALID_CHAR && ch<=kLAST_VALID_CHAR)
                            {
                                    word += String.valueOf((char)ch);
                            }
                            else
                                    break;
                    }
            }
            catch(EOFException eof)
            {
                    return word;
            }

            return word;
    }

/**
 *      Method          - Skips n lines of input from file
 *
 *      Arguements      - n - the number of lines to be skipped
 *
 *      Throws          - IOException
 *
 *      Returns         - void
 */
    public void SkipLines(int n) throws IOException
    {
            try
            {
                    for(int i=0;i<n;i++)
                    {
                            ReadLine();
                    }

            }
            catch(EOFException eof)
            {}
    }

    public void Close() throws IOException
    {
            close();
    }

/**
 *      Method          - Skips all non-word characters until the first
 *                              Valid word character is encountered
 *
 *
 *      Arguements      - none
 *
 *      Throws          - IOException
 *
 *      Returns         - pos - The new offset within the file
 */
```

```
        private long SkipNonWord() throws IOException
        {
                byte ch;
                long len = length();
                long pos = getFilePointer();

                try
                {
                        for(;pos<=len;pos++)
                        {
                                ch = readByte();

                        // breaking as soon as we encounter the
                        // first non - word character
                                if(ch>=kFIRST_VALID_CHAR && ch<=kLAST_VALID_CHAR)
                                {
                                        return pos;
                                }
                        }

                }
                catch(EOFException eof)
                {
                        return pos;
                }
                return pos;
        }

        public static String GetUniqueSuffix()
        {
                String strdate = null;

                java.util.Calendar rn = java.util.Calendar.getInstance();
                java.util.TimeZone tz = java.util.TimeZone.getTimeZone("GMT");
                rn.setTimeZone(tz);
                java.util.Date currentDate = rn.getTime();
                long t = currentDate.getTime();
                //Uncomment this line if default locale set to GMT
                //long t = currentDate.getTime()+(9*60*60*1000);

                java.sql.Date sqldt = new java.sql.Date(t);
                java.sql.Time sqltm = new java.sql.Time(t);

                String sqld = sqldt.toString();
                java.util.StringTokenizer st = new java.util.StringTokenizer(sqld,"-");
                String y = st.nextToken();
                String m = st.nextToken();
                String d = st.nextToken();

    String sqlt = sqltm.toString();
    java.util.StringTokenizer st2 = new java.util.StringTokenizer(sqlt,":");
    String hr = st2.nextToken();
                String mn = st2.nextToken();
                String sc = st2.nextToken();

    strdate = "OLD_" + d + m + hr + mn + sc;

                return strdate;
        }
}
//EOF ANSIFileStream
```

```
/**********************************************************************************************************************
 *
 * class DataInsertor - Class used as a type-safe container for the data
 *                      contained in a single object. When used on the client side the
 *                      series of get methods are used to unpackage the data and
 *                      when sed on the server side the series of put methods are used
 *                      to package the data for transport onto the stream
 *
 * Written by Adrian Collins
 *
 * RELEASE - Sept 25th 2001
 *
 */
import java.util.*;
import java.io.*;
import ExceptionLog;
import XMLManager;
import DBConnection;
import RunParms;

public class DataInsertor
{
    private  ExceptionLog     m_uLog        = null;
    private String m_FileName, m_URL = null;
    private XMLManager  m_schema = null;
    private RunParms  m_runParms;
    private DBConnection  m_DBConnection = null;
    private String [] m_values = null;
    private String m_firstPartOfQuery  = null;
    private String m_lastPartOfQuery  = null;
    private int gi_cols;
    private int gi_rowCount;
    private final static String kTHIS       = "DataInsertor";

/**
 *          Constructor - Constructs an empty row
 *
 *
 *          Throws              - IllegalArguementException if String does not represent
 *                                      valid attribute
 *
 *          Returns             - n/a
 */
public DataInsertor(ExceptionLog a_uLog, RunParms  a_runParms) throws IllegalArgumentException
          {
         m_uLog = a_uLog;
         m_runParms = a_runParms;
         gi_cols = 0;
         gi_rowCount = 0;
         buildDBConnection();
         try
         {

           // this class provides the url of the schema to the XML Manager class
           m_uLog.LogMessage(kTHIS, "Posting the schema to XMLManager");
           m_schema = new XMLManager (m_uLog, m_runParms);
           m_DBConnection.setOffAutoCommit();
         }
         catch(Exception e)
         {
             m_uLog.LogException(kTHIS,"DataInsertor issues...");
             return;
         }
          }
```

```
/**
*         initialise for a new column to be prepared and sent
*
*
*
*         Returns              - n/a
*/
          public void Initialise()
          {
         try
         {
            // open the xml file and get hold of the xsd url
            // we need this to build the schema tree
            logInsertCommence(m_schema.getlogCommence());
            m_schema.buildXMLReader();
            m_schema.getSchema();

         }
         catch (Exception e)
         {
            m_uLog.LogException(e.toString(),"Data Base Meta Data could not be located...");
            System.exit(1);
         }

    }

/**
*         clean up
*
*
*
*         Returns              - n/a
*/
          public void Cleanup()
          {
         try
         {
            m_schema.Cleanup();
            m_DBConnection.setCommit();
            m_DBConnection.Close();
         }
         catch (Exception e)
         {
            m_uLog.LogException(e.toString(),"Data Base connection closed...");
            return;
         }

    }
/**
*         Build a database cpnnection
*
*
*
*         Returns              - n/a
*/
public void buildDBConnection()
    {
        String s_dsnName = null;
        String s_userID = null;
        String s_passWord = null;
        try
        {
           s_dsnName = m_runParms.getParameterValue("dsn");
           s_userID = m_runParms.getParameterValue("userid");
           s_passWord = m_runParms.getParameterValue("password");

           m_DBConnection  = new DBConnection(s_dsnName, s_userID, s_passWord);
        }
        catch (Exception e)
        {
           m_uLog.LogException(e.toString(),"Data Base connection could not be successfully completed...");
           System.exit(1);
        }

    }
```

```
/**
 *          output the data in nominated format
 *
 *
 *
 *          Returns              - n/a
 */
public void streamInput()
        {
       BufferedReader m_in = null;
       m_in = m_schema.getXMLStream();
       // data in the schema (xsd) tells us
       // how many columns in a row of this table
       gi_cols = m_schema.getColumnCount();

       if (gi_cols == 0)
       {
          m_uLog.LogException(kTHIS, " nothing in the xml file" );
          System.exit(1);
       }
       m_values = new String [gi_cols];
       String ls_Line = null;
       prepInsert();
        try
        {
            while (true)
            {
               ls_Line = m_in.readLine();
               if (getColumnValues(ls_Line))
               {
                  // **** this is where we increment the row count ****//
               }
               else
               {
                  m_uLog.LogMessage(kTHIS, "Completed reading XML file");
                  return;
               }
            }
        }
        catch(Exception e)
            {
              m_uLog.LogException(e.toString()," could not read xml file" );
        }
    }
/**
 *          output the data in nominated format
 *
 *
 *
 *          Returns              - n/a
 */
          public void prepInsert()
          {
       String ls_tableName = m_schema.getSchemaName();
            String [] ls_columnNames = m_schema.getColumnNames();
       m_firstPartOfQuery = "INSERT INTO " + ls_tableName + "(";

            // adding the variable names to the insert statement
       for(int i=0;i<gi_cols;i++)
            {
                  m_firstPartOfQuery +=ls_columnNames[i];

          if (i < (gi_cols - 1)){m_firstPartOfQuery +=",";}

            }
          // query1 += ") VALUES('"+weldid+"',";
       m_firstPartOfQuery += ") VALUES(";
    }
/**
 *          output the data in nominated format
 *
 *
 *
 *          Returns              - n/a
 */
public boolean getColumnValues(String a_line)
```

```
        {
            String [] ls_columnCodes = m_schema.getColumnCodes();
        int li_startPointer = 0;
        int li_endPointer = 0;

        // loop thorugh each row and strip out the xml codes, leaving
        // an array of column values in ls_values

        // return if not a data row, ie. <row>... </row>
        li_startPointer = a_line.indexOf("<row>");
        li_endPointer = a_line.indexOf("</row>");
        if ((li_startPointer == 0) && (li_endPointer > 0))
        {
           for (int i = 0; i<gi_cols;i++)
           {
              li_startPointer = a_line.indexOf("<" + ls_columnCodes[i]+ ">");
              li_startPointer = li_startPointer + ls_columnCodes[i].length() + 2;

              li_endPointer = a_line.indexOf("</" + ls_columnCodes[i] + ">");

              try
              {
                 m_values[i] = a_line.substring(li_startPointer, li_endPointer);
              }
              catch(Exception e)
              {
                 m_uLog.LogException(e.toString(),"xsd column count does not match actual, abort");
                 System.exit(1);
              }
           }
           DoInsert();
           return true;
        }
        return false;
     }
/**
 *        output the data in nominated format
 *
 *
 *
 *        Returns              - n/a
 */
public void DoInsert()
        {
        // we have the query statement ready to go, except for the values
        // themselves; which by now will be available in the m_values array

        // first get the data types of each column
        String [] ls_columnTypes = m_schema.getDataTypes();
        String ls_value = null;
        m_lastPartOfQuery = m_firstPartOfQuery;
        for(int i=0;i<gi_cols;i++)
             {
                    ls_value = m_values[i];
          if ((ls_value.equals("null"))||(ls_value.equals("")))
          {
             m_lastPartOfQuery += "null";
          }
          else
          {
             if ((ls_columnTypes[i].equals("Integer"))||(ls_columnTypes[i].equals("Float")))
             {
                m_lastPartOfQuery += m_values[i];
             }
             else
             {
                m_lastPartOfQuery += "'"+m_values[i]+"'";
             }
          }


          if (i < (gi_cols - 1)){m_lastPartOfQuery +=",";}
        }

        m_lastPartOfQuery += ");";
        try
```

```
        {
            m_DBConnection.runSQLUpdate(m_lastPartOfQuery);
        }
        catch(Exception e)
        {
            m_uLog.LogMessage(kTHIS, "Could not insert data, String = " + m_lastPartOfQuery);
            m_uLog.LogException(kTHIS, e.toString());
            // most probably a duplicate key
            return;
        }
    }
/**
 *          Ensure the table exists, otherwise create it
 *
 *
 *
 *          Returns             - n/a
 */
public void BuildTable(String a_query)
        {
        // create table
        try
        {
            m_DBConnection.runSQLSelect(m_schema.getlogCommence());
        }
        catch(Exception e)
        {
            m_uLog.LogMessage(kTHIS, "Could not create table from script ");
            m_uLog.LogException(kTHIS, e.toString());
            return;
        }
    }
/**
 *          output the data in nominated format
 *
 *          Returns             - n/a
 */
public void logInsertCommence(String a_query)
        {
        // log the Insert
        try
        {
            m_DBConnection.runSQLUpdate(a_query);
        }
        catch(Exception e)
        {
            m_uLog.LogMessage(kTHIS, "Could not insert log record ");
            m_uLog.LogException(kTHIS, e.toString());
            // most probably a duplicate key
            return;
        }
    }
/**
 *          output the data in nominated format
 *
 *
 *
 *          Returns             - n/a
 */
public void logInsertComplete()
        {
        // log the Insert completion
        String ls_query = m_schema.getlogComplete(gi_rowCount);
        try
        {
            m_DBConnection.runSQLUpdate(ls_query);
        }
        catch(Exception e)
        {
            m_uLog.LogMessage(kTHIS, "Could not insert log record ");
            m_uLog.LogException(kTHIS, e.toString());
            // most probably a duplicate key
            return;
        }
    }
}
```

```
//EOF DataInsertor
```

```
/**
 * Title:       DBConnection
 * Description:  The class acts as a retriever container for the table management process
 *
 * Copyright:    Copyright (c) ajc<p>
 * Company:      <p>
 * @author ajc
 * @version 1.0
 *
 *
 *  Written by and Adrian Collins
 *
 *  RELEASE - Sept 21st 2001
 */
import java.io.*;
import java.sql.Date;
import java.sql.*;
import ExceptionLog;

public class DBConnection
{

        private final static String kDBDSN      = "dqm_ss7";
        private final static String kLOGFILE    = "dqm_server_DB.log";
        private final static String kUID        = "sa";
        private final static String kPWD        = null;
        static final String kTHIS      = "DBConnection";
        public Connection  m_dbsConn            = null;
        public ExceptionLog m_uLog              = null;

        // closes the database connection
        public void Close()
        {
          try
          {
            m_uLog.LogMessage(kTHIS," Closing connection to database ");
        m_dbsConn.close();
          }
          catch(Exception x)
          {}
        }
/**
 *       Method     - Creates a new connection
 *
 *    Throws          - Exception is thrown up the chain if the connection object
 *         fails to connect - this will be a fatal error
 *
 *       Returns              - n/a
 */
        public DBConnection(String a_DB, String a_uid, String a_pwd) throws Exception
        {
          boolean   isRegistered = false;
       String    DB   = a_DB;
          String    url  = "jdbc:odbc:" + DB;
          String    pwd  = a_pwd;
          String    uid  = a_uid;

      try
      {
        m_uLog = new ExceptionLog(kLOGFILE);
        // set loggin on
        m_uLog.setLog(true);
      }
      catch (Exception e)
      {
        m_uLog.LogException(e.toString(),"Logging Process could not be started...");
        System.exit(1);
      }


      // Load the jdbc-odbc bridge driver
          Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");

      // Attempt to connect to a driver.
      m_uLog.LogException("Debug", "Attempting to connect to database " + url);
      m_uLog.LogMessage(kTHIS," Attempting to connect to database " + url);
```

```
            m_dbsConn = DriverManager.getConnection (url, uid, pwd);
    }


/**
 *      Method              - Retrieves the desired SQL select statement and
 *            returns a Result Set
 *
 *
 *      Arguements          - some sql string
 *
 * Throws                   - Exception to notify caller functions that it failed (handled above - non-fatal error)
 *
 *      Returns             - Result Set
 */
        public  ResultSet runSQLSelect(String a_SQL)throws Exception
        {
                ResultSet  rs    = null;
                String     query1 = a_SQL;
        try
                {
                  Statement  stmt1  = m_dbsConn.createStatement ();
          rs = stmt1.executeQuery (query1);
          return rs;
        }
        catch (SQLException ex)
                {
                  // A SQLException was generated.

             String line = "\n*** CreateStatement SQLException caught ***\n";

                          while (ex != null)
                          {
                m_uLog.LogMessage(kTHIS," tSQLState: = " + ex.getSQLState ());
                m_uLog.LogMessage(kTHIS," tMessage:  " + ex.getMessage ());
                line = line + "\n\t\tMessage:  " + ex.getMessage ();
                m_uLog.LogMessage(kTHIS," tVendor:   " + ex.getErrorCode ());
                ex = ex.getNextException ();
                          }
                          m_uLog.LogException(ex.toString(),line);
                          throw new SQLException();
                }
        }
/**
 *      Method              - Retrieves the desired SQL select statement and
 *            returns a Result Set
 *
 *
 *      Arguements          - some sql string
 *
 * Throws                   - Exception to notify caller functions that it failed (handled above - non-fatal error)
 *
 *      Returns             - Result Set
 */
        public  void runSQLUpdate(String a_SQL)throws Exception
        {
                  String     query1 = a_SQL;
        int retcode = 0;
                  try
                  {
                    Statement  stmt1  = m_dbsConn.createStatement ();
          retcode = stmt1.executeUpdate(query1);
        }
        catch (SQLException ex)
                  {
                    // A SQLException was generated.
                    String line = "\n*** CreateStatement SQLException caught ***\n";

                          while (ex != null)
                          {
                                  line +="\n\t\tSQLState: " + ex.getSQLState ();
                                  line +="\n\t\tMessage:  " + ex.getMessage ();
                                  line +="\n\t\tVendor:   " + ex.getErrorCode ();
                                  ex = ex.getNextException ();
                          }
                          m_uLog.LogException(ex.toString(),line);
```

```
                    throw new SQLException();
                }
        }

/**
 *      Method          - Set Auto commit mode to false
 *      Returns         - n/a
 */
        public  void setOffAutoCommit()throws SQLException
        {
       try
                {
                  m_dbsConn.setAutoCommit (false);
        }
        catch (SQLException ex)
                {
                  // A SQLException was generated.

                  String line = "\n*** Set Auto Commit SQLException caught ***\n";

                        while (ex != null)
                        {
                                line +="\n\t\tSQLState: " + ex.getSQLState ();
                                line +="\n\t\tMessage:  " + ex.getMessage ();
                                line +="\n\t\tVendor:   " + ex.getErrorCode ();
                                ex = ex.getNextException ();
                        }
                        m_uLog.LogException(ex.toString(),line);
                        throw new SQLException();
                }
        }
        public  void setCommit()throws SQLException
        {
       try
                {
                  m_dbsConn.commit();
        }
        catch (SQLException ex)
                {
                  // A SQLException was generated.

                  String line = "\n*** Commit SQLException caught ***\n";

                        while (ex != null)
                        {
                                line +="\n\t\tSQLState: " + ex.getSQLState ();
                                line +="\n\t\tMessage:  " + ex.getMessage ();
                                line +="\n\t\tVendor:   " + ex.getErrorCode ();
                                ex = ex.getNextException ();
                        }
                        m_uLog.LogException(ex.toString(),line);
                        throw new SQLException();
                }
        }

}

//EOF DBConnection
/*****************************************************************************************************************
 *
 * class ElementAttribute       - Class used as a type-safe container for a table attribute.
 * Written by Adrian Collins
 *
 * RELEASE - Sept 25th 2001
 *
 */

class ElementAttribute
{
     public String      m_Type  = "n/a";
     public String      m_Line  = "n/a";
     public String      m_Name  = "n/a";
     public String      m_Title = "n/a";
     public String      m_Code  = "n/a";
     public String      m_Value = "n/a";
     static final String kTHIS     = "ElementAttribute ";
```

```
/**
 *          Method      Stores the xsd data for review by a service requestor
 *
 *      Throws          - Exception is thrown up the chain if the connection object
 *              fails to connect - this will be a fatal error
 *
 *          Returns             - n/a
            public ElementAttribute()
            {

        }
*/
        public void setName(String a_Name)
            {
                    m_Name = a_Name;
            }

        public void setTitle(String a_Title)
            {
                    m_Title = a_Title;
            }

        public void setCode(String a_Code)
            {
                    m_Code = a_Code;
            }

        public void setValue(String a_Value)
            {
                    m_Value = a_Value;
            }
        public void setType(String a_Type)
            {
                    m_Type = a_Type;
            }

        public String getName()
            {
                    return m_Name;
            }

        public String getTitle()
            {
                    return m_Title;
            }

        public String getCode()
            {
                    return m_Code;
            }

        public String getValue()
            {
                    return m_Value;
            }
        public String getType()
            {
                    return m_Type;
            }

}
//EOF ElementAttribute
```

```
/**
 * Title:       ElementProperties
 * Description:  Container class for the xsd data
 * @author ajc
 * @version 1.0
 *
 *
 *  Written by Adrian Collins
 *
 *  RELEASE - 15/05/01
 */
import java.io.*;
import java.util.*;
import ElementAttribute;
import java.util.Vector;

public class ElementProperties
{

      private String gs_elementName = "n/a";
      static final String kTHIS     = "ElementProperties ";
      private Vector v_attributes = null;
      private ElementAttribute m_attribute = null;

/**
 *          Method      Stores the xsd data for review by a service requestor
 *
 *     Throws           - Exception is thrown up the chain if the connection object
 *              fails to connect - this will be a fatal error
 *
 *          Returns                 - n/a
 */
            public ElementProperties(String a_name) throws Exception
            {

         gs_elementName   = a_name;
         v_attributes = new Vector();
      }
*           Returns   - the name of the element/schema
 */
            public String getName()
            {

         return gs_elementName;
      }

/**
 *          Method                  - get name of the url
 *
 *                      - stored in the attribute named "schema"
 *
 *          Throws                  - DataFormatException
 *
 *          Returns   - the schema url
 */
            public String getSchemaURL()
            {
         String ls_value = null;

         for(int i=0;i<v_attributes.size();i++)
         {
            m_attribute = (ElementAttribute) v_attributes.elementAt(i);
            if (m_attribute.getName().equals("schema"))
            {
               ls_value = m_attribute.getValue();
               try
               {
                  if (!(ls_value==null))
                  {
                     return  ls_value;
                  }
                  else
                  {
                     throw new Exception();
                  }
               }
```

```
                    catch(Exception e)
                    {
                System.out.println(kTHIS + " Could not access schema URL, mandatory data - ends");
            System.exit(1);
                    }
        }
      }
      return null;
    }
/**
 *      Method          - get script
 *
 *              - stored in the attribute named "script"
 *
 *      Throws          - DataFormatException
 *
 *      Returns  - the schema url
 */
      public String getScript()
      {
    String ls_value = null;

    for(int i=0;i<v_attributes.size();i++)
    {
      m_attribute = (ElementAttribute) v_attributes.elementAt(i);
      if (m_attribute.getName().equals("script"))
      {
        ls_value = m_attribute.getValue();
        try
        {
          if (!(ls_value==null))
          {
            return  ls_value;
          }
          else
          {
            throw new Exception();
          }
        }
              catch(Exception e)
              {
            System.out.println(kTHIS + " Could not access script, tough");
              }
      }
    }
    return null;
  }
/**
 *      Method          - get name of the localhost url
 *
 *              - stored in the attribute named "local"
 *          - this one is used when the remote host url cannot be contacted
 *      Throws          - DataFormatException
 *
 *      Returns  - the schema url
 */
      public String getLocalSchemaURL()
      {
    String ls_value = null;

    for(int i=0;i<v_attributes.size();i++)
    {
      m_attribute = (ElementAttribute) v_attributes.elementAt(i);
      if (m_attribute.getName().equals("local"))
      {
        ls_value = m_attribute.getValue();
        try
        {
          if (!(ls_value==null))
          {
            return  ls_value;
          }
          else
          {
            throw new Exception();
          }
```

```
            }
                    catch(Exception e)
                    {
                System.out.println(kTHIS + " Could not access schema URL, mandatory data - ends");
            System.exit(1);
                    }
        }
      }
      return null;
    }

/**
 *      Method          - get number of column in the expected schema
 *
 *              - stored in the attribute named "count"
 *
 *      Throws          - DataFormatException
 *
 *      Returns  - number of columns in the schema
 */
      public int getColumnCount()
      {
      String ls_value = null;

      for(int i=0;i<v_attributes.size();i++)
      {
        m_attribute = (ElementAttribute) v_attributes.elementAt(i);
        if (m_attribute.getName().equals("count"))
        {
          ls_value = m_attribute.getValue();
          try
          {
            if (!(ls_value==null))
            {
              int i_colCount = Integer.parseInt(ls_value.trim());
              return  i_colCount;
            }
            else
            {
              throw new Exception();
            }
          }
                    catch(Exception e)
                    {
                System.out.println(kTHIS + " Could not access schema URL, mandatory data - ends");
            System.exit(1);
                    }
        }
      }
      return -1;
    }
```

```
/**
*       Method          - Searches the string for a specific word, and
*                               The word must not contain any whitespace characters
*
*       Arguements      - word - the word to search for
*
*       Throws          - IOException
*
*       Returns   - the last position of the word in the string, OR 0 if not found.
*/
      public void loadAttribute(String ls_Line)
      {
    String a, b;
    try
    {
      StringTokenizer st = new StringTokenizer(ls_Line,"\"<=");
      a = st.nextToken();
      a = st.nextToken();
      // we know that the string 'attribute name' is buried in this string
      // we want tokens with the following tags:
      // attribute name
      // title
      // code
      // type
      // value
      a.toLowerCase();
      if (a.equals("attribute name "))
      {
        m_attribute = new ElementAttribute();
        m_attribute.setName(st.nextToken());
        while (st.hasMoreTokens())
        {

          b = st.nextToken();
              b.toLowerCase();
          if (b.equals("title")||b.equals(" title"))
          {
            m_attribute.setTitle(st.nextToken());
          }
          if (b.equals("code")||b.equals(" code"))
          {
            m_attribute.setCode(st.nextToken());
          }
          if (b.equals("type")||b.equals(" type"))
          {
            m_attribute.setType(st.nextToken());
          }
          if (b.equals("value")||b.equals(" value"))
          {
            m_attribute.setValue(st.nextToken());
          }
        }
        // add the elements to the vector;
        v_attributes.addElement(m_attribute);
      }
      // after all attributes have been loaded, resize the vector accordingly
      v_attributes.trimToSize();
    }
        catch(Exception e)
        {
      System.out.println(kTHIS + " Could not access schema URL, mandatory data - ends");
      return;
        }
      }
```

```
/**
 *          Method              - return the set of xml codes in the element
 *
 *
 *          Throws              - IOException
 *
 *          Returns   - an array of strings
 */
            public String [] getCode()
            {
        String [] ls_codes = new String[v_attributes.size()];
        for(int i=0;i<v_attributes.size();i++)
        {
            m_attribute = (ElementAttribute) v_attributes.elementAt(i);
            ls_codes[i] = m_attribute.getCode();
        }
        return ls_codes;
    }
/**
 *          Method              - return the set of column names
 *
 *
 *          Throws              - IOException
 *
 *          Returns   - an array of strings
 */
            public String [] getColumnNames()
            {
        String [] ls_names = new String[v_attributes.size()];
        for(int i=0;i<v_attributes.size();i++)
        {
            m_attribute = (ElementAttribute) v_attributes.elementAt(i);
            ls_names[i] = m_attribute.getName();
        }
        return ls_names;
    }
/**
 *          Method              - return the set of column data types
 *
 *
 *          Throws              - IOException
 *
 *          Returns   - an array of strings
 */
            public String [] getColumnTypes()
            {
        String [] ls_types = new String[v_attributes.size()];
        for(int i=0;i<v_attributes.size();i++)
        {
            m_attribute = (ElementAttribute) v_attributes.elementAt(i);
            ls_types[i] = m_attribute.getType();
        }
        return ls_types;
    }
}
//EOF ElementProperties
```

```
/*****************************************************************************************************************
 *
 *          ExceptionLog        - Class provides an exception logging mechanism.
 *                                                    Logs Run-Time errors as exceptions in a given log file.
 *                                                    Does not truncate the log file.
 *
 */

import java.io.*;
import java.util.*;
import java.net.*;
import java.sql.*;

import ANSIFileStream;

public class ExceptionLog
{
// Strings that identify pre-defined exceptions
    ANSIFileStream outLog;
        private final static String kIOException = new IOException().toString();
        private final static String kEOFException = new EOFException().toString();
        private final static String kInterruptedIOException = new InterruptedIOException().toString();
        private final static String kSocketException = new SocketException().toString();
        private final static String kBindException = new BindException().toString();
        private final static String kConnectException = new ConnectException().toString();
        private final static String kNoRouteToHostException = new NoRouteToHostException().toString();
        private final static String kUnknownHostException = new UnknownHostException().toString();
        private final static String kRuntimeException = new RuntimeException().toString();
        private final static String kIllegalThreadStateException = new IllegalThreadStateException().toString();
        private final static String kNumberFormatException = new NumberFormatException().toString();
        private final static String kIndexOutOfBoundsException = new IndexOutOfBoundsException().toString();
        private final static String kNegativeArraySizeException = new NegativeArraySizeException().toString();
        private final static String kNoSuchElementException = new NoSuchElementException().toString();
        private final static String kNullPointerException = new NullPointerException().toString();
        private final static String kSQLException = new SQLException().toString();
        private final static String kSQLWarning = new SQLWarning().toString();
        private final static String kIllegalArgumentException  = new IllegalArgumentException().toString();

        private String m_sFname = null;
    private boolean m_softLog = true;

/**
 *          Constructor - Creates a File Stream to log file
 *
 *          Arguements          - fname - the name of the logfile
 *
 *          Throws              - none
 *
 * Returns                      - n/a
 */
        public ExceptionLog(String fname)
        {
          try
          {
                  m_sFname = fname;
          }
          catch(Exception e)
          {}
        }
```

```
/**
 *      Method          - Creates an exception of the appropriate type (except)
 *                        and a given detail message (msg)
 *                        Logs an Exception e to a Log File
 *                        Logged Exceptions are written as Strings in the format:
 *
 *                              DATE Exception_Type - location, action taken
 *
 *      Arguements      - excpt - is the String that identifies the type of exception,
 *                        msg - is the detailed message for the exception
 *
 *      Throws          - none
 *
 *      Returns         - void
 */
        public synchronized void LogException(String excpt, String msg)
        {
        try
                {
            if (outLog == null)
            {
              ANSIFileStream outLog = new ANSIFileStream(m_sFname);
            }
                        Exception e;
                        if( excpt.equals(kIOException))
                        {
                                e = new  IOException(msg);
                        }
                        else if( excpt.equals(kEOFException))
                        {
                                e = new  EOFException(msg);
                        }
                        else if( excpt.equals(kInterruptedIOException))
                        {
                                e = new  InterruptedIOException(msg);
                        }
                        else if( excpt.equals(kSocketException))
                        {
                                e = new  SocketException(msg);
                        }
                        else if( excpt.equals(kBindException))
                        {
                                e = new  BindException(msg);
                        }
                        else if( excpt.equals(kConnectException))
                        {
                                e = new  ConnectException(msg);
                        }
                        else if( excpt.equals(kNoRouteToHostException))
                        {
                                e = new  NoRouteToHostException(msg);
                        }
                        else if( excpt.equals(kUnknownHostException))
                        {
                                e = new  UnknownHostException(msg);
                        }
                        else if( excpt.equals(kRuntimeException))
                        {
                                e = new  RuntimeException(msg);
                        }
                        else if( excpt.equals(kIllegalThreadStateException))
                        {
                                e = new  IllegalThreadStateException(msg);
                        }
                        else if( excpt.equals(kNumberFormatException))
                        {
                                e = new  NumberFormatException(msg);
                        }
                        else if( excpt.equals(kIndexOutOfBoundsException))
                        {
                                e = new  IndexOutOfBoundsException(msg);
                        }
                        else if( excpt.equals(kNegativeArraySizeException))
                        {
                                e = new  NegativeArraySizeException(msg);
```

```
                }
                else if( excpt.equals(kNoSuchElementException))
                {
                        e = new   NoSuchElementException(msg);
                }
                else if( excpt.equals(kNullPointerException))
                {
                        e = new   NullPointerException(msg);
                }
                else if( excpt.equals(kSQLException))
                {
                        e = new   SQLException(msg);
                }
                else if( excpt.equals(kSQLWarning))
                {
                        e = new   SQLWarning(msg);
                }
                else if( excpt.equals(kIllegalArgumentException))
                {
                        e = new   IllegalArgumentException (msg);
                }
                else if( excpt.equals("Debug"))
                {
                        String line = GetCurrentDate() + "\tDebug.....:" + msg;
                        System.out.println(line);
                        outLog.AppendLine(line);
                        return;
                }
                else
                {
                        e = new   Exception(msg);
                }

                        String line = GetCurrentDate() + "\tException :" + e.toString();
                        System.out.println(line);
                        outLog.AppendLine(line);
                        outLog.Close();
        }
        catch(Exception e)
        {}
    }

public synchronized void setLog (boolean a_toggle)
{
   m_softLog = a_toggle;
}
public synchronized boolean getLogStatus ()
{
   return m_softLog;
}
```

```
/**
 *      Method          - Creates an exception of the appropriate type (except)
 *                        and a given detail message (msg)
 *                        Logs an Exception e to a Log File
 *                        Logged Exceptions are written as Strings in the format:
 *
 *                                DATE Exception_Type - location, action taken
*
 *      Arguements      - excpt - is the String that identifies the type of exception,
 *                        msg - is the detailed message for the exception
 *
 *      Throws          - none
 *
 *      Returns         - void
 */
        public synchronized void LogMessage(String Process, String msg)
        {
                try
                {
          if (getLogStatus())
          {
            if (outLog == null)
            {
                        ANSIFileStream outLog = new  ANSIFileStream(m_sFname);
                        String line = "\t" + GetCurrentDate() + "; Message from " + Process + " <----> " + msg;
                  outLog.AppendLine(line);

            }
          }
        }
                catch(Exception e)
                {}
    }
/**
 *      Method          - Creates an exception of the appropriate type (except)
 *                        and a given detail message (msg)
 *                        Logs an Exception e to a Log File
 *                        Logged Exceptions are written as Strings in the format:
 *
 *                                DATE Exception_Type - location, action taken
 *
 *
 *      Arguements      - excpt - is the String that identifies the type of exception,
 *                        msg - is the detailed message for the exception
 *
 *      Throws          - none
 *
 *      Returns         - void
 */
        public synchronized void LogMessage(String Process, int number)
        {
                try
                {
          if (getLogStatus())
          {
            if (outLog == null)
            {
                        ANSIFileStream outLog = new  ANSIFileStream(m_sFname);
                        String line = "\t" + GetCurrentDate() + "; Message from " + Process + " <----> " + number;
                  outLog.AppendLine(line);

            }
          }
        }
                catch(Exception e)
                {}
    }
```

```
/**
*       Method          - Creates a current date string in the format of:
*
*                                       dd/mm/yyyy, hh:mm:ss
*
*       Arguements      - none
*
*       Throws          - none
*
*       Returns - String representing the current date
*/
        public synchronized static String GetCurrentDate()
        {
                String strdate = null;

                java.util.Calendar rn = java.util.Calendar.getInstance();
                java.util.TimeZone tz = java.util.TimeZone.getTimeZone("GMT");
                rn.setTimeZone(tz);
                java.util.Date currentDate = rn.getTime();
                long t = currentDate.getTime();
                //Uncomment this line if default locale set to GMT
                //long t = currentDate.getTime()+(9*60*60*1000);

                java.sql.Date sqldt = new java.sql.Date(t);
                java.sql.Time sqltm = new java.sql.Time(t);

                String sqld = sqldt.toString();
                java.util.StringTokenizer st = new java.util.StringTokenizer(sqld,"-");
                String y = st.nextToken();
                String m = st.nextToken();
                String d = st.nextToken();

                strdate = d + "/" + m + "/" + y + ", " + sqltm.toString();

                return strdate;
        }
/**
*       Method          - Closes log
*
*
*       Arguements      - none
*
*       Throws          - none
*
*/
        public synchronized void Close()
        {
       try
       {
                outLog.Close();
       }
       catch(Exception e){}
        }
}
//EOF ExceptionLog
```

```
/*****************************************************************************************************************
 *
 *          class ExtractRequestHandler  - Handles an Archiving Client request
 *
 * Written by Adrian Collins
 *
 * RELEASE - Sept 24th 2001
 */

import javax.servlet.http.*;
import java.util.*;

public class ExtractRequestHandler extends RequestHandler implements Runnable
{
   private final static String kTHIS       = "ExtractRequestHandler";
   private final static String kWEBADD     = "/ServletData/";


/**
 *          Method              - Constructs a ExtractRequestHandler Object
 *                - Performs Validation Checks on connection request
 *
 *          Arguements          - aSock - the socket that will be used for communication with client
 *                - el    - logfile Stream
 *
 * Throws                       - Exception - signalling that connection was refused
 *
 *          Returns             - n/a
 */


          public ExtractRequestHandler(ExceptionLog a_uLog, RunParms  runParms, Vector a_Vmessage)
          {
            super (a_uLog, runParms, a_Vmessage);

     }
/**
 *          Method              - ExtractRequestHandler sheck files thread
 *
 *
 *          Arguements          - none
 *
 * Throws                       - none
 *
 *          Returns             - none
 */
   public void run()
          {
        getFileName();
        respondToServlet();
        persistance();
        // wait at least as long as the client waits for the file name
        CleanExit();
     }

}
//EOF extractrequestHandler
```

```
/*****************************************************************************************************************
 *
 *          class InsertRequestHandler    - The running scenario for this module is as follows:
 *
 *            collect the data
 *              build a Vector of keys matching selection criteria
 *              Enumerate the Vector and build a class which contains
 *              the data for each enum. (row)
 *              send the data
 *
 *
 *
 *
 */
import java.util.*;
import java.io.*;
import java.net.*;
import java.sql.*;
import DataInsertor;
import RunParms;
import ExceptionLog;


public class InsertRequestHandler
{
   public ExceptionLog    m_uLog        = null;
   public boolean         m_bRunning    = true;
   public String          xml_Directory = null;    // path to data files
   public String          xsd_Directory = null;    // path to data files
   public String          s_File = null;    // data files
   private DataInsertor m_Insertor = null;
   private RunParms  m_runParms;
   private String m_FileURL = null;
   private String s_dsnName = null;
   private String s_userID = null;
   private String s_passWord = null;
   private final static String kTHIS       = "InsertRequestHandler";
   private final static String kLOGFILE    = "dqm_server.log";

/**
 *          Method              - Constructs a InsertRequestHandler Object
 *                - Performs Validation Checks on connection request
 *
 *  Throws                      - Exception - signalling that connection was refused
 *
 *          Returns             - n/a
 */


          public InsertRequestHandler()
          {
          try
              {

             m_uLog = new ExceptionLog(kLOGFILE);
             // set loggin on
             m_uLog.setLog(true);

        }
        catch(Exception e)
        {
             m_uLog.LogException(e.toString(),"Handler Unable to connect to DataBase");
             CleanExit();
        }
    }
```

```
/**
 *          Method              - InsertRequestHandler sheck files thread
 *
 *
 *          Arguements          - none
 *
 *  Throws                      - none
 *
 *          Returns             - none
 */
   public void runInsert()
   {
        System.out.println("Commencing Insertion Process");
        m_runParms = new RunParms();
        m_runParms.addParm("xmllocation");
        m_runParms.addValue(xml_Directory);
        m_runParms.addParm("xsdlocation");
        m_runParms.addValue(xsd_Directory);
        m_runParms.addParm("filename");
        m_runParms.addValue(s_File);
        m_runParms.addParm("dsn");
        m_runParms.addValue(s_dsnName);
        m_runParms.addParm("userid");
        m_runParms.addValue(s_userID);
        m_runParms.addParm("password");
        m_runParms.addValue(s_passWord);

        try
        {

           m_Insertor = new DataInsertor (m_uLog, m_runParms);
           m_Insertor.Initialise();
           m_Insertor.streamInput();
           m_Insertor.logInsertComplete();
           m_Insertor.Cleanup();

           CleanExit();
        }
        catch(Exception e)
        {
            m_uLog.LogMessage(kTHIS,"Insertion Process could not be completed...");
            m_uLog.LogException(kTHIS,"Insertion Process could not be completed...");
            System.exit(1);
        }

    }
/**
 *          Method              - Pass in the runParm
 *  Throws                      - Exception
 *
 *          Returns             - n/a
 */
public void passInRunParm(String axml_Directory, String as_filename, String axsd_Directory,
String as_dsnName, String as_userID, String as_passWord)
{
         try
              {
              xml_Directory = axml_Directory;
              xsd_Directory = axsd_Directory;
              s_File = as_filename;
              s_dsnName = as_dsnName;
              s_userID = as_userID;
              s_passWord = as_passWord;
         }
         catch(Exception e)
         {
              m_uLog.LogException(e.toString(),"Handler Unable to get runParms");
              CleanExit();
         }
    }
/**
 *          Method              - Tidies up and kills this thread
 *          - Sends a S_QUIT to appropriate Client
 *          - Closes and releases all open objects
 *
 *          Arguements          - none
```

```
*
* Throws                    - Exception
*
*       Returns             - none
*/
        private void CleanExit()
        {
          try
          {
            m_uLog.LogException(kTHIS,"Shut down via clean exit");
            m_bRunning = false;
        m_uLog.Close();

          }
          catch(Exception e)
          {
          m_uLog.LogException(e.toString(),"Handler Unable to shut down via clean exit");
          }
        }
}
//EOF InsertRequestHandler
```

```
/**********************************************************************************************************************
 *
 *          class PersistStream - Class extends functionality of ANSIFileStream to persist data base data to a
 *     file format
 *
 *
 * Written by Adrian Collins
 *
 * RELEASE - 7/5/01
 *
 */
import java.io.*;
import ANSIFileStream;
import ExceptionLog;

public class PersistStream extends ANSiFileStream
{
     private ExceptionLog     m_uLog          = null;
     static final byte kBACK_SLASH = (byte) '/';
          static final byte kCOLON  = (byte) ':';
          static final byte kFLOATING_POINT = (byte) '.';
          static final byte kNEGATIVE = (byte) '-';
          static final byte kUNDERSCORE = (byte) '_';
     private final static String kTHIS       = "PersistStream";

/**
 *          Constructor - Simply calls the ANSIFileStream constructor
 *
 *          Arguments          - fname - filename of desired file
 *
 *          Throws             - IOException
 *
 *          Returns            - n/a
 */
          public PersistStream(String fname) throws IOException
          {
                    super(fname);
          }

          public void passInLog(ExceptionLog a_uLog)
          {
          m_uLog = a_uLog;
          }
/**
 *          Method             - Writes the given data to file in line format
 *
 *          Arguments          - fname - filename of desired file
 *
 *          Throws             - IOException
 *
 *          Returns            - n/a
 */
          public void WriteXMLData(String val) throws IOException
          {
                    WriteLine(val);
          }

     /**
 *          Method             - Writes the given data to file in line format
 *
 *          Arguments          - fname - filename of desired file
 *
 *          Throws             - IOException
 *
 *          Returns            - n/a
 */
          public void WriteCSVData(String val) throws IOException
          {
          WriteLine(val);
          }

}
//EOF PersistStream
```

```
/*****************************************************************************************************************
 *
 * class QueryHandler          - Class used prepare request parameters for execution by DBConnection.
 *                 Prepared for use with the runParam container passed in at construction
 *
 *                 It is assumed the usage will be mostly generic, so extend as required
 *
 *                  Typical entries
 *                  Table    -  Table to be extracted
 *                  KeyName    - Multiplt keys may be passed in if required, if a
 *                          keyName is nominated, a corresponding KeyValue must be provided
 *                  KeyValue
 *
 * Written by Adrian Collins
 *
 * RELEASE - Sept 25th 2001
 *
 */
import java.util.*;
import ExceptionLog;
import RunParms;


class QueryHandler
{
    public ExceptionLog     m_uLog     = null;
    private String m_TableName = null, m_Query = null;
    private String [] m_Arg_Names = null, m_Arg_Values = null;
    private int m_Keys = 0;
    private RunParms  m_runParms;
    static final String kTHIS      = "QueryHandler ";
    ListIterator m_Parameters = null, m_Values = null;

/**
 *          Method     Stores the parameter data in a convenient container
 *
 *          Returns                - n/a
 */
          public QueryHandler(ExceptionLog a_uLog, RunParms  runParms) throws
IllegalArgumentException
          {
         m_uLog = a_uLog;
         m_runParms = runParms;
         setTableName();
         buildQuery();
         if (sizeKeys() == 0)
         {
            return;
         }
         loadKeys();
         buildWhereClause();

    }

/*          Method                - provide a public means of comparing the schema, with the
 *                table requested
 *
 * Throws                         - Exception
 *
 *          Returns               - String containing the name of the table to be queried
 */
          public String getTableName()
          {
         return m_TableName;
    }
```

```
/*      Method          - provide public access to the requested query
 *
 *
 *      Returns         - String containing the name of the table to be queried
 *              and any requestd where clause
 */
        public String getTableQuery()
        {
       return m_Query;
    }
/*      Method          - build sql query from run parameters entered by request
 *
 *
 * Throws               - Exception
 *
 *      Returns         - String containing the sql query to be executed
 */
        private void buildQuery()
        {
      m_Query = "Select * from " + m_TableName;

    }

/*      Method          - build sql query from run parameters entered by request
 *
 *
 * Throws               - Exception
 *
 *      Returns         - String containing the sql query to be executed
 */
        private void buildWhereClause()
        {
      m_Query = "Select * from " + m_TableName;
      m_Query = m_Query + " WHERE ";

      if (m_Keys > 0)
      {
        m_Query = m_Query + m_Arg_Names[0] + " = " + m_Arg_Values[0];
        for (int i = 1 ; i < m_Keys; i++)
        {
          m_Query = m_Query + " AND " + m_Arg_Names[i] + " = " + m_Arg_Values[i];
        }
      }
      m_uLog.LogMessage(kTHIS," Where Clause =" + m_Query);
    }
/**     Method          - provide a public means of comparing the schema, with the
 *              table requeste
 *
* Throws            - Exception
 *
 *      Returns         - String containing the name of the table to be queried
 */
        private void setTableName()
        {
       try
       {
         m_TableName = m_runParms.getParameterValue("table");

         if (m_TableName == null)
         {
           throw new Exception();
         }
       }
       catch(Exception e)
       {
           m_uLog.LogMessage(kTHIS," Could not determine the table name of the query");
           m_uLog.LogException(e.toString()," Could not determine the table name of the query");
           return;
       }
    }
/**      Method          - build a set of strings which are then used to append to the where clause
 *      Remember that we have no way of matching the parameter names with the respective parameter name
 *      For example if the sql query we are building requires two arguments in the where clause
 *      say "select * from xxxx where 'parameter_name_1' = 'parameter_value_1' AND
 *                      'parameter_name_2' = 'parameter_value_2'
 *      The servlet engine passes the arguments into the request buffer in random order
```

```
*       so I use a simple method of constraining the interface to accept only pairs named as follows
*          KeyName 1 or KeyName_1 must match KeyValue 1 or KeyValue_1 etc
*          KeyName 2 or KeyName_2 must match KeyValue 2 or KeyValue_2 etc
*
*       The number of the keyphrase is then used to position the matching pairs in an array
*
* Throws                - Exception
*
*       Returns         - n/a
*/
        private void loadKeys()
        {
       String parm = null, value= null;
       String token = null, tokenPre = null, tokenPost = null;
       int i = 0;
       m_Arg_Names = new String[m_Keys];
       m_Arg_Values = new String[m_Keys];
       m_Parameters = m_runParms.getParms();
       m_Values = m_runParms.getValues();
       try
       {

          while (m_Parameters.hasNext())
          {
             parm = (String) m_Parameters.next();
             StringTokenizer st = new StringTokenizer(parm," _");
             value = (String) m_Values.next();
             // we assume by now that the validation in the container has ensured
             // that the correct number of pairs are presented, so we will
             // increment j_count after keyname
             if (parm.startsWith("keyname"))
             {
                tokenPre = st.nextToken();
                m_uLog.LogMessage(kTHIS," tokenPre =: " + tokenPre);
                i = Integer.parseInt(st.nextToken().trim());
                //reduce the element count to enable the array to load
                // from element zero
                i--;
                m_Arg_Names[i] = value;

             }
             if (parm.startsWith("keyvalue"))
             {
                tokenPre = st.nextToken();
                i = Integer.parseInt(st.nextToken().trim());
                i--;
                m_Arg_Values[i] = value;

             }

          }
       }
       catch(Exception e)
       {
            m_uLog.LogMessage(kTHIS," Could not determine key value pairs");
            m_uLog.LogException(kTHIS," Could not determine key value pairs");
            System.exit(1);
       }
    }
```

```
/*          Method              - count the number of keyname parameters
 *
 * Throws                       - Exception
 *
 *          Returns             - n/a
 */
          private int sizeKeys()
          {
         String parm = null;
         m_Parameters = m_runParms.getParms();
         while (m_Parameters.hasNext())
         {
            parm = (String) m_Parameters.next();
            if (parm.startsWith("keyname"))
            {
               m_Keys++;
            }
         }
         return m_Keys;
      }

}
//EOF QueryHandler
```

```
/*****************************************************************************************************************
 *
 * class RunParms  - Class used as a type-safe container for parameters. Prepared by
 *                   servlet for use by each request handler thread
 *                   we are looking for the following input Parameters:
 *                   Table    - Table to be extracted
 *                   KeyName    - MultipIt keys may be passed in if required, if a
 *                                keyName is nominated, a corresponding KeyValue must be provided
 *                   KeyValue
 *                   Format    - Current values supported XML or CSV
 *
 * For those applications which do not wish to use the pairing constraint, the logic is as follows.
 * A counter is maintained to increment when the parameter name is "keyname" and decrement when the
 * parameter name is "keyvalue". When the parameter name is neither "keyname" nor "keyvalue"
 * the count should be zero. So to turn off, simply force zero into the counter prior to throw
 * the exception
 *
 * Written by Adrian Collins
 *
 * RELEASE - Sept 25th 2001
 *
 */
import java.util.*;

class RunParms
{
    private  String   m_Param = null, m_Value = null;
    private int m_Param_Count = 0, m_Value_Count = 0, m_Key_Count = 0;
    public  List m_Param_List = null, m_Value_List = null;
    static final String kTHIS     = "RunParms ";

/**
 *        Method     Stores the parameter data in a convenient container
 *
 *        Returns                - n/a
 */
        public RunParms()
        {
                m_Param_List = new ArrayList();
                m_Value_List = new ArrayList();
                  }
        public void addParm(String a_Parm)
        {
                  m_Param = a_Parm;
        try
        {
          if (m_Param_Count == m_Value_Count)
          {
            m_Param_List.add(m_Param);
            m_Param_Count++;
          }
          else
          {
            throw new Exception();
          }
          // code to constrain the matching of parameter pairs
          // to KeyName and KeyValue
          if (m_Param.startsWith("keyname"))
          {
            m_Key_Count++;
          }
          else
          if (m_Param.startsWith("keyvalue"))
          {
            m_Key_Count--;
          }
        }
                  catch(Exception e)
                  {
                  System.out.println(kTHIS + " Could not add Parameter, mandatory data - ends");
              System.exit(1);
                  }
    }
    public void addValue(String a_Value)
    {
        m_Value = a_Value;
```

```
            try
            {
               m_Value_Count++;
               if (m_Param_Count == m_Value_Count)
               {
                  m_Value_List.add(m_Value);
               }
               else
               {
                  throw new Exception();
               }
            }
                  catch(Exception e)
                  {
                  System.out.println(kTHIS + " Could not add Parameter, mandatory data - ends");
               System.exit(1);
                  }
      }
      public String getParameterValue(String a_Param)
      {
          // returns null if no match is found
          String parm, value = null;
          ListIterator m_Parameters, m_Values = null;
          m_Parameters = getParms();
          m_Values = getValues();
          while (m_Parameters.hasNext())
          {
             parm = (String) m_Parameters.next();
             value = (String) m_Values.next();
             if (parm.equals(a_Param))
             {
                return value;
             }
          }
          return value;
      }
      public ListIterator getValues()
      {
                  return m_Value_List.listIterator();
      }

      public ListIterator getParms()
      {
                  return m_Param_List.listIterator();
      }
/**
 *       Method     returns a status on the configuration of the container
 *    The count of key name and key value pairs must be the same
 *    the underlying list containers don't guarantee addition in any order
 *    so the count toggles between -1 and 1 when pairs are received, at end
 *    the counter must be zero
 *
 *       Returns              - n/a
 */
      public void getConfiguration()
      {
        try
        {
          if (m_Key_Count == 0)
          {
             return;
          }
          else
          {
             throw new Exception();
          }
        }
             catch(Exception e)
             {
          System.out.println(kTHIS + " Could not add Parameter, mandatory data - ends");
          System.exit(1);
        }
      }
}
//EOF RunParms
```

```
/*****************************************************************************************************************
 *
 *     servletExtractor Servlet constructed to broker the users request for data, and the
 *     method of presentation
 *
 *
 * Written by Adrian Collins
 *
 * RELEASE - 24th Sept 2001
 *
 */
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import ExtractRequestHandler;
import ExceptionLog;
import RunParms;

public class servletExtractor extends HttpServlet
{
   PrintWriter          out;
   private HttpServletResponse m_response;
   private ExceptionLog       m_uLog          = null;
   RunParms  runParms = null;
   private Vector    m_Vmessage = null;
   private ExtractRequestHandler   extractRequest = null;
   private Thread requestThread = null;
   private String m_File = null;
   private int m_timerCount = 0;
   final static int kWAIT       = 3;
   final static int kSLEEP       = 5000;
   final static char kQUOTE       = '"';
   final static String kTHIS       = "ServletExtractor";
   private final static String kLOGFILE    = "dqm_server.log";

   public void service (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
   {
      m_response = response;
      try
      {
         m_uLog = new ExceptionLog(kLOGFILE);
         m_Vmessage = new Vector();
         // set loggin on
         m_uLog.setLog(true);
      }
      catch (Exception e)
      {
         m_uLog.LogException(e.toString(),"Server Process could not be started...");
         System.exit(1);
      }
      processRequest(request, response);
   }

   public synchronized void processRequest(HttpServletRequest request, HttpServletResponse
response)throws ServletException, IOException
   {

      Enumeration inputParms;
      runParms = new RunParms(m_uLog);
      ListIterator m_Parameters, m_Values = null;
      String inParam, curParam, inValue, curValue, outSolution;
      int i_keyCount = 0;
      out = m_response.getWriter();
      String  title = "Data Extraction Request";

            // set content type and other response header fields first
      m_response.setContentType("text/html");

            // then write the data of the response
            out.println("<HTML><HEAD><TITLE>");
            out.println(title);
            out.println("</TITLE></HEAD><body BACKGROUND=" + kQUOTE + "/images/backgrnd.gif" + kQUOTE +
            " text=" + kQUOTE + "#043E07" + kQUOTE + " link=" + kQUOTE + "#990000" + kQUOTE + " vlink=" +
```

```
        kQUOTE + "#666666" + kQUOTE + " alink=" + kQUOTE + "#CC9900" + kQUOTE + "><font face=" +
        kQUOTE + "trebuchet ms, arial, helvetica" + kQUOTE + ">");
        out.println("<H1>" + title + "</H1>");

    // we are looking for the following input Parameters:
    // Table    - Table to be extracted
    // KeyName   - Multiplt keys may be passed in if required, if a
    //           keyName is nominated, a corresponding KeyValue must be provided
    // KeyValue
    // Format   - Current values supported XML or CSV


    inputParms = request.getParameterNames();
    // create a parameter pair for this host
    runParms.addParm("host");
    runParms.addValue(request.getServerName());

    while (inputParms.hasMoreElements())
    {

     // load up the runParams container with all the gear
     // then pass it into the thread.
     // Used by the persistance class to determine the output type
     // Used by the database connection to prepare the sql request

     inParam = (String) inputParms.nextElement();
     curParam = inParam.toLowerCase();
     inValue = request.getParameter(inParam);
     if (!(inValue == null)) // belt and braces
     {
        curValue = inValue.toLowerCase();
        runParms.addParm(curParam);
        runParms.addValue(curValue);
     }

    }
    runParms.getConfiguration();
    try
    {
       curValue = runParms.getParameterValue("button");
       if (curValue.equals("extract data"))
       {
          // creating and starting a ExtractRequestHandler thread
          extractRequest = new ExtractRequestHandler(m_uLog, runParms, m_Vmessage);
          requestThread = new Thread(extractRequest);
          requestThread.start();
          outSolution = pollThread();

          if (outSolution != null)
          {
          out.println("<a href=http://" + request.getServerName() + "/ServletData/" + outSolution + ">Your Request ("
          + outSolution + ") is being Processed </a>");
          out.println("<p><center> <h4>Note: Until extraction processing has completed, selection will result in an
          error (HTTP 404 - File not found) </h4></center></p>");
          }
          else
          {
             out.println("<H3> Could not create extract file at this time </H3>");
          }
       }
       else
       {
          throw new Exception();
       }
    }
    catch(Exception e)
    {
       m_uLog.LogMessage(kTHIS," Could not determine the type of persistance, mandatory data - ends");
       System.exit(1);
    }
    out.println("</BODY></HTML>");
          out.close();

}
public synchronized String pollThread()
{
```

```
    boolean b_ok = true;
    String l_host = null;
    int i = 0;
        // Check the vector a couple of times to see if a message has been posted
    try
    {
      m_timerCount = 0;
      while (b_ok)
      {
        if (m_Vmessage.isEmpty())
        {
            if (!(waitForThread()))
            {
              throw new Exception();
            }
        }
        else
        {
          l_host = m_Vmessage.toString();
          i = l_host.length();
          // take out the stupid parenthases left by the toString
          String temp = l_host.substring(1,(i-1));
          l_host = temp;
          break;
        }
      }
     }
     catch(Exception e)
          {
            m_uLog.LogMessage(kTHIS,"<H3> Could not contact Host </H3>");
       l_host = null;
       return l_host;
     }
     return l_host;
  }
  public synchronized boolean waitForThread()
  {
    m_timerCount++;
    m_uLog.LogMessage(kTHIS," waitForThread " + m_timerCount);
    if (m_timerCount > kWAIT){return false;}
    try
    {
      Thread.sleep(kSLEEP);
    }
    catch(Exception e)
    {
      return false;
    }

    return true;
  }

}
//EOF servletExtractor
```

```
/**********************************************************************************************************************
 *
 *          class TableAchiveClient       - Listens on a given input directory, detects weld data which has been
 *      deposited into the input directory for insertion into the database
 *
 * Written by Adrian Collins
 *
 * RELEASE - Saturday Jan 8th 2002
 *
 */
import java.io.*;
import java.net.*;
import java.util.*;
import java.sql.*;
import DBConnection;
import InsertRequestHandler;

public class TableArchiveClient
{
        final static String kTHIS      = "TableArchiveClient";
        private long          m_nSleeptime = 0;
        public InsertRequestHandler   m_RequestHandler;    // File Trasporting class
        public String         s_HostName = null;     // Host name
                public String         s_dsnName = null;     // ODBC DSN name
                public String         s_userID = null;      // User ID
                public String         s_passWord = null;    // Password
        public String         xml_Directory = null;    // path to data files
        public String         xsd_Directory = null;    // path to data files
        public String         s_File = null;    // data files

    public TableArchiveClient(String argv[])
    {
            s_HostName   = argv[0].trim();
                    s_dsnName   = argv[1].trim();
            m_nSleeptime = Integer.parseInt(argv[2].trim());
            s_userID    = "sa";
            s_passWord    = null;

    }
    private void Sleep(long sec)
    {
        System.out.println(kTHIS + " Sleeping");
        try{Thread.sleep(sec*1000);}
        catch(Exception x){}
    }
```

```
/**
*       Method          - checks cpmmand line arguments are OK
*
*       Arguements      - argv[] - [DataBase DSN name] [uid] [passwd]
*
* Throws                - none
*
*       Returns         - none
*/
        public static void main(String argv[])
        {
        try
        {
                if(argv.length < 3)
                {
                        System.exit(1);
                }


            try
            {
               TableArchiveClient cl = new TableArchiveClient(argv);
               cl.runClient();
            }
            catch (Exception e)
            {
               System.out.println("Logging Process could not be started...");
               System.exit(1);
            }
        }
        catch(Exception e)
        {
                System.out.println("Couldn't interpret first argument as host name");
                System.exit(0);
        }

   }
/**
*       Method          - Retrieves the appropriate schema URL for the desired extraction table
*               returns the URL as a string
*               We need to get two pieces of information
*               the host details and the schema file name
*
*
*       Arguements      - a table name
*
* Throws                - Exception to notify caller functions that it failed (handled above - non-fatal error)
*
*       Returns         - String
*/
        public void runClient()
    {
        String ls_oldname = null;
        String ls_newname = null;
        String ls_fileLocation = null;
        try
        {
           m_RequestHandler = new InsertRequestHandler();
           getDirInfo();
        }
        catch (Exception e)
        {
           System.out.println("Insertion Process could not be started...");
           System.exit(1);
        }
```

```
        for(;;)
        {

            try
            {
               File   listeningPost = new File(xml_Directory);  // listening directory
                      String  fileList[] = listeningPost.list();      // a listing of the directory

               // bib and braces
               if (listeningPost.exists())
               {
                  for (int i=0; i < fileList.length; i++)
                  {

                     ls_oldname = fileList[i].toLowerCase();
                     // First test that the application can have the file, ie. The copy in
                     // has indeed finished with the creation process
                     ls_newname = aquireFile(xml_Directory, ls_oldname);
                     if(ls_newname!=null)
                     {
                        System.out.println(kTHIS + " Processing "+ ls_oldname);
                              m_RequestHandler.passInRunParm(xml_Directory, ls_newname, xsd_Directory,
                              s_dsnName, s_userID, s_passWord );
                              m_RequestHandler.runInsert();
                     }
                  }
               }
            }
            catch(Exception e)
            {
               System.out.println(kTHIS + " Insertor encountered an error while preparing files");
            }

            Sleep(m_nSleeptime);

        }
    }
/**
*       Method              - Listens to the input directory, and detects whether all
*         the files for a specific batch have arrived
*
*       Arguments           - none
*
*       Throws              - none
*
*       Returns             - fprefix - the input filename prefix if all the required files are here
*         null   - if all the required files are not here
*/
    private String aquireFile(String as_fileLocation, String as_name)
    {
        String ls_suffix = "old";

        // leave out the old files
        if (as_name.endsWith(ls_suffix))
           {
           return null;
        }

        File oldfile = new File(as_fileLocation+as_name);
             File nfile = new File(as_fileLocation+as_name+ls_suffix);

        if ((oldfile.exists()) && (oldfile.renameTo(nfile)))
        {
           return as_name+ls_suffix;
        }
             else
             {
                return null;
        }
      }
/**
*       Method              - Listens to the input directory, and detects whether all
*         the files for a specific batch have arrived
*
*       Arguments           - none
*
```

```
*       Throws              - none
*
*       Returns             - fprefix - the input filename prefix if all the required files are here
*         null   - if all the required files are not here
*/
    private void getDirInfo()
    {

        String    query = null;
        ResultSet  m_rs    = null;
        try
        {

            DBConnection m_DBConnection  = new DBConnection(s_dsnName, s_userID, s_passWord);
            query = "SELECT host_drop_directory FROM weld_host WHERE host_name = " +  "'" + s_HostName
+ "'" ;


            m_rs =  m_DBConnection.runSQLSelect(query);
            // assume only a singleton exists

            if (m_rs.next())
            {
               xml_Directory = m_rs.getString("host_drop_directory");
            }


            m_rs    = null;
            query = "SELECT host_schema_location FROM weld_host WHERE host_name =
                  " +  "'" + s_HostName + "'" ;

            m_rs =  m_DBConnection.runSQLSelect(query);
            // assume only a singleton exists
            if (m_rs.next())
            {
               xsd_Directory = m_rs.getString("host_schema_location");
            }
            m_DBConnection.Close();
        }
        catch(Exception e)
        {
           System.out.println(kTHIS + " Host Details could not be located...");
           System.exit(1);
        }

    }

}
//EOF TableAchiveClient
```

```
/**********************************************************************************************************************
 *
 * class TablePersistance - Class used as a type-safe container for the data
 *                  contained in a single object. The data is
 *                  consigned to the server as an array of strings
 *                  Note, the first element (0) contains the number of elements
 *                      also
 *                  Note that this class is abstract and is implemented as either
 *                  PersistanceXML or persistanceCSV
 *
 * Written by Adrian Collins
 *
 * RELEASE - Sept 25th 2001
 *
 */
import java.util.*;
import PersistStream;
import ExceptionLog;
import java.util.StringTokenizer;
import java.sql.*;
import XMLManager;
import DBConnection;
import RunParms;
import QueryHandler;

public abstract class TablePersistance
{
    protected PersistStream   m_Stream;
    protected ExceptionLog     m_uLog        = null;
    protected String m_FileName, m_URL = null;
    protected XMLManager  m_schema = null;
    protected DBConnection  m_DBConnection = null;
    protected ResultSet         m_rs          = null;
    private QueryHandler  m_QueryHandler;
    private RunParms  m_runParms;
    private final static String kDBDSN      = "dqm_ss7";
    private final static String kUID        = "sa";
    private final static String kPWD        = null;
    private final static String kTHIS       = "TablePersistance";
    String filePos = null; // This field is used by all variants and is fetched as part of super

/**
 *        Constructor - Constructs an empty row
 *
 *
 *        Throws              - IllegalArguementException if String does not represent
 *                                  valid attribute
 *
 *        Returns             - n/a
 */
public TablePersistance(ExceptionLog a_uLog, RunParms  a_runParms) throws IllegalArgumentException
        {
        m_uLog = a_uLog;
        m_runParms = a_runParms;
        try
        {

          // eventually, this class will have to provide the
          // url of the schema to the XML Manager class
          m_uLog.LogMessage(kTHIS, "Posting the schema to XMLManager");
          m_DBConnection  = new DBConnection(m_uLog, kDBDSN, kUID, kPWD);
          getPersistLoc();
          m_QueryHandler  = new QueryHandler(m_uLog, a_runParms);
          m_schema = new XMLManager (m_uLog, a_runParms);
        }
        catch(Exception e)
        {
            m_uLog.LogException(kTHIS,"TablePersistance issues...");
            return;
        }
         }
/**
 *        Method              - Creates a unique filename prefix in the form of:
 *                  ddmmyyhhmmss, this prefix will be used when
 *                  creating the persisted file, with the appropriate
 *                  suffix.
```

```
*
*
*          Arguments          - none
*
*          Throws             - Exceeption
*
*          Returns            - uf - the unique filename prefix
*/
  protected String CreateUniqueFileName() throws Exception
  {
      String uf=null;

          java.util.Calendar rn = java.util.Calendar.getInstance();
          java.util.TimeZone tz = java.util.TimeZone.getTimeZone("GMT");
          rn.setTimeZone(tz);
          java.util.Date currentDate = rn.getTime();
          long t = currentDate.getTime();

          //Uncomment this line if default locale set to GMT
          //long t = currentDate.getTime()+(9*60*60*1000);

          java.sql.Date sqldt = new java.sql.Date(t);
          java.sql.Time sqltm = new java.sql.Time(t);

          String sqld = sqldt.toString();
          java.util.StringTokenizer st = new java.util.StringTokenizer(sqld,"-");
          String yer = st.nextToken();
          String mnt = st.nextToken();
          String day = st.nextToken();

      String sqlt = sqltm.toString();
      java.util.StringTokenizer st2 = new java.util.StringTokenizer(sqlt,":");
          String hrs = st2.nextToken();
          String min = st2.nextToken();
          String sec = st2.nextToken();

      uf = day + mnt + yer + hrs + min + sec;

          return uf;
  }
/**
*          Method             - Retrieves the appropriate schema URL for the desired extraction table
*                 returns the URL as a string
*                 We need to get two pieces of information
*                 the host details and the schema file name
*
*
*          Arguements         - a table name
*
*  Throws                     - Exception to notify caller functions that it failed (handled above - non-fatal error)
*
*          Returns            - String
*/
          private  String getURL(String a_SQL)
          {
             String     query = "http://", query1 = null;

      query1 = "SELECT schema_url FROM schema_control WHERE schema_tablename= " + "'" + a_SQL + "'";
      try
      {

         m_rs =  m_DBConnection.runSQLSelect(query1);
         // assume only a singleton exists
         if (m_rs.next())
         {
            query = query + m_runParms.getParameterValue("host") + m_rs.getString("schema_url");
         }

      }
      catch(Exception e)
      {
          m_uLog.LogMessage(kTHIS,"URL could not be located...");
          m_uLog.LogException(kTHIS,"URL could not be located...");
          System.exit(1);
      }
      return query;
```

```
        }
/**
*       Method              - Retrieves the physical file location for the extracted
*             data to be persisted for this host
*
*
*       Arguements          - a table name
*
* Throws                    - Exception to notify caller functions that it failed (handled above - non-fatal error)
*
*       Returns             - String
*/
        private void getPersistLoc()
        {
          String    query = null;

      query = "SELECT host_file_location FROM weld_host WHERE host_address= " + "'" +
      m_runParms.getParameterValue("host")+ "'";
      try
      {

        m_rs =  m_DBConnection.runSQLSelect(query);
        // assume only a singleton exists
        if (m_rs.next())
        {
          filePos = m_rs.getString("host_file_location");
        }

      }
      catch(Exception e)
      {
         m_uLog.LogMessage(kTHIS,"URL could not be located...");
         m_uLog.LogException(kTHIS,"URL could not be located...");
         System.exit(1);
      }
        }
/**
*       execute the supplied procedure
*
*
*
*       Returns             result set
*/
        public void execProc()
        {
       try
       {
          String script = null;
          script = m_QueryHandler.getTableQuery();
          m_rs =  m_DBConnection.runSQLSelect(script);
          m_uLog.LogMessage(kTHIS, "Executing sql script");
       }
       catch (Exception e)
       {
          m_uLog.LogException(e.toString(),"DB procedure could not be processed...");
       }
    }
/**
*       initialise for a new column to be prepared and sent
*
*
*
*       Returns             - n/a
*/
        public void Initialise()
        {
       String s_URL = null;
       try
       {
          s_URL = m_QueryHandler.getTableName();
          m_URL = getURL(s_URL);
          m_schema.getSchema(m_URL);
       }
       catch (Exception e)
       {
          m_uLog.LogException(e.toString(),"Data Base Meta Data could not be located...");
```

```
            System.exit(1);
        }

    }

/**
*       pass back the filename
*
*
*
*       Returns             - file name as a string
*/
        public String getFileName()
        {
      return m_FileName;
        }

        public void Cleanup()
        {
       try
       {
          m_DBConnection.Close();
          m_Stream.Close();
          // rename the file after completion of the extraction
          // the user may now download the file
          if (!( m_Stream.renameFile()))
          {
             throw new Exception();
          }
          m_rs.close();
       }
       catch (Exception e)
       {
           m_uLog.LogException(e.toString(),"Data Base connection could not be satisfactorily closed...");
           return;
       }

    }
/**
*       output the data in nominated format
*
*
*
*       Returns             - n/a
*/
        public void streamOutPut()
        {

    }
}
//EOF TablePersistance
```

```
/**********************************************************************************************************************
 *
 * class TablePersistanceCSV - Class accepts the resultset from an extractor class and
 * generates a csv file, this file name is then converted to a url, which is fed back to the user
 * to allow them to determine the next action.
 *
 * Written by Adrian Collins
 *
 * RELEASE - Sept 25th 2001
 *
 */
public final class PersistanceCSV extends TablePersistance
{

   private final static String kTHIS       = "PersistanceCSV";

/**
 *          Constructor - Constructs an empty row
 *
 *
 *          Throws              - IllegalArguementException if String does not represent
 *                                      valid attribute
 *
 *          Returns             - n/a
 */
          public PersistanceCSV(ExceptionLog a_uLog, RunParms  runParms) throws
IllegalArgumentException
          {
         super(a_uLog, runParms);

          try
          {
             String ls_physical = null;
             String ls_Rename = null;
             m_FileName = CreateUniqueFileName();
             m_FileName = m_FileName + ".csv";
             ls_physical = filePos + "tmp_" + m_FileName;
             ls_Rename = filePos + m_FileName;
             m_Stream = new PersistStream(ls_physical);
             m_Stream.passInLog(a_uLog);
             m_Stream.passInFileName(ls_Rename);
             // filePos is resolved in super and tmp_ is used to
             // stop the user from starting the download until file creation
             // has been completed. On completion, the tmp_ is removed
          }
          catch(Exception e)
          {
              m_uLog.LogException("Terminal Error","Unique Filename could not be created...");
              System.exit(1);
          }

          }


/**
 *          output the data in csv format within the nominated file
 *
 *
 *
 *          Returns             - n/a
 */
          public void streamOutPut()
          {
         String resultContents = null;
         try
         {
            // output the header first
            String ls_colNames [] = m_schema.getColumnNames();
            int colCount = m_schema.getColumnCount();
            resultContents = ls_colNames[0];
            for (int i = 1; i < colCount ; i++)
               {
                  resultContents += ", " + ls_colNames[i];
               }
            m_Stream.WriteCSVData(resultContents);
            while (m_rs.next())
```

```
            {
               resultContents = null;
               // get the first, then loop through each of the others
               resultContents = m_rs.getString(1);
               for (int i = 2; i < colCount ; i++)
               {
                  resultContents += ", " + m_rs.getString(i);
               }
               m_Stream.WriteCSVData(resultContents);
            }
            // rename the file after completion of the extraction
            // the user may now download the file

            if (!( m_Stream.renameFile()))
            {
               throw new Exception();
            }


         }
         catch (Exception e)
         {
            m_uLog.LogException(e.toString(),"Data could not be persisted...");
            return;
         }

      }


}
//EOF persistanceCSV
```

```
/*****************************************************************************************************************
 *
 * class TablePersistanceXML - Class accepts the resultset from an extractor class and
 * generates a csv file, this file name is then converted to a url, which is fed back to the user
 * to allow them to determine the next action.
 *
 * Written by Adrian Collins
 *
 * RELEASE - Sept 25th 2001
 *
 */
public final class PersistanceXML extends TablePersistance
{

   private final static String kTHIS       = "PersistanceXML";

/**
 *          Constructor - Constructs an empty row
 *
 *
 *          Throws              - IllegalArguementException if String does not represent
 *                                      valid attribute
 *
 *          Returns             - n/a
 */
          public PersistanceXML(ExceptionLog a_uLog, RunParms  runParms) throws
IllegalArgumentException
          {
         super(a_uLog, runParms);

         try
         {
            String ls_physical = null;
            String ls_Rename = null;
            m_FileName = CreateUniqueFileName();
            m_FileName = m_FileName + ".xml";
            ls_physical = filePos + "tmp_" + m_FileName;
            ls_Rename = filePos + m_FileName;
            // filePos is resolved in super and tmp_ is used to
            // stop the user from starting the download until file creation
            // has been completed. On completion, the tmp_ is removed
            m_Stream = new PersistStream(ls_physical);
            m_Stream.passInLog(a_uLog);
            m_Stream.passInFileName(ls_Rename);
         }
         catch(Exception e)
         {
            m_uLog.LogException("Terminal Error","Unique Filename could not be created...");
            System.exit(1);
         }

          }


/**
 *          output the data in csv format within the nominated file
 *
 *
 *
 *          Returns             - n/a
 */
          public void streamOutPut()
          {
         String resultContents = null;
         String ls_column_codes [] = null;
         try
         {
            // output the schema name first
            resultContents = "<";
            resultContents = resultContents + m_schema.getSchemaName() + ">";
            m_Stream.WriteXMLData(resultContents);
            // output the schema URL next
            resultContents = "<schema>";
            resultContents = resultContents + m_schema.getSchemaURL() + "</schema>";
            m_Stream.WriteXMLData(resultContents);
            int colCount = m_schema.getColumnCount();
```

```
        // get the coulumn codes array
        ls_column_codes = new String[colCount];
        ls_column_codes = m_schema.getColumnCodes();

        while (m_rs.next())
        {
           resultContents = "<row>";
           int i = 0;
        // loop through each row of the resultset and match with the
        corresponding xml column code ie.
        <C1>xx</C1><C2>yy</C2>...

           for (i = 0; i < colCount ; i++)
           {
                 resultContents = resultContents + "<" + ls_column_codes [i] + ">" + m_rs.getString(i+1) + "</" +
                 ls_column_codes [i] + ">";
           }
           resultContents = resultContents + "</row>";
           m_Stream.WriteXMLData(resultContents);
        }
        // output the top level end-schema tag
        resultContents = "</";
        resultContents = resultContents + m_schema.getSchemaName() + ">";
        m_Stream.WriteXMLData(resultContents);
     }
     catch (Exception e)
     {
        m_uLog.LogException(e.toString(),"Data could not be persisted...");
        System.exit(1);
     }

  }

}
//EOF persistanceXML
```

```
/*****************************************************************************************************************
 *
 * class XMLManager  - This class reads the xsd file and builds a set of collections
 *   which host the properties of the table. Column names, data type, Length of char and varcjar size etc.
 *   The class is instanciated at both client and server, and plays a different role when used by each requestor.
 *   on construct, the class builds a series of collections which describe the physical attributes of the data (a table)
 *   Collections include, Column names. data types of column. a result set which carries the data requested by the
user
 *
 *
 * Loading the Schema: Three components of the schema are mandatory:
 * a)  The Namespace, W3C conformity
 * b)  Element entity describing the table ddl, which enables remote data bases to create a table to
 *     to house the data
 * c)  Element entity describing the table attributes, supported attributes are:
 *     name Attribute Name;
 *     title Used for Human Interface;
 *     code Used to delimiter tags for the columns in xml format;
 *     type data type for converter classes
 *
 * Loading the XML file: The input data file is first opened in this class to
 *     provide information to the class which is needed internally, ie. the
 *     url of the xsd file. The stream is constructed within this class then
 *     passed by reference to the parent instance ie. Class UnloadXML
 *
 *  Written by Adrian Collins
 *
 *  RELEASE - Sept 25th 2001
 *
 */
import java.io.*;
import java.util.*;
import java.sql.*;
import java.net.*;
import XMLReader;
import RunParms;
import ElementProperties;
import ExceptionLog;

public class XMLManager
{

   private ElementProperties m_element = null;
   private ExceptionLog      m_uLog    = null;
    private Vector   v_elements = null;
   private String m_nameSpace = null;
   private String m_fileLocation = null;
   private RunParms  m_runParms;
   private XMLReader  m_xmlReader = null;
   static final byte kFIRST_VALID_CHAR = (byte) '!';
   static final byte kLAST_VALID_CHAR = (byte) '}';
   static final String kTOKEN     = "<";
   static final String kSCHEMA    = "schema";
   static final String kELEMENT   = "element name";
   static final String kATTRIBUTE  = "attribute name";
   static final String kEND       = "</schema>";
   static final String kTHIS     = "XMLManager";


/**
 *        Constructor
 *
 *
 *        Throws              - IllegalArguementException if String does not represent
 *                                    valid attribute
 *
 *        Returns             - n/a
 */
public XMLManager(ExceptionLog a_uLog, RunParms a_runParms) throws IllegalArgumentException
        {
        m_uLog      = a_uLog;
        m_runParms = a_runParms;
        v_elements = new Vector();

        }
/**
```

```
*       Build an xml reader class to open the xml file and get the
*   url for the schema. This method is required by the client
*   when processing the transported data
*
*
*
*       Returns             - n/a
*/

    public synchronized void buildXMLReader ()
    {
      try
      {
        m_xmlReader = new XMLReader(m_uLog);
      }
      catch(Exception e)
          {
           m_uLog.LogException(e.toString()," Could not build XML reader" );
           System.exit(1);
      }
    }
/**
*       Get the schema, and build a tree of the schema
*
*
*
*       Returns             - n/a
*/

    public synchronized void getSchema ()
    {
      URL  m_xsd = null;
      URLConnection m_xsdConn = null;
      BufferedReader xsd_in = null;
       try
      {

        m_xsd = new URL(getSchemaSource());
        m_xsdConn = m_xsd.openConnection();
        xsd_in = new BufferedReader(new InputStreamReader(m_xsdConn.getInputStream()));
        m_uLog.LogMessage(kTHIS," Loading Schema" );
        loadSchema(xsd_in);
        xsd_in.close();
        m_uLog.LogMessage(kTHIS," Closing buffered reader" );

      }
      catch(Exception e)
      {
                m_uLog.LogException(e.toString()," Could not load URL to open Buffered Reader" );
                System.exit(1);
      }
    }
```

```
/**
*       Get the schema, and build a tree of the schema
*
*
*
*
*       Returns             - n/a
*/

    private synchronized void loadSchema(BufferedReader xsd_in)
    {
       String ls_Line = null;
       try
       {
           if (xsd_in.ready())
           {
              ls_Line = xsd_in.readLine();
              // find schema name space first, protocol demands it will be in the first row
              if (SearchWord(kSCHEMA, ls_Line))
              {
                 loadNameSpace(ls_Line);
              }
              if (locateElements(xsd_in))
              {
                 m_uLog.LogMessage(kTHIS," Finished load of elements with No Errors" );
              }
              else
              {
                 m_uLog.LogMessage(kTHIS," Finished load of elements with some issues" );
              }
           }
           // dump the contents of the schema to the log
           // do this if logging has been set to 'true'

          /* if (m_uLog.getLogStatus())
           {
              getElementNames();
              getColumnCodes();
              getColumnNames();
           }
          */
       }
       catch(Exception e)
            {
             m_uLog.LogException(e.toString()," loadSchema, could not execute" );
             System.exit(1);
       }
    }

    private synchronized void loadNameSpace(String a_line)
    {
       m_uLog.LogMessage(kTHIS + " loadNameSpace ", "Begin");

    }

/**
*       Method              - Searches the string for a specific word, and
*                                   The word must not contain any whitespace characters
*
*       Arguements          - word - the word to search for
*
*       Throws              - IOException
*
*       Returns   - the last position of the word in the string, OR 0 if not found.
*/
       private synchronized boolean locateElements(BufferedReader xsd_in)
       {
      m_uLog.LogMessage(kTHIS + " locateElements ",  "Begin");
      String ls_Line = null;
      boolean b_Running = true;
          try
      {
         while (b_Running)
                  {
            ls_Line = xsd_in.readLine();

            // find the element name' tags and then load the appropriate attributes
```

```
            // The line is used to search for attributes, so if we don't find an
            // attribute, it may be a new element, otherwise it may be an error
            // or end of file, so we again check for schema/>

            if (SearchWord(kELEMENT, ls_Line))
            {
                loadElement(ls_Line, xsd_in);
            }

            if (xsd_in.ready())
            {
               ls_Line = xsd_in.readLine();

               if (SearchWord(kEND, ls_Line))
               {
                  b_Running = false;
               }
            }
         }
      }
          catch(Exception e)
          {
         m_uLog.LogMessage(kTHIS," locateElements, ended with issues");
         return false;
          }
      return true;
         }
/**
 *       Method              - Searches the string for "element name" key word
 *
 *       Arguements          - word - the word to search for
 *
 *       Throws              - IOException
 *
 *       Returns   - the last position of the word in the string, OR 0 if not found.
 */
         private synchronized void loadElement(String ls_Line, BufferedReader xsd_in)
         {
      m_uLog.LogMessage(kTHIS + " loadElement ",  "Begin");

      try
      {
         StringTokenizer st = new StringTokenizer(ls_Line,"\"<=");
         while (st.hasMoreTokens())
         {
            String a = st.nextToken();
                  // we know that the string 'element name' is buried in this string
            // so if the token = name, the next token is the one we want
            if (a.equals("element name"))
            {
                String b = st.nextToken();
               m_element = new ElementProperties(b);
               xsd_in.mark(3072);
               ls_Line = xsd_in.readLine();
               while (SearchWord(kATTRIBUTE, ls_Line))
                        {
                     m_element.loadAttribute(ls_Line);
                     ls_Line = xsd_in.readLine();
               }
               xsd_in.reset();
               // add the elements to the vector;
               v_elements.addElement(m_element);
            }
         }
         v_elements.trimToSize();
      }
          catch(Exception e)
          {
         m_uLog.LogMessage(kTHIS," loadElements, could not execute; kill thread");
         System.exit(1);
          }
         }

/**
 *       Method              - Searches the string for a specific word, and
 *                                   The word must not contain any whitespace characters
```

```
*
*       Arguements      - word - the word to search for
*
*       Throws          - IOException
*
*       Returns   - the last position of the word in the string, OR 0 if not found.
*/
    private synchronized boolean SearchWord(String as_word, String ls_Line)
    {
int li_LinePointer = 0;
int li_LineLength = 0;
int  li_length, li_offset = 0;

String word = as_word;
li_length = word.length();
li_offset = word.length();
li_LineLength = ls_Line.length();
String nextWord = null;
   boolean b_Running = true;
              try
              {
      while (b_Running)
                 {
         if (li_LineLength == (li_LinePointer + li_length))
         {
            b_Running = false;
            return false;
                       }

                    nextWord = ls_Line.substring(li_LinePointer, li_offset);
            if (nextWord.equals(word))
                       {
            b_Running = false;
            return true;
                       }
                       li_LinePointer++;
         li_offset++;
      }
              }
              catch(Exception e)
              {
       m_uLog.LogMessage(kTHIS," SearchWord, could not execute");
       return false;
              }
return true;
   }
```

```
/**
 *        Get the table element names
 *
 *
 *
 *        Returns              - string Array
 */
    public synchronized void getElementNames ()
    {
      ElementProperties l_element;

      m_uLog.LogMessage(kTHIS + " Number of Elements = " , v_elements.size());
      for(int i=0;i<v_elements.size();i++)
      {
       l_element = (ElementProperties)v_elements.elementAt(i);
       m_uLog.LogMessage(kTHIS + " Element Name = " + Integer.toString(i) + " contains: ",
       l_element.getName());
      }
    }
/**
 *        Get the name of the schems
 *
 *
 *
 *        Returns              - string
 */
    public synchronized String getSchemaName ()
    {
      ElementProperties l_element;
      // the name of the schema is stored in the 2nd element vector

      l_element = (ElementProperties)v_elements.elementAt(1);
      return l_element.getName();
    }

/**
 *        Get the URL of the schema
 *
 *
 *
 *        Returns              - string
 */
    public synchronized String getSchemaURL ()
    {
      ElementProperties l_element;
      // the name of the schema URL is stored in the 1st element vector

      l_element = (ElementProperties)v_elements.elementAt(0);
      return l_element.getSchemaURL();
    }
/**
 *        Get the schema name from the xml file
 *
 *
 *
 *        Returns              - string
 */
    public synchronized BufferedReader getXMLStream ()
    {
      BufferedReader m_in = null;
      try
      {
        m_in = m_xmlReader.getStream();
      }
      catch(Exception e)
      {
          m_uLog.LogMessage(kTHIS," Could not locate xml Input Stream, mandatory data - ends");
          System.exit(1);
      }
      return m_in;
    }
/**
 *        Get the schema name from the xml file
 *
 *
 *
```

```
*       Returns             - string
*/
    public synchronized String getSchemaSource ()
    {
      try
      {
        m_fileLocation = m_runParms.getParameterValue("xmllocation");
        m_fileLocation += m_runParms.getParameterValue("filename");
        if ( m_fileLocation == null)
        {
          throw new Exception();
        }
        m_xmlReader.getxmlFromFile(m_fileLocation);
        m_fileLocation = m_xmlReader.getxsdURL();
      }
      catch(Exception e)
      {
         m_uLog.LogMessage(kTHIS," Could not determine the Host to contact, mandatory data - ends");
        System.exit(1);
      }
      return m_fileLocation;
    }
/**
*       Get the column count
*
*
*
*       Returns             - int
*/
    public synchronized int getColumnCount()
    {
      ElementProperties l_element;
      // the number of columns is stored in the 1st element vector

      l_element = (ElementProperties)v_elements.elementAt(0);
      return l_element.getColumnCount();
    }
/**
*       Get the table column names
*
*
*
*       Returns             - string Array
*/
    public synchronized String [] getColumnNames ()
    {
      // assume the columns we need are in the second vector element
      ElementProperties l_element;
      l_element = (ElementProperties) v_elements.elementAt(1);

      return l_element.getColumnNames();
    }
```

```
/**
 *      Get the table data types
 *
 *
 *
 *      Returns         - string Array
 */

    public synchronized String [] getDataTypes ()
    {
      ElementProperties l_element;
     // assume the types we need are in the second vector element

        l_element = (ElementProperties)v_elements.elementAt(1);
        return  l_element.getColumnTypes();

    }
/**
 *      Get the table script
 *
 *
 *
 *      Returns         - string Array
 */

    public synchronized String getScript ()
    {
      ElementProperties l_element;
     // assume the codes we need are in the first vector element

        l_element = (ElementProperties)v_elements.elementAt(0);
        return  l_element.getScript();

    }
/**
 *      Get the table column codes
 *
 *
 *
 *      Returns         - string Array
 */

    public synchronized String [] getColumnCodes ()
    {
      ElementProperties l_element;
     // assume the codes we need are in the second vector element

        l_element = (ElementProperties)v_elements.elementAt(1);
        return  l_element.getCode();

    }
/**
 *      clean up
 *
 *
 *
 *      Returns         - n/a
 */
        public synchronized void Cleanup()
        {
     m_xmlReader.CleanUp();
    }
/**
 *      output a row to the log to indicate commencement
 *
 *
 *
 *      Returns         - n/a
 */
        public synchronized String getlogCommence()
        {
     Calendar rightNow = Calendar.getInstance();
     java.util.Date thisDate = rightNow.getTime();
     String ls_now = thisDate.toString();

     String filePath = m_runParms.getParameterValue("xmllocation");
```

```
        String fileName = m_runParms.getParameterValue("filename");

        String query = "INSERT INTO dqm_log (fileName, filePath, fileCreated, fileStatus) values (" +  "'" + fileName
        +  "'" +  "," +  "'" + filePath +  "'" + "," +  "'" + ls_now +  "'" + "," +  "'" + "Commenced" +  "'" + ")";
        return query;

    }

/**
 *          output a row to the log to indicate commencement
 *
 *
 *
 *          Returns             - n/a
 */
          public synchronized String getlogComplete(int ai_rows)
          {
        // ai_row count for the log
        String filePath = m_runParms.getParameterValue("xmllocation");
        String fileName = m_runParms.getParameterValue("filename");

        String query = "Update dqm_log set fileStatus = " + "'" + "Complete" + "'" + ", tableName = " + "'" +
        getSchemaName() + "'" + ", fileRowCount = " + ai_rows +  " where fileName = "
        + "'" + fileName + "'" + " and  filePath = " + "'" + filePath + "'";
        return query;

    }

}
//EOF XMLManager
```

```
/**********************************************************************************************************************
 *
 * class XMLReader  - The input data file is first opened in this class to
 *     provide information to the XMLManager class which is needed internally, ie. the
 *     url of the xsd file. The stream is constructed within this class then
 *     passed by reference to the caller.
 *
 * Written by Adrian Collins
 *
 * RELEASE - Sept 25th 2001
 *
 */
import java.io.*;
import java.net.*;
import ExceptionLog;

public class XMLReader
{

   private ExceptionLog      m_uLog        = null;
   private String m_nameSpace = null;
   private String m_XSDName = null;
   private BufferedReader xml_in = null;
   private  URLConnection m_xmlConn = null;
   static final String kTOKEN     = "<";
   static final String kSLASH     = "/";
   static final String kSCHEMA     = "<schema>";
   static final String kEND        = "</schema>";
   static final String kTHIS     = "XMLReader";

/**
 *          Constructor
 *
 *
 *          Throws              - IllegalArguementException if String does not represent
 *                                      valid attribute
 *
 *          Returns             - n/a
 */
          public XMLReader(ExceptionLog a_uLog) throws IllegalArgumentException
          {
        m_uLog      = a_uLog;
    }

    public String getxsdURL()
         {
        return m_nameSpace;
    }

    public String getxsdName()
         {
        return m_XSDName;
    }
    public BufferedReader getStream()
         {
        return xml_in;
    }
```

```
/**
 *        Get the xml file and read the url of the xsd file
 *
 *
 *
 *
 *        Returns              - n/a
 */

    public void getxml (String a_URL)
    {
      URL  m_xml = null;
       try
      {

        m_uLog.LogMessage(kTHIS," URL = " + a_URL );
        m_xml = new URL(a_URL);

        m_xmlConn = m_xml.openConnection();
        xml_in = new BufferedReader(new InputStreamReader(m_xmlConn.getInputStream()));

        m_uLog.LogMessage(kTHIS," Loading Schema" );

        locateSchema(xml_in);

      }
      catch(Exception e)
           {

       m_uLog.LogException(e.toString()," Could not load URL to open Buffered Reader" );
           System.exit(1);
      }
    }
/**
 *        Get the xml file and read the url of the xsd file
 *
 *
 *
 *
 *        Returns              - n/a
 */

    public void getxmlFromFile (String a_File)
    {
      File   m_xmlFile = new File(a_File);  // listening directory
       try
      {
        URL  m_xml = m_xmlFile.toURL();

        m_xmlConn = m_xml.openConnection();
        xml_in = new BufferedReader(new InputStreamReader(m_xmlConn.getInputStream()));

        m_uLog.LogMessage(kTHIS," Loading Schema" );

        locateSchema(xml_in);

      }
      catch(Exception e)
          {
           m_uLog.LogException(e.toString()," Could not load URL to open Buffered Reader" );
          System.exit(1);
      }
    }
```

```
/**
 *        Get the schema, and build a tree of the schema
 *
 *
 *
 *
 *        Returns              - n/a
 */

    private void locateSchema(BufferedReader xml_in)
    {
       String ls_Line = null;
       try
       {
           if (xml_in.ready())
           {
              ls_Line = xml_in.readLine();
              ls_Line = xml_in.readLine();
              // find schema name space first, protocol demands it will be in the second row
              if (SearchWord(kSCHEMA, ls_Line))
              {
                 loadXSDName(ls_Line);
              }
              else
              {
                 m_uLog.LogMessage(kTHIS," Finished load of elements with some issues" );
              }
           }
       }
       catch(Exception e)
            {
        m_uLog.LogMessage(kTHIS," locateSchema, could not execute" );
             m_uLog.LogException(e.toString()," locateSchema, could not execute" );
            System.exit(1);
       }
    }

    private void loadXSDName(String a_line)
    {
     m_uLog.LogMessage(kTHIS + " loadSchema XSD Name ",  "Begin");
     //<schema>http://192.168.1.254/xml/welddes.xsd</schema>
     String word = null;
     int li_startPointer = 0;
     int li_endPointer = 0;

     li_startPointer = a_line.indexOf(kSCHEMA);
     li_startPointer = li_startPointer + kSCHEMA.length();
     li_endPointer = a_line.indexOf(kEND);

     m_nameSpace = a_line.substring(li_startPointer, li_endPointer);

     // now get the name of the xsd file, in case the url name cannot
     // be located, we can try (as a last resort) to read it from
     // the localhost
     int li_end = 0;
     int li_length = m_nameSpace.length();
     for (int i = li_length; i > 0; i--)
     {
        li_end = i-1;
        if (m_nameSpace.substring(li_end,i).equals(kSLASH))
        {
           m_XSDName = m_nameSpace.substring(i, li_length);
           break;
        }
     }


    }
```

```
/**
 *      Method          - Searches the string for a specific word, and
 *                                The word must not contain any whitespace characters
 *      Arguements      - word - the word to search for
 *      Throws          - IOException
 *
 *      Returns   - the last position of the word in the string, OR 0 if not found.
 */
        private boolean SearchWord(String as_word, String ls_Line)
        {
      int li_LinePointer = 0;
      int li_LineLength = 0;
      int  li_length, li_offset = 0;

      String word = as_word;
      li_length = word.length();
      li_offset = word.length();
      li_LineLength = ls_Line.length();
      String nextWord = null;
        boolean b_Running = true;
                try
                {
          while (b_Running)
                {
            if (li_LineLength == (li_LinePointer + li_length))
            {
              b_Running = false;
              return false;
                        }

                    nextWord = ls_Line.substring(li_LinePointer, li_offset);
              if (nextWord.equals(word))
                        {
              b_Running = false;
              return true;
                        }
                        li_LinePointer++;
            li_offset++;
          }
                }
                catch(Exception e)
                {
           m_uLog.LogMessage(kTHIS," SearchWord, could not execute");
           return false;
                }
      return true;
        }
/**
 *      Method          - Tidies up
 *
 *      Arguements      - none
 *
 * Throws               - Exception
 *
 *      Returns         - none
 */
        public void CleanUp()
        {
          try
          {
         xml_in.close();
      }
          catch(Exception e)
          {
            m_uLog.LogException(e.toString()," Could not close url connection" );
          }
        }
}
//EOF XMLReader
```

APPENDIX B

B.1 DATABASE SCRIPTS

```
/****** Object:  Table [dbo].[Weld_Host] ******/
CREATE TABLE [dbo].[weld_host] (
    [host_address] [varchar] (50) NOT NULL ,
    [host_name] [varchar] (50) NULL ,
    [host_url] [varchar] (50) NULL ,
    [host_insertion_only] [char] (1) NULL ,
    [host_odbc_dsn] [varchar] (50) NULL ,
    [host_schema_location] [varchar] (50) NULL ,
    [host_file_location] [varchar] (50) NULL
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Weld_Description] ******/
CREATE TABLE [dbo].[Weld_Description] (
    [Welder_id] [int] NULL ,
    [Weld_ID] [int] NOT NULL ,
    [Batch] [int] NULL ,
    [Weld_Date] [datetime] NULL ,
    [Travel_Speed_Set_Point] [int] NULL ,
    [Travel_Speed_Tolerance] [int] NULL ,
    [Voltage_Set_Point] [float] NULL ,
    [Voltage_Tolerance] [int] NULL ,
    [Current_Set_Point] [int] NULL ,
    [Current_Tolerance] [int] NULL ,
    [Wire_Feed_Rate_Set_Point] [float] NULL ,
    [Wire_Feed_Rate_Tolerance] [float] NULL ,
    [Heat_Input_Set_Point] [float] NULL ,
    [Heat_Input_Tolerance] [float] NULL ,
    [Deposition_Set_Point] [float] NULL ,
    [Deposition_Tolerance] [float] NULL ,
    [Time_Set_Point] [int] NULL ,
    [Time_Tolerance] [int] NULL ,
    [TOTAL_VOLT_F] [int] NULL ,
    [TOTAL_CURR_F] [int] NULL ,
    [TOTAL_WFR_F] [int] NULL ,
    [TOTAL_HEAT_F] [int] NULL ,
    [TOTAL_DEP_F] [int] NULL ,
    [TOTAL_TIME_F] [int] NULL ,
    [Client_Detail_File] [nvarchar] (50) NULL ,
    [Client_Detail_File_Requested] [smallint] NULL ,
    [Client_Detail_File_Here] [smallint] NULL ,
    [checksum] [int] NULL
```

```
) ON [PRIMARY]
GO
```

```
/****** Object:  Table [dbo].[Abstract] ******/
CREATE TABLE [dbo].[Abstract] (
    [Welder_id] [int] NULL ,
    [Weld_ID] [int] NULL ,
    [Batch] [int] NULL ,
    [Weld_Time] [datetime] NULL ,
    [Part_No] [int] NULL ,
    [Weld_No] [int] NULL ,
    [VOLT_F] [smallint] NULL ,
    [CURR_F] [smallint] NULL ,
    [WFR_F] [smallint] NULL ,
    [HEAT_F] [smallint] NULL ,
    [DEP_F] [int] NULL ,
    [TIME_F] [int] NULL ,
    [checksum] [int] NULL
) ON [PRIMARY]
GO
```

```
/****** Object:  Table [dbo].[Detail]    ******/
CREATE TABLE [dbo].[Detail] (
    [Welder_id] [int] NULL ,
    [Weld_ID] [int] NULL ,
    [Batch] [int] NULL ,
    [Weld_Time] [datetime] NULL ,
    [Part_No] [int] NULL ,
    [Weld_No] [int] NULL ,
    [Section] [int] NULL ,
    [VOLT] [float] NULL ,
    [STD_V] [float] NULL ,
    [Weld_CURRENT] [float] NULL ,
    [STD_CURR] [float] NULL ,
    [WFR] [float] NULL ,
    [STD_WFR] [float] NULL ,
    [HEAT] [float] NULL ,
    [DEP] [float] NULL ,
    [WTIME] [int] NULL ,
    [VOLT_F] [int] NULL ,
    [CURR_F] [int] NULL ,
    [WFR_F] [int] NULL ,
    [HEAT_F] [int] NULL ,
    [DEP_F] [int] NULL ,
    [TIME_F] [int] NULL ,
    [Weld_end] [int] NULL ,
    [checksum] [int] NULL
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[server_status]  ******/
CREATE TABLE [dbo].[server_status] (
    [stop_flag] [int] NULL
) ON [PRIMARY]
GO

/****** Object:  Table [dbo].[table_properties]  ******/
CREATE TABLE [dbo].[table_properties] (
    [table_name] [varchar] (25) NOT NULL ,
    [create_string] [varchar] (1000) NULL
) ON [PRIMARY]
GO
```

```
/****** Object:  Table [dbo].[welder_info]    ******/
CREATE TABLE [dbo].[welder_info] (
    [welder_id] [int] NOT NULL ,
    [welder_name] [varchar] (100) NULL ,
    [default_time_period] [int] NULL ,
    [location] [varchar] (100) NULL ,
    [default_selection_no] [int] NULL ,
    [archive_status] [int] NULL ,
    [transporter_status] [int] NULL
) ON [PRIMARY]
GO
```

APPENDIX C

C.1 USER MANUAL

The DDM Framework Application has been developed to enable the transfer of data from a secure Source Host to a Target Host. Data may be extracted from a nominated database using a Web Browser to locate and select the required data (assuming you have access rights to that data). The extracted data can be then be transported to the requesting users desktop, or to a nominated Relational Database. There are two currently supported formats, XML and CSV, XML is also used to package the data for transport, when a user requests insertion into a distributed (remote) database.

When the user wishes to extract the data and have it delivered to the desktop, the only requirement is knowing how to request the data, however, when the user wishes to have the data inserted into a nominated database (not necessarily that users host machine), the Data Insertion components of the framework need to be implemented on the target host machine.

C.2 SYSTEM REQUIREMENTS

The system was developed using Servlet Technology (refer to Chapter 2). Servlets are modules that run inside request/response-oriented servers, such as Web Servers, and extend them by allowing a user to call Java Objects, by referencing them in a URL (Uniform Resource Locator refer to definition section). Once the URL is accepted, the request is then passed through to the Servlet Engine by the Web Server. In order to use the framework, both Server (Source Host) and Client (Target Host) sides need the following technology components in place.

WEB SERVER

Which accomodates either a Servlet in-process web server plug-in, or an out-of-process web Application Server.

Application Server

The Servlet in-process web server plug-in provides best performance and is the easiest to install and configure, but if a failure does occur, the Web Server application can also become corrupted, necessitating a system restart. The out-of-process application server provides the best architectural flexibility and safety because of Servlet process isolation, allowing the Application Server to be stopped and started independently of the Web Server.

Servlet Engine

A servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually via an HTTP request. If the system is running Microsoft IIS or the Netscape Enterprise Server, you can typically use an in-process Servlet engine, however, the Apache Web Server does not support an in-process Servlet engine.

For development of the framework, the 'ServletExec' Application Server was used,

(Servlet Engine) Unify which is currently available from

http://shop.unify.com/download

and can be purchased for $600 US per server. Unify offer developers an unlimited trial version for free, which is constrained to three (3) concurrent client requests. The framework uses this trial version (currently version 3.0 is implemented); it works extremely well, is easy to install and provides a number of tools for development and debugging. The Engine automatically initialises and creates a Java VM instance when the web server starts, this means that ServletExec is always running and available when the web server is running.

Relational Database (RDBMS)

Any ansii standard database which can be supported by an ODBC connection. The currently implemented framework application is based around an SQL Server 6.5 database Engine. Early versions were implemented over a Microsoft Access Database, however, the code needed to be modified to include 'braces' around the string fields when executing dynamic SQL strings.

ODBC

The Microsoft® Open Database Connectivity (ODBC) interface is a C programming language interface that makes it possible for applications to access data from a variety of database management systems (DBMSs). The ODBC interface permits maximum interoperability by allowing an application to access data in diverse DBMSs through a single interface. Furthermore, that application will be independent of any DBMS from which it accesses data. Proprietory data base Vendors publish software drivers, which provide an interface between an application and a specific DBMS. In order to use the framework, an ODBC Data Source Name (DSN) which points to the underlying RDBMS.

C.3 SYSTEM ADMINISTRATION

There are a number of setup tasks which need to be completed prior to successfully using the system.

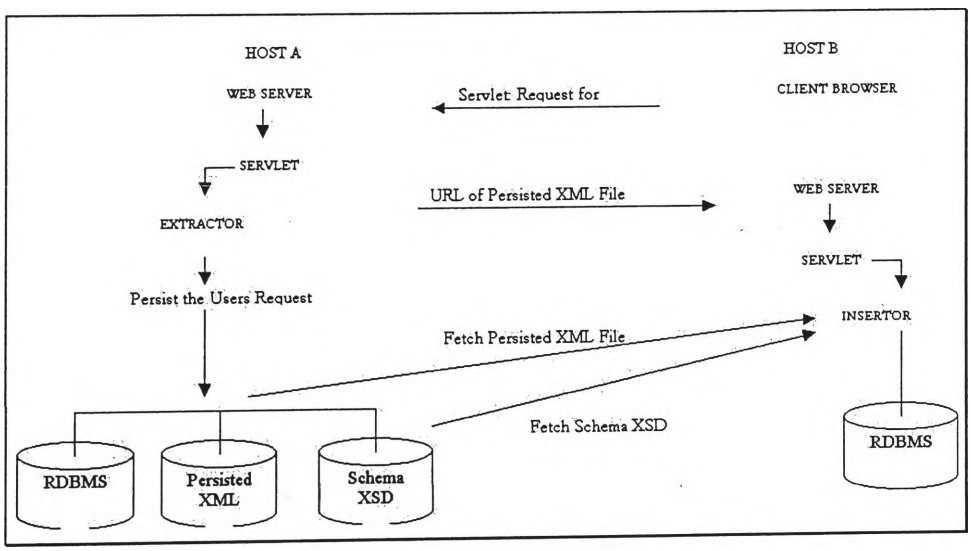

Figure C.1 Architecture Diagram

WEB SERVER AVAILABILITY

The system assumes that the data is accessible using a Web Browser. Request parameters are passed into the framework via HTML and all User access with the system is via the Web Browser. Data can be transported to machines on a subnet, ie. with no external access to the Internet as long as a Web Server is available at the source machine. The target machine may only receive the extracted data in CSV (Comma Separated Variable) format without itself, needing to support a Web Server. However, if the User wishes to use the framework to first extract data, then post that data into an RDBMS, then a number of additional steps need to be completed

prior to activating the system. Most importantly, a Web Server must be implemented on the target machine to initiate the Servlet engine which interacts with the target database.

C.4 COMPONENT SOFTWARE INSTALLATION

The installation process documented below assumes the system will be fully utilised to extract data from a nominated Source Relational Database and insert data into a nominated Target Relational Database. In order to accommodate this processing facility, the following components need to be implemented on ALL hosts which will share the data.

UNIFY INSTALLATION

The complete installation documentation is included in the ServletExec download from http://shop.unify.com/download. ServletExec (from Unify eWave) is a Servlet Engine that implements the Java™ Servlet API 2.2 and JavaServer Pages (JSP) 1.1 standards defined by Sun Microsystems, Inc. as component technologies of the Java 2 Platform, Enterprise Edition (J2EE™). Refer to http://www.unify.com/products/index.htm. The main issue is to install the Servlet Engine, then update the Web Server Properties tab for ISAPI Filters to reference

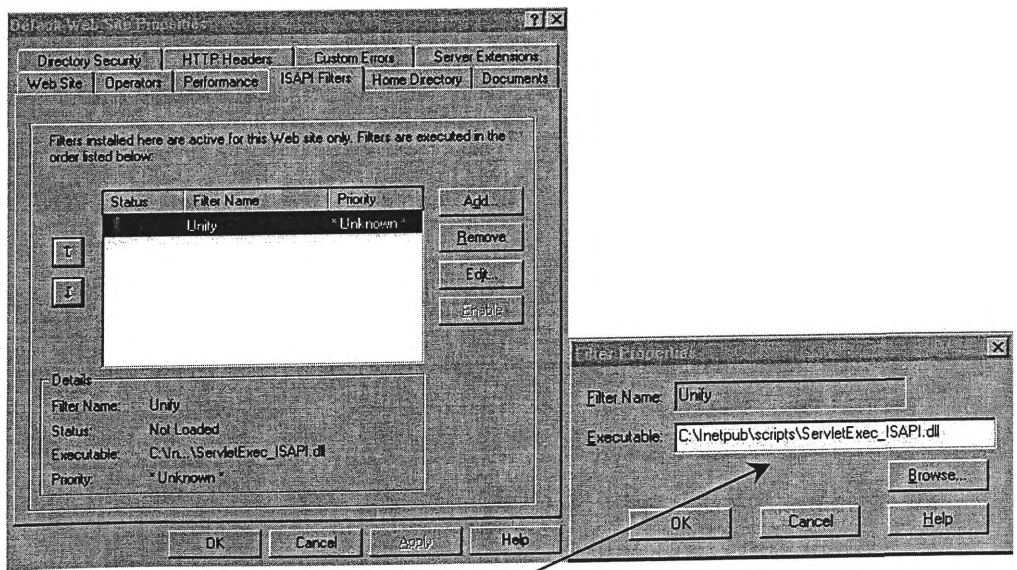

Figure C.2 Web Server Configuration

the DLL plugin for the engine.

Once the Web Server has been stopped, then restarted, the Status flag (refer to figure 5.2) in the ISAPI (Internet Service Application Program Interface) Filter tab should now show as green, indicating that the plug in has been successful. The

Unify configuration process which allows the administrator to setup ServletExec is accessible from the administration user interface at the following URL

http://127.0.0.1/servlet/admin assuming you are installing the ServletExec on the local machine (Replace the local host address if the engine is being installed on a remote host). If the software has been installed successfully, the screen shown in figure 5.2 will be presented. Note the 'classpath' entry on the menu frame which should be set to installDirectory\Unify\ServletExec_ISAPI\servlets, this can be changed to suit the directory where the administrator wishes to deploy the framework classes. The Data Extraction process is initiated by a java class named servletExtractor, which (for example) may be physically located at d:\ProgramFiles\Unify\ServletExec_ISAPI\servlets. To test the installation, send the following request from the client browser with the following URL http://127.0.0.1/servlet/DateServlet If all is well, the Web Server should return a screen with the following content:

Wed Oct 10 13:00:09 GMT+11:00 2001

If not follow the 'trouble shooting' and help documentation in the ServletExec_30_User_Guide.pdf included in the download package. The install is very straightforward and generally, should provide no problem.

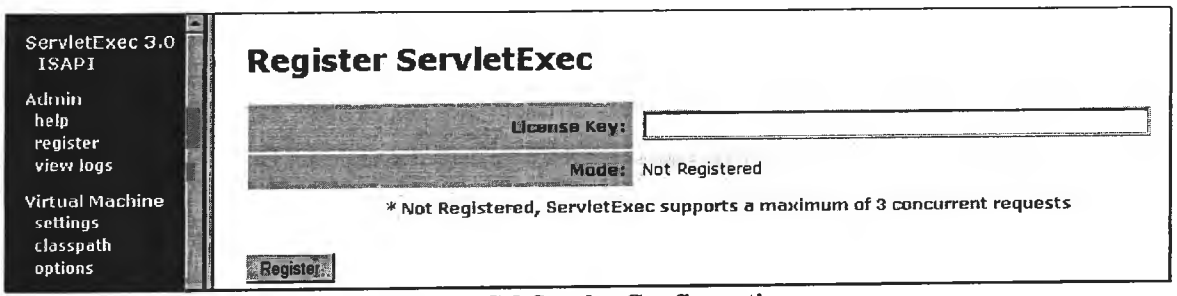

Figure C.3 Servlet Configuration

Additional documentation ServletExec 3.0 Installation Guide is located at:

http://support.unify.com/documents/servletexec/SE3_IG.pdf

In order to view the Installation Guide or the User Guide, you'll need Acrobat Reader which is available (as a free download) at:

http://www.adobe.com/products/acrobat/readstep.html

If you have still have problems installing ServletExec then check the ServletExec technical support FAQ:

http://support.unify.com/supportal/ServletExecFAQ.htm

as well as free installation support, available via email to support@unify.com

Note, that you must indicate the version of JDK/JRE and the web server (and version) being usedJava installation.

DATABASE SETUP

The current version of the framework is installed using Microsoft® SQL Server Version 6.5. As mentioned earlier, any ansii standard database which provides an ODBC driver is an acceptable repository for data storage. SQL Server was chosen because of it's availablility, cost and the fact that Stored Procedures and Triggers are supported. While there is no current use of Stored Procedures, the framework will operate with result set's returned by Stored Procedure. (Refer to Method/prototype/client side model/DBConnection). Once SQL Server has been installed, the database needs to be created with the following properties:

Name	DQM
Table Size	50 Megabbytes
Log Size	10 Megabytes

SYSTEM TABLES

There are a number of system tables which need to be generated, these tables provide dynamic data to the framework , and facilitate the creation of XML data, and the dynamic generation of Schema files (XSD).

The system and data tables can be created using the scripts provided in Appendix/Database Scripts.

SCHEMA_CONTROL: The framework uses a schema to prepare the XML data (as well as decipher the packaged data). When the framework executes the Client SQL string, the schema_control table is accessed to provide the name of the schema file (xsd) to use when building the XML data. Refer to figure 5.4, the name of the table is used as a key to match the correct xsd file, and the location. The detail provided by the request is also included as referencing schema URL in the XML document.

schema_tablename	schema_url
weld_description	/xml/welddesc.xsd
Abstract	/xml/abstract.xsd
Detail	/xml/detail.xsd

WELD_HOST: The framework uses data from the weld_host table as runtime parameters to allow the application to be deployed using the specific site administration rules. Each host must have an IP address which equates to the request.getServerName() (available from the Servlet request), this ip address is used to:

- Identify the local ODBC DSN name associated with the framework
- Nominate the location of the XML document generated by the framework.

host_address	host_odbc_dsn	host_file_location
192.168.1.254	dqm_target	d:/dqm/development/ServletData/

Note this folder must be made publicly (refer to figure 5.6) available as a URL, and be available as a subordinate directory under the Hosts published Web directory. In the example setup shown in figure 5.5, remote hosts will be told to locate the XML document at http://192.168.1.254/ServletData/

ODBC SETUP

Refer to figure 5.5 above, which identifies the odbc dsn which is used by the framework to access the underlying database. Once the database has been installed, the next step is to build the DSN. Bring up the ODBC Data Source Administrator panel (figure 5.7), from the panel, select <System DSN>, then <ADD> a new DSN. Select the appropriate database driver (in the example case it is SQL Server).

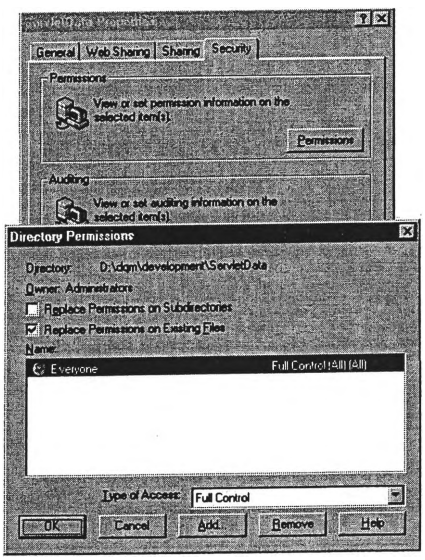

Figure C.4 Permissions

Provide a name and description for the service (note this name must then me inserted in the weld_host table in the host_odbc_dsn column, using the appropriate host_address as the key.

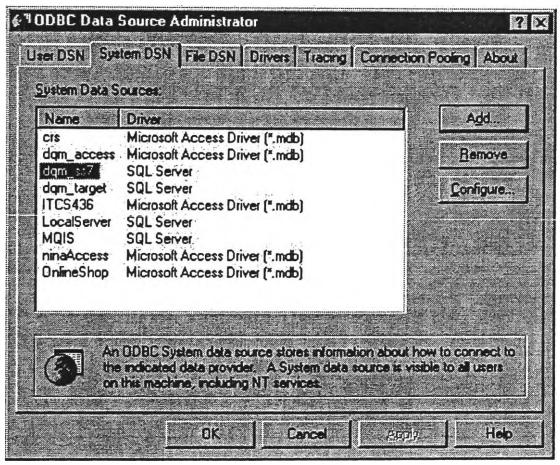

Figure C.5 ODBC Setup

From the create DSN screen wizard, select the server which matches the IP Address entered into host_address, then provide related security information. As a prototype, the framework uses the standard "sa" User account and null password. If you wish to use a more rigorous level of security, the kUID and kPWD constants must be modified in the DBConnection module (then recompiled), and the related accounts setup in the underlying database. While this is not a complicated procedure, it is considered outside the scope of the User manual, and is fully documented in the Method/Security_and_Data_Access section. In any event, once the security details are entered (refer to 5.8) ensure that the client configuration option is set to TCP/IP (figure 5.9).

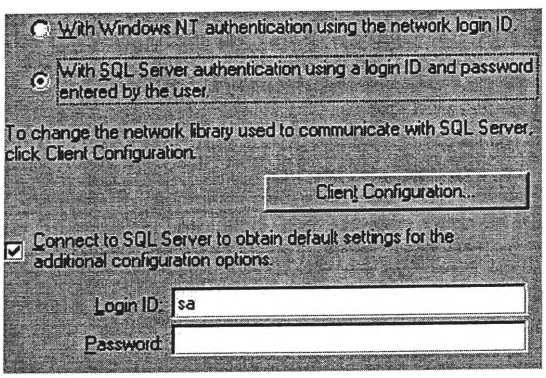

Figure C.6 SQL Server Configuration

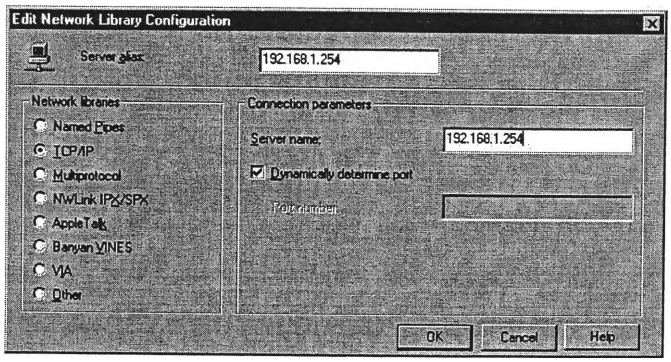

Figure C.7 SQL Server Configuration

All other setup parameters are self explanatory, however, do carry out a test to ensure the ODBC service can reach the target database.

WEB SERVER CONFIGURATION

This section relates to the setup of the Web Server and reiterates that the directories underneath the Home directory must be flagged as public (figure 5.6). There are two directories which must be created in order for the framework to operate. First, the schema directory, which is user defined. In the example shown in figure 5.4, the schema directory is nominated as /xml/ which means that a remote host would be presented with the http://192.168.1.254/xml/abstract.xsd URL when requesting the schema URL from the XML document being processed.

Secondly, the framework needs a public directory to maintain the XML documents being produced as a result of extraction requests. These requests generate an XML document which is placed in the directory identified as **host_file_location** in figure 5.5. the entry is **d:/dqm/development/ServletData/**, so, the Web Server Home directory is **d:/dqm/development**, which means that **ServletData** is publicly accessible when provided as a URL, and framework accessible from within the extraction application.

CLIENT CONFIGURATION

If the Client Host will be used to INSERT data extracted from a Remote Host into a locally maintained RDBMS, then the client machine will need to be setup as per the Software Installation guidelines. Obviously, if the client host is only receiving data and is not itself an extraction host, then the database entriy for weld_host (which identifies the name of the ODBC DSN) is the only requirement. Security is also an issue if the site is not using the standard system security for Database access. As mentioned earlier, if a more rigorous level of security is required, the **kUID** and **kPWD** constants must be modified in the DBConnection module (then recompiled),

and the related accounts setup in the underlying database (fully documented in the Method/Security_and_Data_Access section).

C.5 USING THE FRAMEWORK

The framework has two setup components which must be implemented and understood, prior to use. The first component (Database Setup), has been covered in the Software Installation section, and facilitates access to the data in the database, and configuring the Web Server to facilitate access to the XML data, once it has been extracted from the database, but is still resident on the server host. The next section covers the request parameters and the variuos methods of placing a request to the extraction component of the framework.

DATA EXTRACTION REQUEST

An extraction request is typically received by way of an HTML page being posted to the Servlet Engine. For example, the following URL contains a request:

http://192.168.1.254/servlet/servletExtractor?Format=XML&Button=Extract+Data&Table=Weld_Description&KeyName+1=welder_id&KeyValue+1=2

This request can be broken down into the following sub components:

URL	http://192.168.1.254/servlet/servletExtractor
Format	XML
Button	Extract+Data
Table	Weld_Description
KeyName+1	welder_id
KeyValue+1	2

Figure C.8 Parameter Interface

All requests to the framework must be in this format, the complete list of allowable parameters is provided.

URL: The mandatory string which names the Servlet directory and the name of the extractor module (located in the Servlet directory), which is launched (as a thread) by the servlet engine.

Format: There are two formats currently supported by the framework, CSV (Comma Separated Variables), and XML (Extensible Markup Language). The request must specify the output format to the extraction module.

Button: This parameter tells the extraction module where to deposit the extraction request. Valid values are transport or extract data, any other value will cause processing to halt.

Table: This parameter refers to the table being queried for data extraction

KeyName 1: This parameter, (note the numericaly appended increment), must match the KEYVALUE appended numeric. The concept is to build a set of matching pairs, that are then used to append to the WHERE clause in the SQL statement used to extract the data from the underlying database. The Java based Servlet technology has no way of matching the parameter names with the respective parameter value. For example if the sql query we are building requires two arguments in the WHERE clause say

```
SELECT * from xxxx WHERE

parm_name_1 = parm_value_1 AND
parm_name_2 = parm_value_2
```

The servlet engine passes the arguments into the request buffer in random order, so the numeric sequence number is a simple method of constraining the interface to accept only pairs named as follows:

KeyName 1 or KeyName_1 must match KeyValue 1 or KeyValue_1

KeyName 2 or KeyName_2 must match KeyValue 2 or KeyValue_2

KeyValue 1: This parameter contains the actual value inserted into the SQL string used to extract the data from the underlying database.

User Interface

Currently, functionality is provided for the user to first view the data, prior to making an extraction choice of transport to the Users desktop in either CSV or XML format, or transport to a relational database of the user choice.

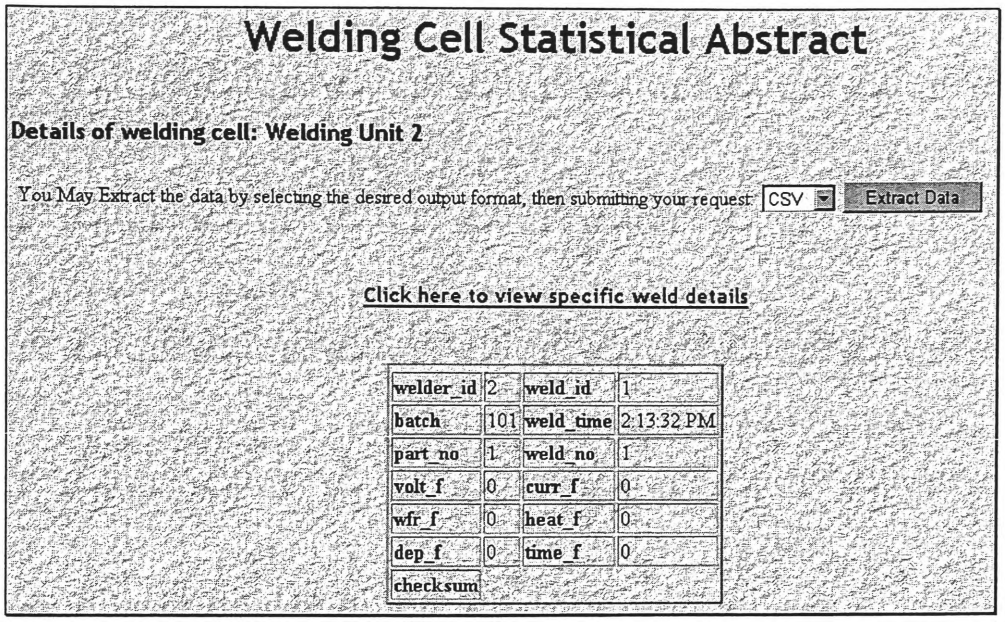

Figure C.9 File Selection

The user can execute ad hoc sql queries which can then be delivered in csv or xml. The HTML interface is presented as a list page with the data shown in table format, the user may then nominate to extract the data and manually download it to the local site.

C.6 ENTERING EXTRACTION PARAMETERS

Extraction parameters must be entered via HTML, which are then passed into the framework for execution. The HTML document presents the parameters as input identifiers via form tags (refer to figure 5.13), which depicts a segment of the mark up used to produce the HTML document in figure 5.11. Note the standard mark up is depicted in a red colour, VBscript in a tangarine colour, parameters in a blue colour and the parameter values in a green colour. Note also the VBscript method of presenting database selections to the user. The parameters are flagged as input within the form, and may be hidden. The action identifier must name servlet/servletExtractor as the URL component submitted when the User selects the button. The parameters are then appended to the end of the URL string and passed into the framework engine.

```
<form name=show_weld action="servlet/servletExtractor" method=GET>
<table>
<tr>
<td align=left>You May:</td>
</tr><tr> <td align=left>Extract the data by selecting the desired output format, then submitting your
request:</td>
<td><select name="Format" size=1  align=center >
                                <option>CSV
                                <option>XML
                                </select>
<input type=submit name="Button" value="Extract Data"></td>  </tr>
<%
        SQLquery = "select * from weld_host "
        set myset = dbConnect.execute(SQLquery)
%>
 <tr><td align=left>Or, transport the file to the desired host: <Transport>:</td>
        <td><select name="Destination" size=1  align=center>
<%
        do while not myset.eof
%>
        <option><%=myset("host_address")%>
<%
        myset.movenext
        loop
%>
</select>
<input type=submit name="Button" value="Transport"></td>
 <input type=hidden name="Table" value ="Weld_Description">
 <input type=hidden name="filelocation" value = "http://192.168.1.254/ServletData/">
  <input type=hidden name="KeyName 1" value = "welder_id">
<input type=hidden name="KeyValue 1" value = <%=WelderNumber%>>
 </tr>
</table>
</form>
```

Figure C.10 Schema

In addition, note the method of identifying the KeyValue 1 parameter, is to use a vbscript variable declared in the markup (extract shown in figure 5.14). Using this method, the developer is then able to dynamically execute the extraction request.

```
<!--Extract the welder number from the passed in QueryString value "Welder_ID" -->
<%
        WelderNumber = Request.QueryString("Welder_ID")
        SQLquery = "select * from Welder_Info where Welder_ID = " & WelderNumber

        dbConnect.open dbSTR
        set ROBOTNAME = dbConnect.execute(SQLquery)

        Welder = ROBOTNAME("Welder_Name")
        numToShow = ROBOTNAME("Default_Selection_No")
%>
```

Figure C.11 VBScript example

C.7 DATA INSERTION

As per the Extraction components of the framework, any host being setup to receive XML for posting into a relational database must be configured in the same manner as documented under paragraph 3, ie. data is received via a Web Server, with a Srevlet engine capturing the dynamic requests from the data source host.

Install the Servlet engine, the database and then create an ODBC connection as documented under paragraph 3.

Ther is no specific setup of the Web Server required, only that the Servlet engine is correctly configured to interact with the Web Server, run the example servlet to confirm the installation of the servlet engine. And also complete the working example when implementing the ODBC connection, to ensure that you have been successful.

The IP address of the Client Host must also be inserted into any Extraction hosts in the framework operational scope. This means an entry into the weld_host table (on any data extraction host), remember, the framework uses data from the weld_host table to provide a 'drop down' listbox of available target hosts. This mode of administration is used to provide more control of the framework operational scope. As an example, if the tartget host is 192.168.1.10, then this information needs to be added into the weld_host table on the extraction host (refer to figure 5.15)

host_address	host_name	host_url	host_odbc_dsn
192.168.1.254	SERVER	http://192.168.1.254	Dqm_ss7
192.168.1.10	CLIENT	http://192.168.1.10	Dqm_ss7

The current version of the framework uses dqm_ss7 as the mandatory name of all ODBC DSN connections.

C.8 SECURITY

There is no application level security in place, as there is no direct (external) access to the Source or Target Database. While it is true that extracted data is placed into a public access area, the only user option is to read the data by transporting it to the users desktop, or transfer the data to another database under program control. The security issue which relates to the modification of data (for malicious purposes), can only be achieved at the insertion end, by altering the contents of the XML data being transported to the target site.

C.9 VIEWING LOGS

The framework system log file contains two types of entries. All genuine exceptions (caught by the system and explicitly logged) are appended to the dqm log file, as well as debugging messages which are appended to the log file by the system maintenance personel. The message log is a debugging tool whichcan be switched

on or off by setting a compile time flag in each of the framework processing components, servletExtractor and servletInsertor. In each module, the Service method contains an entry which allows the developer to set logging on with a m_uLog.setLog(true); statement. The ExceptionLog constructor automatically calls the setLog method and sets message logging to false. So unless the developer calls setLog with a boolean true, logging will remain off.

The log file will be automatically generated as dqm_server.log in the WINNT\system32 directory.

```
12/10/2001, 22:41:18; Message from RunParms <----> Value: 192.168.1.254
12/10/2001, 22:41:18; Message from RunParms <----> begin getParmValue, using: filelocation
12/10/2001, 22:41:18; Message from RunParms <----> Parm: host
12/10/2001, 22:41:18; Message from RunParms <----> Parm: table
12/10/2001, 22:41:18; Message from RunParms <----> Parm: button
12/10/2001, 22:41:18; Message from RunParms <----> Parm: keyvalue 1
12/10/2001, 22:41:18; Message from RunParms <----> Parm: destination
12/10/2001, 22:41:18; Message from RunParms <----> Parm: format
12/10/2001, 22:41:18; Message from RunParms <----> Parm: keyname 1
12/10/2001, 22:41:18; Message from RunParms <----> Parm: filelocation
12/10/2001, 22:41:18; Message from RunParms <----> Value: http://192.168.1.254/servletdata/
12/10/2001, 22:41:18; Message from TransportRequestHandler <----> URL = http://192.168.1.254/servlet/servletInsertor?Destina
14/10/2001, 23:27:55; Message from DataInsertor <----> Posting the schema to XMLManager
14/10/2001, 23:27:55; Message from DBConnection <----> Attempting to connect to database jdbc:odbc:dqm_ss7
14/10/2001, 23:27:56; Message from RunParms <----> begin getParmValue, using: filelocation
14/10/2001, 23:27:56; Message from RunParms <----> Parm: filelocation
14/10/2001, 23:27:56; Message from RunParms <----> Value: http://192.168.1.254/xml/26092001104512.xml
14/10/2001, 23:27:56; Message from XMLReader <----> URL = http://192.168.1.254/xml/26092001104512.xml
14/10/2001, 23:27:56; Message from servletInsertor <----> Contact lost with Thread, mandatory process- ends
14/10/2001, 23:27:56; Message from XMLManager <----> URL = null
```

Figure C.12 Error Log

Figure 5.18 provides an example of the message content which is available when the messaging is enabled.

There is also a Servet log (specific to the Unify Servlet Engine only) which provides an indication of when the engine is started and stopped, and can be useful in the tracking of a problem at a remote site. Note the logs are central to each host, so if the host is supporting both the frameworks servletExtractor and the servletInsertor components, the log will contain entries from both components, separated by the time stamp.

C.10 SYSTEM INFORMATION

The online help system, which is essentially this document converted to the HTML format is provided to cover as many uses as possible. Any developer using the framework, should ensure that this documentation is provided to administrators and users alike

ERROR HANDLING

There are many reasons for the framework failing to successfully extract data or, insert that extracted data into a remote relational database. There are a number of 'co operating' pieces of technology which need to be functioning, in order for the framework to operate as expected. As a means of both debugging and resolving runtime problems, the Framework system log contains two types of entries. All genuine exceptions (caught by the system and explicitly logged) are appended to the dqm log file, as well as this, the system supports the use of the message log as a debugging tool which can be switched on or off. By setting a compile time flag in each of the Framework processing components, servletExtractor and servletInsertor, the application will log specific events as the process runs. This is particularly helpful for the extraction process which is run under the administration of the Web Server, and as such is not able to be directly debugged. In each module, the Service method contains an entry which allows the developer to set logging on with a `m_uLog.setLog(true);` statement. The ExceptionLog constructor automatically calls the setLog method and sets message logging to false. So unless the developer calls setLog with a Boolean `true`, logging will remain off. The log file will be automatically generated as `dqm_server.log` in the `WINNT\system32` directory. Figure 6.4 provides an example of the message content which is available when the messaging is enabled. There is also a Servlet log (specific to the Unify Servlet Engine only) which provides an indication of when the engine is started and stopped, and can be useful in the tracking of a problem at a remote site.

```
12/10/2001, 22:41:18; Message from RunParms <----> Value: 192.168.1.254
12/10/2001, 22:41:18; Message from RunParms <----> begin getParmValue, using: filelocation
12/10/2001, 22:41:18; Message from RunParms <----> Parm: host
12/10/2001, 22:41:18; Message from RunParms <----> Parm: table
12/10/2001, 22:41:18; Message from RunParms <----> Parm: button
12/10/2001, 22:41:18; Message from RunParms <----> Parm: keyvalue_1
12/10/2001, 22:41:18; Message from RunParms <----> Parm: destination
12/10/2001, 22:41:18; Message from RunParms <----> Parm: format
12/10/2001, 22:41:18; Message from RunParms <----> Parm: keyname_1
12/10/2001, 22:41:18; Message from RunParms <----> Parm: filelocation
12/10/2001, 22:41:18; Message from RunParms <----> Value: http://192.168.1.254/servletdata/
12/10/2001, 22:41:18; Message from TransportRequestHandler <----> URL = http://192.168.1.254/servlet/servletInsertor?Destina
14/10/2001, 23:27:55; Message from DataInsertor <----> Posting the schema to XMLManager
14/10/2001, 23:27:55; Message from DBConnection <----> Attempting to connect to database jdbc:odbc:dqm_ss7
14/10/2001, 23:27:56; Message from RunParms <----> begin getParmValue, using: filelocation
14/10/2001, 23:27:56; Message from RunParms <----> Parm: filelocation
14/10/2001, 23:27:56; Message from RunParms <----> Value: http://192.168.1.254/xml/26092001104512.xml
14/10/2001, 23:27:56; Message from XMLReader <----> URL = http://192.168.1.254/xml/26092001104512.xml
14/10/2001, 23:27:56; Message from servletInsertor <----> Contact lost with Thread, mandatory process- ends
14/10/2001, 23:27:56; Message from XMLManager <----> URL = null
```

Figure C.13 Logging

JAVA VM

The java virtual machine must be operating on any host utilising the framework. In order to determine that the VM is operating, initiate the Web Browser and enter the following URL http://127.0.0.1/servlet/DateServlet, this should result in a response from the Server Host with a display of date and time.

ODBC DSN

Refer to section 2.3 of this chapter for details on setting up ODBC and potential problems. The configuration wizard provides a utility to verify the ODBC

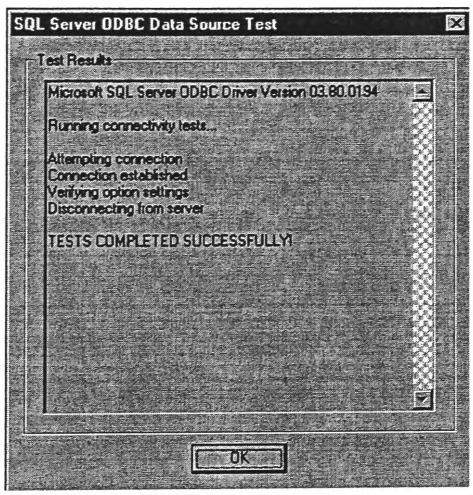

Figure C.14 ODBC DSN

connection for a Data Source Name (DSN) created for an RDBMS.

RELATIONAL DATA BASE MANAGEMENT SYSTEM

The framework has only one direct interface with the underlying database, which is via the weld_host table. Refer to the database setup section for configuration information on this table and the setting up of the framework options. Also refer to the appendix for a build script and information on the physical nature of the table. It should also be understood, that all access to the weld_host table is via the ODBC connection DSN.

WEB SERVER

In order to determine that the local host Web Server is operating, perform the same validation check previous Java VM instructions. Initiate the Web Browser and enter the following URL http://127.0.0.1/servlet/DateServlet, this should result in a response from the Server Host with a display of date and time.

SERVLET ENGINE

In order to determine that the local Host Servlet Engine is operating, perform the same validation check previous Java VM instructions. Initiate the Web Browser and enter the following URL http://127.0.0.1/servlet/DateServlet, this should result in a response from the Server Host with a display of date and time.

OPERATIONAL DEFICIENCIES

There are only two issues which need to be monitored by users of the application, one is the Application Server which manages the Servlet engine, and the other is processing large files.

APPLICATION SERVER

Refer to the section of the appendix relating to the licencing of the Servlet Engine, there are many available, the one used for the pipleine application has changed ownership hands twice during development. The current license allows for 3 concurrent servlet request to be handled concurrently, which is adequate for developemtn and testing, but the Administrator must be aware that if the operational environment requires more concurrent requests to be serviced, a full license may need to be applied for, or a different Servlet model sourced.

APPENDIX D

TECHNOLOGY AND STANDARDS

D.1 INTRODUCTION

With the acceptance of the Internet as a means of increasing market penetration to a global level, even small to medium organisations are now looking to communicate with both their trading partners and customers via the Internet. Inter organisational access must also incorporate a simple set of standards. Developers are now accepting international standards as a mandatory basis for development, particularly Web development. This acceptance is passed on to Stakeholders in the form of Development Environment requirements. Design, implementation and operating Standards are mandatory for any technology centric Web Application which is being put in place. The reason is that all current technology is developed using layered architecture methodologies. The acceptance of standards as an integral part of the construction of these layers has been a long and costly process for the Enterprise.

This appendic covers the various technologies and related standards which have been investigated as part of the application design. Also included is a brief history of the development of Distributed Objects and relevant technology issues.

D.2 REMOTE PROCEDURE CALL

The Remote Procedure Call (RPC) protocol was put forward by Sun in 1988 (rfc1050) and is based on earlier work at XEROX by Birrell and Nelson in 1984. The RPC model is similar to the local procedure call model in that the caller sends data as arguments (by value), or makes the address of arguments available (by reference) to a locally scoped function or procedure. The process then transfers control to the procedure, and eventually gains back control. At that point, the results of the procedure are extracted and the caller continues execution.

The remote procedure call is similar, in that one thread of control logically winds through two processes -- one is the caller's process, the other is a server's process. That is, the caller process sends a call message to the server process and waits (blocks) for a reply message. The call message contains the procedure's parameters, and the reply message contains the procedure's results. Once the reply message is received, the results of the procedure are extracted, and caller's execution is resumed.

On the server side, a process is dormant awaiting the arrival of a call message. When one arrives, the server process extracts the procedure's parameters, computes the results, sends a reply message, and then awaits the next call message. The protocol makes no restrictions on the concurrency model implemented, and others are certainly possible. For example, an implementation may choose to have RPC calls be asynchronous (and not block while waiting for a response), or have the server create a task (fork) to process an incoming request, so that the server can be free to receive other requests. The design goal of the RPC was to abstract the actual connection and allow the client function to make a standard function call and wait for control to return to the calling function. In order to achieve the abstraction, the client module is compiled with additional code (**stub**). The stub is responsible for marshalling (converting the argument/s into text) the clients data for the call into a machine independent format for the transport to the server side (refer to D.1). SUN has produced a standard for machine independent representation of this data called External Data Representation (XDR).

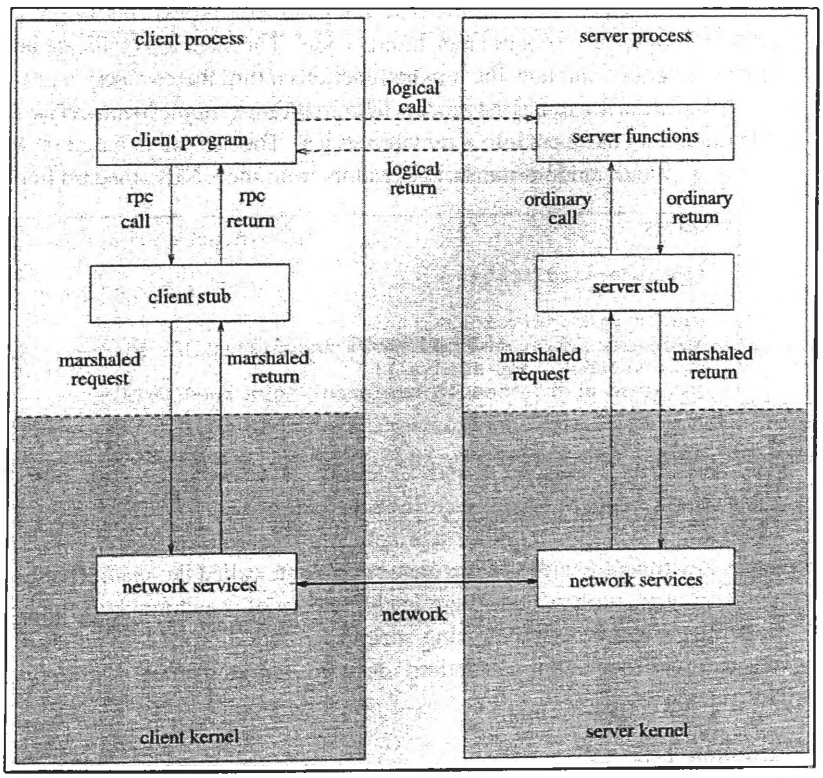

Figure D.1 Remote Procedure Call

The Server side also has a stub which is responsible for receiving the message text from the client process, unmarshalling the arguments and making a standard function call to the desired module. When data is to be returned to the client, the server side stub is also responsible for marshalling the data to be returned. It is interesting to note that the concept of abstraction stubs and the marshalling of data

into a machine independent format is the still followed by the CORBA model. Also of note is the development of semantics which provide developers with a methodology for dealing with failure between the client and the server, ie. a number of possible scenarios are possible:

- The network was slow and the client did not wait long enough for a reply from the server.
- The initial message was lost
- The server received the message, performed the request, then crashed.
- The server performed the service, but the acknowledgement was lost.

In TCP oriented connections, the TCP client acknowledges each receipt packet, the server then retransmits packets for which it did not receive an acknowledgement, in addition, the TCP client manages the order of packets and presents these to the client application in the correct order [ROBBINS96]. The physical data transport issues which are managed through TCP are abstracted from the RPC protocol allowing the developer to concentrate on the application issues of managing the content of the data transmitted between the client and the server. SUN has also provided developers with software to further reduce the possibility of error, by taking responsibility for generating the stub and skeleton software 'jackets' for each end of the RPC. The de facto standard for TCP/IP implementations is the one from the Computer Systems Research Group at the University of California, Berkeley (BSD). This source code has been available since 1983 and is the starting point for many other implementations [STEVENS94]. However, BSD also put forward an API which abstracted the messaging service for client/server applications, and allowed the developer a lot more flexibility than the RPC model. This abstraction is termed Berkeley Sockets, developed by BSD and released with UNIX 4.1c circa 1983).

D.3 SOCKETS

While RPC's explicitly hide the connection details from the developer and present the remote application as a 'black box', the Socket implementation allows the communication process to be driven much more as a functional component of the application program. The developer has many more design choices and options for managing the TCP service within the program scope. Most importantly is the concurrent multitasking options available via the socket implementation. When

coupled with the 'THREAD' (circa 1990) implementation, the concept of concurrency becomes even more important to the distributed application developer. Originally, the socket implementation was provided to allow Interprocess Communication (IPC), but designers very quickly saw the benefit in sitting the socket API over TCP/IP which allowed Interprocess Communication. When the designer incorporates the socket API model into the program module, as opposed to the RPC model, the mechanism of Interprocess Communication can be compared to playing one dimensional tic-tac-toe versus multi dimensional tic-tac-toe. These days, we should consider that the Socket API implicitly enforces the use of threads, 'enforces' may be too strong a word to describe the embodiment of threads within the socket model, but we don't want to bark when we have a dog. The thread is an abstract data type representing the flow of control within a process. A thread has it's own execution stack, program counter, register set and state. It shares with peer threads, it's code section, data section and operating system resources such as open files and other task related components. The major benefit of the thread is that the operating system does not have to swap the thread out in it's entirety at context switching time, since the other peer threads share these resources. A context switch takes on average 10,000 nanoseconds, or, in human terms, if a process cycle takes 1 second, then a context switch takes 180 minutes [ROBBINS96]. It would be unprofessional of the developer not to take advantage of these opportunities to increase the throughput of an application under development.

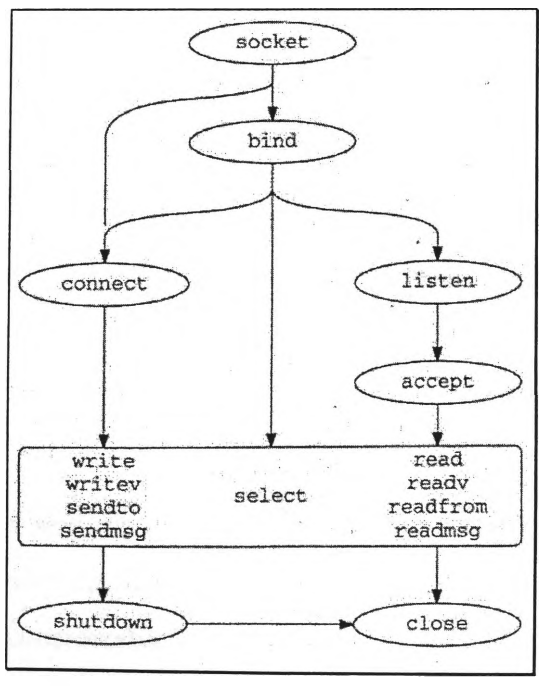

Figure D.2 Socket Architecture

The socket has become the ubiquitous, base level component in the Distributed Model. Telnet, SMTP, FTP , HTTP and most other protocols; all have the socket model built into their architecture. The implementation which we confine ourselves to in this discussion relates to a TCP connection, which uses a STREAM buffer to provide communication between the Client and the Server. Once the Server has bound the process to the desired listening port via the bind() system call, the process then issues an accept() system call to await a connection request from the client, via the connect() system call (refer to figure D.2). Once a connection has taken place, the communication process is facilitated by read() and write() calls to the buffered stream. It should also be noted that support for STREAM devices use the standard SIGIO blocking signal on read() and write() to maintain asynchronous control between the client and the server process. I/O to the buffer can be made synchronous, but for the Framework application under discussion, the initial SOCKET implementation used the asynchronous mode, this was preferable for the type of data transport processing of the project, once handshaking had occurred, the sending process could send it's data, a row at a time, then signal end once all data had been transmitted.

The receiving process could then read each row, convert the ASCII data into an INSERT statement, then post the data through the ODBC connection to the database. The implementation used a COMM class which received the socket as an argument, then instanciated a buffered reader and a buffered writer. The Comm class was instanciated on both client and server and allowed the physical reading and writing to be encapsulated within the Comm class as per:

```
public Comm(Socket s) throws IOException
 {
   fIn  = new BufferedReader(new InputStreamReader (s.getInputStream()));
   fOut = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
 }
```

ISSUES

The process itself was efficient from a transport perspective, given that any method of distributed transport would require a marshalling process to 'flatten' the data into a text stream for transmission over the network. There were a number of other issues which required considerable enhancement of the basic prototype:

- If the data model changed, the insertion modules on each host would need to be modified to reflect the new table layout.
- If the tables did not exist on the receiving host, a create script would need to be sourced and executed prior to data being transported to the remote host.
- The send and receive modules required manual starting, so if the machine was unreachable or required rebooting, there was potential for the socket communication approach to require some administration for reconnection. This is a genuine concern and has enormous implications, since the data gathering Hosts are installed on trucks, in what can only be considered primitive conditions. It is expected that these Hosts will have access to the Internet via wireless technology, with availability subject to atmospheric conditions, climatic conditions and geographical location.
- Security was not a genuine concern, given the application, however, malicious interference or corruption during the transport process was. Data integrity is considered a mandatory requirement, and the standard socket API provided no real means of ensuring the data was received in exactly the same form as it was sent.
- One of the project objectives was to provide an extraction method which ported the data to a distributed Host in Comma Separated Variable format. This facility allows the data to be used for other analytical purposes or in third party applications such as Excel, Lotus Approach etc.

D.4 DISTRIBUTED OBJECTS

The task of transporting the data from one host to another necessitated a review of the requirements of the application to determine the importance of the issues raised above. The trade off with the socket scenario was the reasonable throughput, which may be compromised if an alternative technology was used to transport the data. Any method which would be adopted would need to focus on throughput and data integrity. The use of Distributed Object technology is now fully accepted within the IT industry as a means of sharing data. This is mainly because of the acceptance and availability of the Internet as a transport medium, allowing distributed objects to become much more of a norm over the last 3-4 years. Prior to this, organisations maintained private switched networks and intranets to manage the distributed components running within their Enterprise Applications. By 1995 (in Australia), it

was quite a common business option to provide an Internet gateway for organisations running secured intranet(s). Email was the main application using the service, but it did not take the IT departments of these organisations long to realise that other data communication applications could be vented through the Internet, using public switching networks, rather than costly point to point dial-up, or leased private networks. With regard to the Internet, standards which formalised distributed objects were quite mature, a number of standards bodies were already in place, such as the United Nations Economic Commission for Europe (UN/ECE) and the World Wide Web Consortium (W3C). These standards bodies were able to quickly interface with the major research and development vendors, putting standards in place which are even more relevant today. As technology has progressed with regard to Distributed Objects (now identified as Web Objects), other standards have become necessary, not so much to enforce conformity, but to provide a framework to facilitate the location of services and resources available on the Internet. Enter the Object Modeling Group (OMG), the international standards group responsible for the promotion of CORBA standards, interprocess API's, bindings and usage documentation, and other major players providing standards and API's for Web Objects are the W3C and SUN (Javasoft)

D.4.1 CORBA (Common Object Request Broker Architecture)

"CORBA was designed to allow intelligent components to discover each other and interoperate" [ORFALI98]. The Common Object Request Broker Architecture is the synergy of over 800 organisations, representing the entire spectrum of the computer industry. "The notable exception is Microsoft, which has it's own competing object broker called the Distributed Component Object Model (DCOM)" [ORFALI98]. There are two issues which need to be understood by the user in order to make use of CORBA components, IDL and Binding services.

IDL

CORBA separates the Objects definition from it's implementation by using an Interface Definition Language to declare the data types of the attributes being passed between the Web Objects. Once the interface attributes have been declared, the developer selects a precompiler suitable for the implementation language being used. For example, in the Framework project, the development language is JAVA, so we would use a commercially available ***idl2java*** compiler. The compiler generates RPC like **stub** and **skeleton** class objects in the applicable language, which we can

then incorporate into our design. The use of IDL abstracts the communication process and allows us to call the remote object using a locally declared method.

BINDING

CORBA also abstracts the physical communication between the client and the server by using a technology termed an "Object Request Broker". The ORB provides a repository of registered services to requesting clients and allows those clients to request a session for service. The technology uses a specific Internet Inter-ORB protocol (IIOP) for communication between one ORB and another. When the server component is instanciated it is registered with it's local ORB as a service provider, and when the client is instanciated, it places a request for service with a local ORB. The ORB is a commercially available piece of technology and runs as a daemon on the application network, both client and server components use named library includes to declare the ORB facilities at compile time. Once instanciated, the components locate, then bind the requestor (client) with the service provider (server) to facilitate interprocess communication.

LATER IMPLEMENTATIONS

CORBA is provided as a set of commercially available components, with the developer purchasing the desired IDL compiler to suit the implementation language. However, since the release of JAVA 1.2, SUN has incorporated the CORBA standard into the RMI API, which negates the need for third party components if the application is written in JAVA [ASBURY99]. Since the Framework application is written in JAVA, this allowed me to focus on the merits of RMI as a component inclusion in the distributed framework modules for the application.

D.4.2 RMI (REMOTE METHOD INVOCATION)

As a candidate for use in the final version of the Transport Framework of the Framework Application, Remote Method Invocation (RMI) presented with a lot of advantages, it is Java Centric, so a lot of the code which had already been written for the Socket prototype could be retained. The RMI implementation of JDK 1.2 had most of the CORBA recommendations in place, particularly RMI over IIOP which would:

- Allow the file or sql selection to be processed on the target host as a Transaction (ie. fully committed or fully rolled back).

- Provide support for persistent Object references, so once the host is located, services can be requested on an ad-hoc basis without the need to continually broker the request.
- Provide support for remote Object activation which overcomes the manual restarting of the Objects when the machine is re-started.

Internet Inter Orb Protocol (IIOP) sits on top of TCP/IP and value adds the CORBA message exchanges prior to passing the information to and from the application. In this way, details of service location, and transactional boundaries are abstracted from the user application [ORFALI96].

RMI clients do not interact directly with distributed Objects, but interface via a published interface, as do CORBA clients. Arguments are marshaled via the Java Serialisation service (java.io.ObjectOutputStream/java.io.ObjectInputStream) and passed to the distributed Object via the relevant stream. The distributed Object is bound to the Java Naming Service and is then accessible via the abstracted proxy stub on the client. For example, a class GetTime, on the local host, can call a local function getTime(). This method is called via the proxy getTime(), which uses the Naming.lookup service to locate a remote instance of getTime(), execute the function, then return the results to the local proxy.

```
public interface GetTime extends Remote
{
    String getTime() throws RemoteException;
}
```

On the remote Host, a set of RemoteObject classes is sub classed to create the required objects

```
public class GetTimeImpl extends UnicastRemoteObject implements GetTime
{
    public GetTimeImpl() throws RemoteException {      super(); }
    public String getTime() {      return  "12:00:00"; }
}
```

In the main method of the distributed Object, the object is instanciated, then bound to the RMI Naming Service registry, as per:

```
GetTimeImpl MyObj = new GetTimeImpl ();
```

```
Naming.rebind("//host:port/name", MyObj);
```

In addition to abstracting the Socket layer, RMI also manages the security aspects of the application automatically registering the distributed Objects with the security service running on the distributed Host. The range of options is extensive and allows the developer to set these options in a persisted security file which is accessed and implemented by the Java Virtual Machine running on the distributed Host.

In analysing the usefulness of RMI; it was still an abstract Socket model, however, there are a couple of genuine benefits over raw sockets. First was the remote activation feature which would allow the module to be called by the client when necessary, and the other was the ease with which the RMI client could persist the data once it was received.

In order to make use of the activation feature, the java.rmi.activation package is included in remote Object. In the above example; the class declaration is modified to

```
public class GetTimeImpl extends Activatable implements GetTime
```

instead of

```
public class GetTimeImpl extends UnicastRemoteObject implements GetTime
```

Activatable and UnicastRemoteObject are both sub classes of RemoteObject, and the Activatable implementation informs the rmiregistry running on the distributed Host to load the GetTimeImpl if not already instanciated.

With regard to the persistence facility, the receiving client would only need to instantiate a class which extended the abstract class of RandomAccessFile in order to persist the rows to an ASCII file, eg. **public class ANSIFileStream extends RandomAccessFile.**

D.4.3 EJB (ENTERPRISE JAVA BEANS)

Enterprise JavaBeans is the latest technology abstraction in the Java family, and provides an abstraction for component transaction monitors (CTMs). Component transaction monitors represent convergence of two technologies; transaction processing monitors, such as CICS, TUXEDO and ENCINA, and distributed Object services such as CORBA, DCOM and native Java RMI [HAEFEL00]. EJB is not a technology in it's own right, it's more an aggregation or consolidation of other architectures. The distributed nature of EJB is facilitated by the abstraction of Java's

Remote Method Invocation (RMI) methodology and below that it's standard socket architecture. Enterprise JavaBeans require a Middleware component to manage the relationship of data on a distributed Server process and a User on a remote Client Host. The relationship is managed via an Application Server specifically engineered for transactional processing use by such technologies as Servlets, Dynamic Server Pages, and Enterprise JavaBeans. The Application Server is a third party proprietary product which is typically licensed to the Host machine/s where the Application Server is implemented. The App Server is responsible for service brokering, transaction management, security, persistence and concurrency, thus allowing the designer to concentrate on the Business requirements and the design issues which are unique to the application. The model assumes the App Server requirements are common and can be accommodated in a generic manner. In addition, the App Server can be implemented in a scalable manner to accommodate very large numbers of users and associated transactions with an acceptable response time for the transaction process.

The third party Application Server software provides transaction management and persistance facilities for the distributed application. Transaction management has successfully been administered in distributed applications since IBM released CICS in 1968. The concept is based on providing remote access to data which may involve a number of separate data tables or database entities. The transaction is either committed or rolled back in it's entirety, depending on whether all components of the transaction complete successfully. The technology has been re-engineered to accommodate Object Oriented entities in a three tier architecture. Clients request access to data by activating a session with the middleware component transaction manager (CTM), responsible for the transaction. The component manager, acts as a broker for the client, by accessing the physical database, and retrieving the data for the client. The data is essentially moved from the database and stored in memory as an 'entity bean', with issues such as concurrency, access authorisation, security and integrity, managed by the CTM. The client may update the entity bean by calling public access methods available in the bean, via the CTM. When the CTM recognises that the entity bean is no longer required in memory, it is deactivated and returned to physical data storage within the database (persisted).

While EJB technology is extremely relevant to the process of distributed processing, it was not specifically relevant to the Framework application. The reasons were as follows:

- Transaction management and concurrency issues were not germain to the project.
- Given the point above, the cost of a third party CTM Application Server was not justifiable.
- The data volumes were too high to be efficiently managed by a CTM.
- The system overheads and administration requirements inherent with the CTM were not justifiable, given the application requirements.
- Moving away from the monkey see monkey do approach to installation, there is absolutely no documentation safety net.

D.4.4 XML - (EXTENSIBLE MARKUP LANGUAGE)

BACKGROUND

The word 'Markup' has it's origins in the Printing trade, and relates to the use of special characters, placed around the text, indicating the text should be treated in a manner designated by the special characters. Originally SGML (Standard Generalized Markup Language, ISO 8879:1986(E)) was developed as a Standard in the Printing and Publishing industries. The standard is very complex and was developed over a 15 year period, however, the important positive benefit is that SGML focuses on structure and allows users to develop their own tag conventions for entities and attributes.

In 1991, Tim Berners Lee developed HTML as a variant of SGML, it's purpose was to specifically deal with the movement of document content over the World Wide Web by the academic community. HTML was not as complex or unfriendly as SGML and was more focused on formatting the contents of the document rather than the structure. HTML did come at the right time though for a world just beginning to accept the Internet as an integral part of life. The scripting method was easy to learn and very forgiving in the hands of a novice, this allowed 'ordinary' people to climb the technology mountain and pass information around the web, even host a web site of their own. Technology had come to the people, which was great news for both the business and consumer sectors. Organisations also jumped

on the HTML bandwagon and began to use the Web as a marketing tool, with instant acceptance, notwithstanding the issue of payment over the web. During the first few years of Internet infancy, developers bent and twisted the HTML page to impossible shapes in order to bolt on facilities for dealing with images, dynamic content, personalisation etc. At the same time, organisational Web Sites "grew to include 10,000 pages or more, organised loosely in hierarchical schemes concocted by developers who knew little about hypertext and less about organisation" [STLAURENT98]. Time exposed a number of other weaknesses of HTML, which is where XML comes in. If HTML encounters markup which is not in the standard, it simply treats the markup as text. This allowed Browser vendors to develop proprietary enhancements to the Standard which, while annoying for the User of a competitors browser, did not cause the Browser to fail, just display the non Standard markup in the browser window. The implications of this were serious though, as developers needed to be able to develop markup as necessary for the industry or application which was specific to the task at hand. This led to the acceptance of formatting templates such as Style Sheets and cascading those style sheets to multiple pages in the Site, which at least gave the Site a standard to base their documents on.

With the acceptance of new technology, developers realise that the Browser is only 'one' of the many, diverse interfaces which are being presented to the information 'hip' customer. Telephone, WAP, PDA, Smart White goods are all areas which provide a User interface and need an underlying set of standards to structure and format the content for the User. Enter XML....

It is also noteworthy that the Object Modeling Group (OMG) who are responsible for CORBA have put forward a set of XML Interfaces with Standards for CORBA, EJB and CICS as well as ebXML

XML was developed by the XML Working Group under the auspices of the World Wide Web Consortium (W3C) in 1996. Development of XML was led by Jon Bosak of Sun Microsystems and work began as a direct response to pressure from developers who had two valid issues with HTML. Firstly, they were limited by HTML and it's lack of structure and adherence to standards and secondly, developers refused to accept SGML as a replacement because of it's complexity and verbose specification.

HTML is limited to a fixed set of markup tags which the developer can use, while XML allows users to create their own tags, or use tags created by others, ie. XML facilitates reusability and extensibility. As with SGML, XML can be formatted and validated by a Document Type Definition, which allows the user to declare what constitutes markup with the XML page and also what the markup means within the page. Once the XML parser has done it's work and used the DTD to validate the document, a document tree is created, based on the hierarchical structure declared in the DTD. This document tree may then be made available to the user, or accessed by processing applications. There are a number of issues with the Document Type Definition which causes me to look more closely at using a Schema (refer to Chapter 3) to develop the framework.

There is no doubt that the process of transporting data over the internet has become a trivial task in the mind of the Internet User, the irony is that the delivery of that task will become proportionally more difficult. As data traffic increases, bandwidth usage will become more of a critical issue. One of the benefits of using layered architecture in the Object Oriented realms, is that sooner or later, an interface (API) will become available to abstract the tedious and complex. Sun are developing a Web Browser Class which is a Rowset implementation that can serialise the data, metadata and properties of a JDBC result set to XML, according to Williams [WILLIAMS00], "That way, the result set can be disconnected, transported across the network, and manipulated by a remote application". While there is potential for a considerable amount of data (in the pipeline project) to be shipped during a session, this API appears to map directly to the system requirements.

D.4.5 ebXML - (Electronic Business Extensible Markup Language)

Background

The ebXML initiative is jointly managed by the United Nations (UN/CEFACT) and OASIS. Any organisation may take part and the initiative enjoys broad industry support with over 175 member companies, and in excess of 2,000 participants drawn from over 30 countries.

The ebXML architecture provides a method of:

- Defining business processes with associated messages and content.
- Registration and discovery of business process sequences with related message exchanges.

- Defining company profiles.
- Defining trading partner agreements.
- Providing uniform message transport layer.

The development of ebXML is evolutionary, based on proven technologies and accepted standards (HTTP, TCP/IP, XML etc.), facilitating an open and vendor-neutral solution to eBusiness. As mentioned earlier, until recently, organisations typically exchanged information with each other electronically, based on Electronic Data Interchange (EDI) standards. This required an organsational investment in technical expertise, and necessitated a tightly coupled, Framework architecture. As a result, the use of EDI has been limited to large enterprises only, ebXML aims to remove these limitations by providing a simple, user friendly set of standards which allow any organisation to develop trading links with other organisations via the Internet and the ebXML facilities. Opening the way for low cost electronic business which is both flexible and easy to use. ebXML seeks to standardise the different types of EDI and electronic trading infrastructures that already exist. OASIS is currently developing of a set of technologies built on open standards which cater for messaging, transport routing and packaging (TRP), trading partner agreements (TPA), repositories and registries (REP/REG). The standards will incorporate architecture, business processes and other core components which will be presented as a 'tool box' framework and specifications are expected to be published in May 2001.

According to Robert S. Sutor, vice chair of ebXML (2000) and member of the OASIS Board of Directors. "commercially integrated ebXML-compliant solutions will reduce the costs of deployment and ensure the flexibility required for e-commerce success in the global market".

OPERATIONAL REQUIREMENTS

Organisations must be able to discover each other and the products and services they have to offer. Determine which shared business processes, and associated document exchanges to use for obtaining products or services from each other. Determine the contact points and forms of communication for the exchange of information. Finally, agree on the contractual terms relating to the selected processes and associated information. Once these 'Inter Business Rules' have been established, the organisations can exchange information and services in an 'ongoing' automated

fashion. ebXML provides an infrastructure that ensures data communication interoperability via a standard message transport mechanism with a well defined interface, packaging rules, and a predictable delivery model. The standards also provides a service interface that handles incoming and outgoing messages at either end of the transport. This is achieved by a semantic framework based on XML with a well definde meta model providing the organisation with re-useable business logic and a method of defining message structures and definitions as they relate to the activities defined in the framework. The framework gives organisations a way to find each other, agree to establish business relationships, and conduct business, through a shared repository where businesses can register and discover each other's business services via partner profile information. Once contact has been made, the framework provides a process for defining and agreeing to a formal set of procedures for communication and business.

ARCHITECTURE

The technical architecture is composed of five main area of emphasis:

1. BUSINESS PROCESS AND INFORMATION MODEL

The model defines how business processes are described and enables an organization to express its business processes so that they are understandable by other organizations. The Information model defines reusable components that can be applied in a standard way within a business context that is meaningful to their business while also maintaining interoperability with other business applications.

2. COMPANY PROFILES

A repository is provided (see Registry and Repository) to allows organisations to maintain their own details and query the repository for organisations they may wish to contact for eBusiness.

3. MESSAGING SERVICE

The messaging service specification defines the services, protocols and methodologies required to exchange data. Standard protocols such as SMTP, HTTP, and FTP are supported, as well as application specific technologies which facilitate encryption, digital signature, secure protocols and authentication.

4. REGISTRY & REPOSITORY

The Repository provides facilities for storing and registration of data such as company profiles, trading partner requirement specifications and the registration

interchange requirements. The repository also supports the maintenance of the data, and gives the registered organisations access to templates, business process definitions and core software components.

5. Collaberative Partner Agreements

The repository stores critical information, necessary for communications between applications and business processes and also records specific technical parameters for conducting eBusiness.

Usage Scenario

There are many different scenarios in which ebXML technology may be implemented, activities may be performed in slightly different sequences and with different scope and focus, however, the following example typifies the steps carried out.

An Australian importer (computer hardware) who is looking in Asia, to locate, then do business with a Chinese exporter may carry out the following activities:

- The Importer would looks through the repository for an Information Model which matches his business, ie. Importing computer equipment and register his orgainsation in the repository. Access to the repository provides the importer with the ability to create and maintain appropriate electronic documentation templates for the importation of computer equipment and elctronic advertising Profiles of the orgainsation and supported business processes for exporters in other countries who may be searching for an Australian importer of computer equipment. Access to the repository also give the importer a query facility to locate exporters in Asia who match the importers selection criteria, and create his own companies profile for
- Once the importer has nominated the appropriate business service interface (importation of computer equipment) they may then locate other orgainsations with whom they wish to enter into Collaborative Agreement. Once an agreement has been reached between the importer and another business partner, a formal Collaborative Partner Profile is registered within the repository.

From a technical perspective, this gives both organisations access to Parsers and the appropriate Document Type Definitions related to Computer Hardware. It should also be understood that both organisations would implement suitable application

software which meets the stpecifications of the Messaging Service which facilitates the capture and processing of ebXML traffic. The result is that both importer and exporter send and receive ebXML messages containing internal business documents, customs documents, travel manifests etc, over a secure and reliable ebXML Messaging Service. ebXML is also backward compatible and is designed to work with existing legacy EDI solutions, as well as being used to develop applications based on emerging technologies using XML.

D.5 XML TECHNOLOGY

D.5.1 INTRODUCTION

Use of XML as an alternative to the socket method proved not to be the only benefit for the Framework as:

- It is very likely that the data model will be continually modified as other modules making up the Framework are enhanced, and new modules incorporated. When database table details are changed, XML is easily modified to reflect the changes.
- Internet outages are a regular occurrence in a field based application, XML acts an external storage container for data waiting to be transported to other sites.
- Use of the DOM facilitates the conversion from XML into other data formats.

This section details XML technology, the use of the various methods for building a document object model (DOM), and provides a context for the use of the technology in the Framework application.

D.5.2 CONCEPTUAL OVERVIEW

In 1996, XML development commenced as a response to pressure from developers who had issues with HTML. The major problems centred around the lack of structure, adherence to standards and lack of extensibility. It has been widely documented that both HTML and XML are derived from the same source, ie. SGML (Standardised General Markup Language ISO 8879). The obvious difference between the two markup applications is that HTML is focused on the presentation of the content within the browser, while XML uses a meta language, allowing the developer to prioritise and provide application specific structure to the content. XML content is not limited to browsers, as is the case with HTML, the document may be accessed by processing applications and used in any manner appropriate to the application.

There are two separate processing steps which are required when accessing an XML document. If the structure of the XML document is invalid when the results are passed to another process, the invalid data will NOT be processed, and may cause the processing application to become unstable or worse, affect the Web Server process. Firstly, parsing is required to ensure that the document is 'well formed', this is a basic validation condition which means that tag statements must open and close in logical order (refer to figure D.3). Data content is then made available via a predefined access template called the document object model (DOM), which facilitates hierarchical access to the data. The DOM concept means the creator of the document can define TAG statements which are meaningful to the application, and not limited to a fixed set of tags, such as this HTML example:

<H3 ALIGN=Center> Heading Line </H3> indicating that the tag is a level 3 Heading and should be centrally aligned. When using XML, developers may choose tags which relate to the content in a more meaningful way.

```
<weld_header>
    <welder_id>
            <title>Welder Id</title><type>Integer</type><value>2</value>
    </welder_id>
    <weld_id>
            <title>Weld Id</title><type>Integer</type><value>2</value>
    </weld_id>
    <batch>
            <title>Batch</title><type>Integer</type><value>911</value>
    </batch>
</weld_header>
```

The above example describes a database table, <weld_header> with each column making up the table described as subordinate attributes <welder_id>, ie. the TAGS contain information about the accompanying content, which gives the developer much more flexibility and extensibility in the way the content is marked up. In practical terms, developers can adopt a set of tags which reflect the application, thus making the raw XML more meaningful to the Human reader. The content is not clogged with presentation syntax, increasing throughput, and providing additional flexibility. As well as describing the content tags, additional structure, content constraints and validation can be achieved via a document definition which may either be encapsulated within the XML document, or externally referenced by the

XML document. The Document Definition (DTD) associated with the XML document also provides many other benefits to the developer:

- Documents may be generated programmatically, both from scratch or from existing documents.
- Document content can be modified programmatically
- Document content can be read and filtered programmatically
- Documents provide a storage and shipping container for the data content

THE METHOD

In order to be considered 'well formed', the document must contain a root element which is closed in a valid manner, in the above example, this is the <weld_header>; the document must contain at least one element, this may be the <batch> element, or the <weler_id>; in addition, if there are other elements contained within the document, they must be nested with no overlap between elements. An example of an invalid document would be:

```
<weld_header>
    <welder_id>
            <title>Welder Id</title><type>Integer</type><value>2</value>
            <weld_id>
                    <title>Weld Id</title><type>Integer</type><value>2</value>
    </welder_id>
            </weld_id>
    <batch>
            <title>Batch</title><type>Integer</type><value>911</value>
    </batch>
</weld_header>
```

which would present the following error message from the Parser:

```
End tag 'welder_id' does not match the start tag 'weld_id'. Line 6,
Position 4

</welder_id>
---^
```

The reason for insisting on a well formed document is that the XML document may be passed to a co-operating application, and with no human intervention, the

receiving application cannot successfully process the document unless it is at least well formed.

Document Parsing

If the XML document is not successfully validated, the parser does not build a document tree and the receiving application is not passed the contents of the XML document. When a Document Type Definition (DTD) is declared, and is associated with the XML document, the Parser also confirms that the document follows the structure laid out by the DTD. However, it should be understood that the XML document only needs to be 'well formed' in order for the Parser to build a treD. Figure D.3 shows the result of a parsing operation, which builds a binary tree of the contents of the XML document.

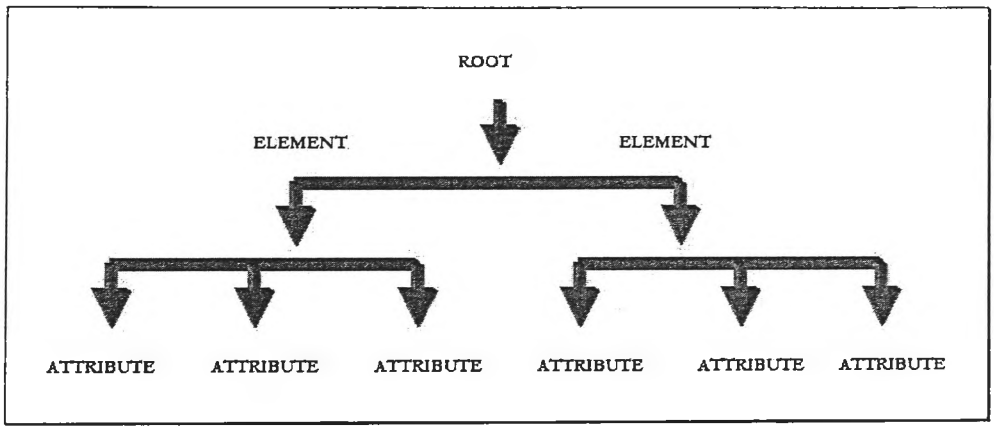

Figure D.3 Binary Tree (DOM)

In the above example (figure D.3), the root of the document would equate to the table name tag <weld_header>, elements would equate to table columns <welder_id>, and attributes would equate to the descriptive attributes which relate to the table columns <type>Integer</type>.

When present, the DTD contains markup declarations that provide a grammatical syntax for the document parser, which may be encapsulated within the users browser or applied to the XML document as part of a receiving application. If present, the DTD can be placed inside the XML document or can be externally referenced from within the document. If the DTD is internal, it must appear before the first element in the document.

Presentation in Human Readable Form

Neither the XML document or the DTD contain any instruction for presenting the data in human readable form, ie. displayed within a browser. However, the XML

model specifies a number of alternatives for presentation. The developer may use Style Sheets written in Cascading Style Sheet (CSS) syntax or another XML application language, Extensible Stylesheet Language (XSL) or, use the Style Sheet to convert the XML into HTML.

PRESENTATION TO A RECEIVING APPLICATION

When XML data is to be presented to a client Application, the developer has a number of options for processing the data, which must conform to the XML specification by being at least 'well formed', ie. syntactically correct:

- The document may either be validated by a third party parser
- The developer may choose to build an application specific parser with rules and constraints which are more appropriate to the underlying application requirements.
- The developer may create his own parser and related DOM (using an application programming interface), and read in the raw XML to generate an application specific in memory treD. This can be useful if there is something specific the developer wishes to do, such as, reduce the amount of memory required to build the DOM tree or provide application specific methods for reading the DOM.

A number of third party parsers are freely available, including Microsoft (MsXML) and IBM (Xerces) who are both extremely committed to XML technology. Parsers can be implemented as external services (which are passed the document to be parsed), or can be encapsulated, as with major (Internet Explorer 5.0 and Netscape 6.0) browsers and at a minimum allow the developer to validate a document.

PARSER OUTPUT

From a developer's perspective, the parser forms a bridge between the document and the processing application. The parser is responsible for handling XML syntax and, if desired, checking the contents of the document against constraints established in a document type definition (DTD) or Schema (refer to D.5), leaving the application free to deal with the content and not be concerned with validity issues. There are two basic kinds of XML parsers defined in the XML specification. Nonvalidating parsers, which simply check document syntax and report all violations of well-formedness. The other type is a Validating parser which performs

the same function as a nonvalidating parser, but also compares the structure of the document to rules declared in the DTD.

The scope and nature of the application determines whether to use a validating or nonvalidating modD. Building 'in house', end to end type components whose scope is lies within the same application probably means that a Nonvalidating Parser will be adequatD. From another perspective, if a Schema is employed to build XML at one end, and unpack the data at the other end, then the validity of the XML is a moot point anyway. Alternatively, if the application is exchanging information with external, component based applications (developed to standard by another organisation), the document will be based on an agreement of both format and content to complete the data exchangD. In this case, a DTD or Schema, coupled with a validating parser provides a layered approach which externalises the validation and structure checking from the application code, allowing both sides to use the same validation components.

The markup declaration can contain ELEMENT type declarations, an ATTRIBUTE-LIST declarations, and ENTITY declarations

ELEMENT TYPE DECLARATIONS

Documents consist of components, typically, sentences, paragraphs and chapters, XML categorises these components as elements, which can also consist of other elements or character data [BOUMPHREY98]. As outlined earlier, the output from an XML document may be directed to a human reader or may be scheduled for processing by another computer process. Elements make up the branches in the tree structure generated by the parser, so validation of the element is critical, and may be considered valid if:

- The DTD contains a matching declaration for the named element
- The DTD contains matching declarations for the named elements attributes (if any)
- The declared data type of the element matches that in the DTD

ATTRIBUTE-LIST DECLARATIONS

An element can have attributes associated with it, attributes may be sub components of the element or further describe the element in some manner.

Each attribute has 3 components which fully describe the attribute: a name whose declaration follows the same naming conventions as the entity; a data type indicating the content which will be passed in; and an indication of the behaviour of the attributD. Each attribute must be declared in the DTD attribute list (ATTLIST) in order to be validated by the parser. There are a number of attribute data types which indicate to the parser, how to validate the data being passed in. If the document is valid and passes the constraints imposed by the parser, the contents are stored in memory in an 'object' tree structure identified as the Document Object Model (DOM).

D.5.3 THE DOCUMENT OBJECT MODEL (DOM)

There are a number of Document Object Model (DOM) application program interface's (API) which support accessing the contents of (valid) HTML and (well-formed) XML documents[WWW3C00-1]. In addition to using the DOM as a means of accessing the contents, the W3C has now produced level 3 of the specification, allowing developers to create documents, add, modify, or delete elements within the content area of a document. The DOM is language independent and the developer simply selects a binding which is appropriate for the development languagD. Bindings are defined for most Web Application languages eg. CORBA, Java, ECMAScript and COM/DCOM, ACTIVEX (Microsoft). The DOM is a programming API for documents, and is based on an object structure that closely resembles the structure of the document it models. For example, the HTML document in figure D.4 can be logically represented using the

```
<TABLE>
        <TBODY>
                <TR>
                        <TD>Heat
                Deposition</TD>
                        <TD>Abstract</TD>
                </TR>
                <TR>
                        <TD>Weld Pool
                Temp</TD>
                        <TD>Detail</TD>
                </TR>
        </TBODY>
</TABLE>
```

Heat Deposition	Abstract
Weld Pool Temp	Detail

Figure D.4 Style Sheet

tree structure shown in figure D.5. The Document Object Model provides a set of interfaces which are accessible after the document has been successfully parsed.

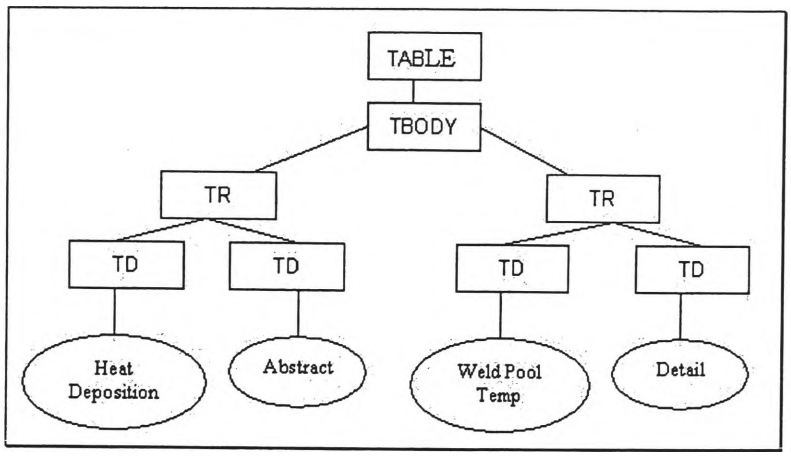

Figure D.5 Binary Tree

The DOM and Parser may be packaged within the same application eg. Microsoft MsXML or may be separate components eg. Docuverse DOM SDK which is a Java API. However the DOM is delivered, the model is presented via 'Node' objects which can be traversed by the client application. The model is accessed using a suite of interfaces which, when implemented, allow the application to enter at the conceptual 'root or document' level and traverse through each of the node levels (in figure D.4, the document level is the <TABLE> tag). Interfaces are implemented to reach the 'element nodes' (TBODY, TR and TD) as well allowing traversal at a logical level. To facilitate this, an interface (Nodelist) is provided which manages a list of Element Nodes and allows the developer to loop through the Nodelist, gaining access to the Elements. Attribute data (subordinate to the Element Node, and exampled as 'Heat Deposition', Weld Pool Temp' etc.) may be referenced using the NamedNodeMap interface which provides access to the various Entity data such as Text, Comments etc.

Figure D.6 lists the various Java Implementation Interfaces which may be used to traverse the in-memory tree, made available by the DOM.

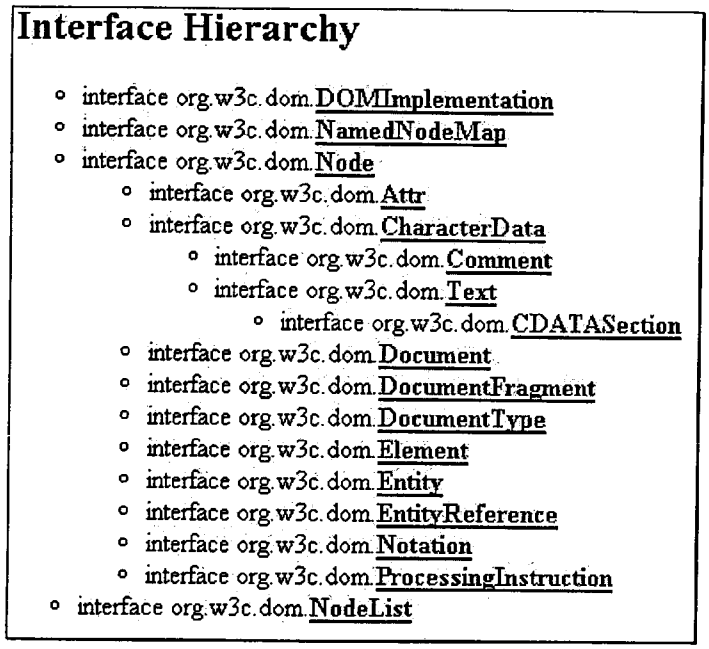

Figure D.6 Hierarchy

It should be noted that the memory requirements necessary to support the DOM are considerable, especially if the number of elements is large, and can be 5-10 times the size of the document itself [BOUMPHREY98]. If the XML document contains a large amount of data as well, the overhead can cause significant performance issues on the client Host.

If this is an issue for the application, another model which can be used to access the contents of the document is the Simple API for XML model (SAX), which delivers the contents of the document to the processing application in linear sequencD. The obvious drawback is that the SAX model does not allow the contents to be accessed randomly, if the processing application needs to randomly locate nodes, to modify or update the document to add value, the DOM is really the only solution.

D.5.4 SIMPLE API FOR XML (SAX)

It should be understood that the downside of using the SAX API, is that the Elements and Attributes which make up the document are not randomly accessible, as per the DOM. In the SAX API, the document is passed through as a Stream, with predetermined events (corresponding to the reading of Elements and Attributes) being triggered when a nominated Entity is read. Typically, the application is programmed to react to these events in a serial manner, the benefit of this model is that:

- If the document is large, and the application is filtering data which is triggered from the stream TAGS, this acts as an index and can be very beneficial in terms of efficiency. Of course, this facility is equally (if not more so) efficient using the DOM, but the overhead of loading the DOM far outweighs any benefit.

- The stream can be persisted then re-opened and read as many times as necessary to complete the document processing requirements. This benefit may be considered intangible, but the facility now exists to use XML as an indexing or lookup tool, eg. A Web application is passed a postcode, which becomes an argument to a SAX query on postcodD.xml. The resultant read returns a Suburb to the Web application.

As with the DOM, there are a number of freely available implementations of the SAX API, all are C++ or Java implementations, except the Microsoft offering which supports VB and COM.

Figure D.7 lists the Java Implementation Interfaces which use the XMLReader stream to facilitate triggering specific ContentHander methods. The ContentHandler interface receives notification (callback) of document events (Element start and end) as the content passes through the XMLReader. The order of events matches the order of information in the document itself. An element's content (data, processing instructions, sub elements etc.) appear in order between matching startElement endElement event pairs.

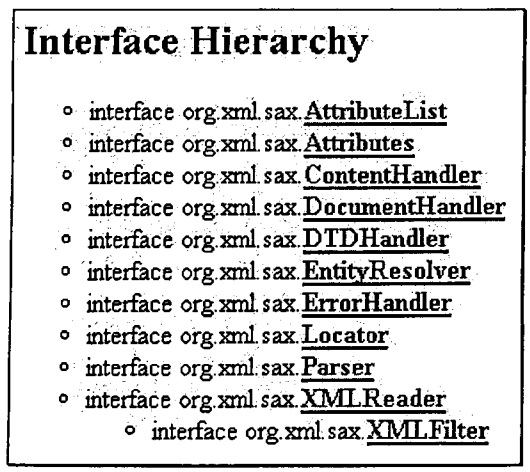

Figure D.7 Hierarchy

Obviously, the code is not cohesive and the developer must test for the tag name/s which the application is interested in, eg;

```
public void startElement(String URI, String Name, String qName, Attributes atts) throws SAXException
{       If  (namD.equals("welder_id") DoSomething();   }
```

but the API is indeed 'simple' and can be very efficient in terms of processing data from a large source where filtering is required or processing can be stopped after a certain event has been handled.

D.5.5 PIPELINE PROJECT RATIONALE

Of the two (DOM vs. SAX), neither model is absolutely suitable for the Framework. The DOM has advantages; because of the variation in output format, the application would need to be able to access different nodes in different circumstances, so random access would be a plus. On the other side, because of the volume of data and the fact that the application knows exactly what format the XML document is in, would be suited to the use of the SAX interfacD.

After analysis of the various models, the best outcome for the project was the use of a Schema as a way of getting the best of both worlds, the Schema can:

- Be used to build the XML document as well as interpret the data at the target host.
- Be used to build other data file formats, and in the case of the framework, the CSV formatted data uses the schems to create the file layout.
- Be designed to build a memory efficient DOM, while still using the stream attributes of the SAX model

The case for using XML is strengthened to an irresistible level when the application can take advantage of self managed data resource components[SOO01] which provide utilities for browser driven upload and download. Using these components, pipeline specific XML documents can be generated, and posted to another site via the plain old Web Server; Data stored within the XML document container can be retrieved from a remote site by simply providing a link to the document on the remote sitD. The Web Server at the remote site gathers up the file and requests whether the User would like to view the file, or save it to the local system. The ease with which third party components have been melded with the specifically written code, and integrated into the overall Framework application has reduced the amount of overall code developed to a minimum. Other than wrappers, which were developed to interact with the API's, the major development area for the framework is the interface to the Schema. The Schema is is used to build a DOM for program

modules to load or unload XML based data, or convert the data into comma separated variable (CSV) format.

D.6 XML SCHEMA

As mentioned earlier, use of the Schema facilitates programmatic control to:

- Build the XML document
- Build files in other formats eg. CSV and this is the example presented , which can be input into a spread sheet
- Read the XML document

ANALOGY

The World Wide Web can be viewed as a rail network, with Hosts (Stations) facilitating TCP/IP sessions (Locomotives) pulling freight containers filled with both raw materials and semi-constructed components (XML data). Some containers are further compartmentalised, and contain discrete components or building blocks which need to be combined with other building blocks before a data component can be constructed. Organisations who have access to the 'rail stations' may now generate XML 'goods' containers programmatically, and forward those 'containers' to designated 'stations', which are received by 'assembly' applications for re-construction into data components. XML 'containers' are always shipped with assembly instructions, which means the applications which re-construct the contents into data components can be generic, and do not have to be specifically designed to deal with the contents of the 'container'.

Continuing with the analogy, the Schema provides a set of assembly instructions for any plant along the way who needs to prepare a data component (document or data transaction). In addition to this, the Schema can be used by applications as assembly instructions on how to GENERATE the XML document, prior to shipment. This is the other side of the process, ie. where the original document, raw data, or component resides.

There has been a lot of discussion on the use of the Schema, and a number of organisations in similar industries have banded together to develop a Schema which is suitable and applicable to their industry. This provides a higher level of generic abstraction, and means that each of the participants can prepare and ship XML containers to anyone who wishes to access that XML document. There is a high

degree of relevance for the use of a Schema when the data being shipped is data which is usually resident in a Relational Data BasD. As mentioned earlier, the issues of constraints, referential integrity, default values and cardinality which are important properties for data storage repositories. To be of any genuine use, the Schema must conform to a structural standard, ie. have a published method for the declaration of elements, and provide a method of data definition. In Object Oriented terms, this is a description of an Object, the purpose of a Schema is to define and describe a class of XML documents by using constructs to constrain and document the meaning, usage and relationships of their constituent parts: data types, elements and their content, attributes and their values, entities and their contents and notations. Schema constructs may also provide for the specification of implicit information such as default values. Schemas document their own meaning, usage, and function [WWW3C00-2]. We can also think of the XML document as an external data 'jacket', which describes the relational structure of the internal Data Repository.

D.6.1 STRUCTURE

The structure of the Schema supports the Element/Attribute component Model which provides the most flexibility for database support. There are 13 kinds of component in all, falling into three groups; primary components, secondary components and helper components. The XML Schema model relates implicitly to the database Schema model, so there are declarations for the definition of constraints on the content which can be applied when processing the data, or using the raw data in a value added process, such as a conversion to HTML.

PRIMARY COMPONENTS

In the components listed below, note the distinction between definition and declaration. Definitions are used to indicate a coming declaration, such as, element and attribute declarations.

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema">
 <element name="PipeWeldData" type="PipeWeldDetails" />
 <complexType name=" PipeWeldDetails">
   <sequence>
     <element name="Welder" type="WelderDetails" />
     <element name="Asset" type="string" />
   </sequence>
 </complexType>
```

```
  <complexType name="WelderDetails">
    <attribute name="firstName" type="string" use="required"/>
    <attribute name="lastName" type="string" use="required"/>
    <attribute name="emailAddress" type="string" use="required"/>
  </complexType>
</schema>
```

Figure D.8 Schema

Note that the use of 'Sequence' within the <PipeWeldDetails> element indicates that the inclusive element information items (Welder and Asset) must match to the XML document in sequential order). The components are named as follows:

Simple type definition: Defined as a set of constraints on strings and information about the values they encodD. Typically, the simple type applies to elements that have no attribute values or no element children. Refer to the line in figure D.8,

```
<element name="Asset" type="string" />
```

which defines an element named 'Asset', declared to be of type 'String'. We could mark the document up as follows:

```
<simpleType>
    <element name="Asset" type="string" />
</simpleType>
```

While this would be irrelevant, and possibly a costly bandwidth overhead, it is indeed a valid syntax for the component, and demonstrates the implicit/explicit nature of that syntax.

Complex type definition: This is the most common type definition and is defined as a set of attribute declarations and a content type, applicable to the attributes of an element information item, or child elements. Notice the 'PipeWeldData' and 'Welder' elements in figure D.8, these elements define complex elements 'PipeWeldDetails' and 'WelderDetails', who have child attribute declarations

```
<complexType name="WeldDetails">
  <attribute name="firstName" type="string" use="required"/>
  <attribute name="lastName" type="string" use="required"/>
  <attribute name="emailAddress" type="string" use="required"/>
</complexType>
```

Element declaration: Mentioned above as a simple type definition, the element declaration <element name="Asset" type="string" /> is a simple type element declaration because it has no child attribute or sub element declarations.

Attribute declaration: The attributes further describe the contents of the parent element, eg . <attribute name="firstName" type="string" use="required"/>.

SECONDARY COMPONENTS

Attribute group definition: An attribute group definition is an association between a name and a set of attribute declarations, enabling a data structure concept for the re-use of an attribute set (or group), in complex type definitions. For example, say we declare the following Address element:

```
< attributeGroup name="Location">
  <attribute name="Street Number" type="string" use="required"/>
  <attribute name="Street Name" type="string" use="required"/>
  <attribute name="Nearest Cross Street Name" type="string" use="required"/>
  <attribute name="City" type="string" use="required"/>
  <attribute name="PostCode" type="integer" use="required"/>
  <attribute name="State" type="string" use="required"/>
  <attribute name="GPS" type="string" use="required"/>
</ attributeGroup >
```

We could then reference this attribute group element in subsequent references as simply 'Location'. In addition, multiple occurrences can be used to facilitate instances of location, refer to the use of 'currentLocation' and 'basedAtLocation'.

```
<complexType name="Welder">
  <attribute name="Welder Name" type="string" use="required"/>
  <attribute name="emailAddress" type="string" use="required"/>
  < attributeGroup ref = "Location" name="basedAtLocation" />
  < attributeGroup ref = "Location" name="currentLocation" />
</complexType>
```

Identity-Constraint definition: In addition to providing a naming convention for Elements and Attributes, the Schema model provides a method of declaring constraints to an engine which is posting the XML to a databasD. Definition components provide for uniqueness and reference constraints with respect to the contents of multiple elements and attributes. Consider the following XML extract

for a pipe bundle where multiple length items <LineItem> appear in the document. Each pipe code represents a specific length of pipD.

```
<BundleData>
  <Bundle
    orderDate="12/1/2000">
    <LineItem lengthIDREF="p1" quantity="17", diameter="18"/>
    <LineItem lengthIDREF="p2" quantity="22", diameter="18"/>
</Bundle>
  <Shipping
    shipDate="12/4/2000" shipMethod="ROAD">
    <LineItem lengthIDREF ="p1" quantity="10" backorder="7" />
    <LineItem partIDREF="p2" quantity="22" backorder="0"/>
  </Shipping>
 </BundleData>
```

An Identity-constraint definition plays one of three roles, Unique, Key and Keyref.

Unique: The Schema may identify an attribute as unique using the following syntax,

```
<unique name "lengthId">
    <selector xpath = "/BundleData/Bundle/LineItem" />
    <field xpath ="lengthIDREF"/>
<unique>
```

which asserts uniqueness, with respect to the content identified by the selector <selector xpath = "/BundleData/Bundle/LineItem" />, the data resulting from evaluation of the lengthIDREF expression will be deemed unique by the parser. Similarly for the key and keyref definitions.

Key: asserts uniqueness (as for unique) for all selected content that meets the evaluated criteria.

```
<key name="lengthNumber">
    <selector xpath = "/BundleData/Bundle/LineItem" />
    <field xpath="@lengthIDREF"/>
</key>
```

This tells the Schema validator to verify that every length ID number (in the bundle) has a lengthIDREF, and that lengthIDREF must be uniquD.

Keyref: asserts a correspondence, with respect to the content identified by the selector, <selector xpath="/BundleData/Shipping/LineItem"/> of lengthIDRef, with those of the referenced key, lengthNumber.

```
<keyref name="lengthIDRef" refer="lengthNumber">
        <selector xpath="/BundleData/Shipping/LineItem"/>
        <field xpath="lengthIDRef"/>
</keyref>
```

This tells the Schema validator that the pipe length number of the Line Item being shipped, must refer to one of the line item elements in the collection defined by the lengthNumber key.

Model Group definition: The component has the same usability as the attribute group, but applies to lists of element information items consisting of declarations, wildcards or other model groups.

```
<group name="WelderGroup">
  <sequence>
    <element ref="Welder"/>
    <element ref="Asset"/>
  </sequence>
</group>
```

This grouping allows the Schema validator to refer to the complex types of Welder and Asset as a group, multiple times within the document. The applicable declarations, wildcards or other model groups are collectively referred to as 'a list of particles', and the model group provides for selective inclusion of the particles in the group by using the syntax sequence (the element information items match the particles in sequential order), all (the element information items match the particles, in any order), or choice (the element information items match one of the particles). For example, we could modify the above example to allow the Schema validator to select either Welder or Asset when the parser hits a reference to WelderGroup

```
<group name="WelderGroup">
  <choice>
    <element ref="Welder"/>
    <element ref="Asset"/>
  </choice>
</group>
```

Notation declaration: The notation element allows an association between the named element and another file (URI). The element item must declare a pair of properties, SYSTEM and PUBLIC, which provide additional information on the correspondence between the notation element and the filD. For example, a digitised scan file may accompany the associated weld data, and can be alternatively stored, but referenced in the XML document, with the physical location details provided as a URI.

```
<notation name="jpeg" public="image/jpeg" system="viewer.exe">
```

declares a name "JPEG" which corresponds to a data type meeting the ISO 8879 specification public="image/jpeg", and associates a named file which can be found at the following URI system="viewer.exe". The most typical use for the notation element is to provide extension information (eg. mime) to an elements which may required additional processing. It is also common practice for data to be stored in binary format within a database as a Binary Large Object (BLOB), the notation element facilitates value added enablers to spawn a process to deal with an element being referred to in the notation.

HELPER COMPONENTS

Annotations: Annotation of the Schema and Schema components, with material for either or both human and computer consumption. The facility is provided for by allowing application information and human information at the beginning of Schema elements, and anywhere at the top level of the Schema. The XML representation for an annotation Schema component is an <annotation> element information item.

```
<annotation>
  <documentation>
        Some indication of delivery expectation should be provided on shipped bundled
        documents, and details when items could not be included in the bundle (backordered)
  </documentation>
  <appinfo>
    <attachment "Backordered lengths will be supplied from alternative sources within 12 hours"
</attachment>
  </appinfo>
</annotation>
```

The content must be well-formed XML, and unlike standard comments, is passed through the parser and becomes available to underlying processing application, however, has no effect on the Schema validation.

Model Groups: The model group definition (described earlier) may refer to a model group identified below. The model group facilitates filtering of a named group in potentially multiple areas of the Schema. As outlined earlier, the model group is 'a list of particles', and the model group provides for selective inclusion of the particles in the nominated group by using the syntax `sequence, all or choicD.`

Particles: A particle is part of the grammatical term making up element content, and consists of either an element declaration, a wildcard or a model group, together with occurrence constraints. Particles contribute to the validation of complex type definitions by providing granular syntax content. Typically, particles provide a syntax to define cardinality of the element content `<element ref="lineItems" minOccurs="1" maxOccurs="12"/>`

Wildcards: A wildcard is a special kind of particle which matches element and attribute information items dependent on their namespace name, independently of their local names. Wildcards allow the author of XML documents to reference other 'external' inclusions (using the namespace concept), and extend the document. The use of process contents such as strict, skip and lax, allow the author to include or exclude certain external content by using the wildcards to filter the inclusion.

```
<any namespace="http://www.w3.org/1999/XSL/Transform" processContents="lax"/>
```

Attribute Uses: An attribute use is a utility component which controls the occurrence and defaulting behaviour of attribute declarations.

```
<attribute name="Welder Name" type="string" use="required"/>
```

The XML Schema language defines mechanisms which allow for:

- The constraining of document structure (namespaces, elements, attributes) and content (data types, entities, notations)
- Enabling inheritance for element, attribute, and data type definitions
- URI reference to standard semantic understanding of a construct
- Embedded documentation
- Application-specific constraints and descriptions
- Addressing the evolution of Schemata

- Enabling the integration of structural Schemas with primitive data types.

D.6.2 DATA

DATA TYPES

The specification provides a syntax for the description of the following data types:

- integer - nonPositiveInteger - negativeInteger - nonNegativeInteger - unsignedInt - int (small int) – positiveInteger
- short – unsignedShort
- long – unsignedLong
- byte - unsignedByte
- normalisedString - token - language - NMTOKEN - NMTOKENS – Name
- NCName - ID - IDREF - IDREFS - ENTITY - ENTITIES

FACETS

In addition, the specification provides the following constraining facets for the document creator to dictate validation constraints, such as Patterns, Enumeration, Whitespace etc.

Pattern: `<pattern value='[0-9]{5}(-[0-9]{4})?'/>`

Enumeration: which constrains allowed values fall within a specified set of values

```
<enumeration value='7'>
    <annotation>  <documentation>Sunday</documentation> </annotation>
</enumeration>
```

WhiteSpace: `<whiteSpace value='collapse'/>` Constrains the value space of types derived from string to a value that must be one of preserve, replace or collapsD. When preserve is nominated, the value is not changed, when replace, all occurrences of tab, line feed and carriage return are replaced with a spacD. Finally, collapse, after the processing implied by replace, contiguous sequences of spaces are collapsed to a single space, and leading and trailing spaces are removed.

The specification provides more many facets which test equality, ordered sequence, bounded sequence, cardinality and numeric equality. The list of these is outside the scope of this discussion, however, the issue is that the W3C is providing both a means of deploying data outside a database, as well as a set of tools for validating

and constraining that external data representation. Third party vendors like IBM and Microsoft have not only adopted XML but are embedding the technology into their core products. SUN are now using XML to deploy Java source code to Clients via the Web for remote instanciation. This concept in itself has enormous implications for distributed software application development and maintenance tasks.

D.7 PIPELINE PROJECT ISSUES

D.7.1 DESIGN CRITERIA

There are a number of issues which make the Schema method a good fit for the Framework under discussion. Most importantly is the volume of data being generated from a weld, which causes an overhead when the data is extracted from the database, shipped over the Web, then unpacked at the target end. Each of these steps take time and increase the total wait time for an interactive user. It is important to note that the Schema methodology provides a lot of scope for the developer to model the Schema so as to maximise the efficiency of the syntax to best suit the application under development. The side effect of this flexibility is that there is no DTD to allow the parser to build an in memory tree, the developer must provide a parser which first reads the Schema and then processes the XML document. As mentioned earlier, there are a number of Schemas which have been put forward for use by interested industry groups, with corresponding Schema parsers to allow the applications using the particular Schema access to the Schema DOM (Document Object Model). This concept may be taken further by the developer, who may build a Schema which is application specific (rather than industry specific), and this is the case with the Framework Schema. There is no need to have the Schema registered if the Schema is being used in house, as long as both sender and receiver have access to the Schema DOM. The Framework Schema has the following design criteria:

- There is no necessity to provide for human access to the XML, although the Schema is 'well formed'.
- The Schema must provide information to allow the extracted data to be persisted in Comma Separated Variable format (CSV).
- Because of the data volume, the Schema must facilitate the most efficient form of document storage.

- The Pipeline project is supported by a data transport Framework, which means that Schemas will be developed for additional tables being transported by the framework. In order to facilitate this process, the Schema must provide a simple format which can be followed by non technical users who wish to use the framework facility. The entire basis for the framework is extensibility for the non technical User, therefore the Schema must be simple enough for a non technical (computer wise) User to generate, and deploy.
- The Schema can be used to build the XML document as well as Read the XML document.

REFERENCES

[ASBURY99] Stephen Asbury and Scott Weiner, Java Enterprise Applications, 1999

[BOUMPHREY98] Boumphrey et al., *XML Applications*, Wrox Press 1998

[HAEFEL00] Richard Monson-Haefel, *Enterprise Java Beans*, O'Reilly, 2000

[ROBBINS96] Kay A. Robbins and Steven Robbins, Practical Unix Programming, Prentice Hall 1996

[ORFALI96] Orfalli, Harkey and Edwards, *The Essential Distributed Objects Survival Guide*, John Wiley & Sons, 1996

[ORFALI98] Orfali and Harkey, Client/Server Programming with Java and Corba, Wiley, 1998

[SOO01] Joo Kyung-Soo, *Design of Middleware components for the connection between XML and RDB*, Industrial Electronics, Vol. 3 June 2001, Proceedings, ISIE 2001, IEEE International Symposium, Pusan, South Korea pp. 1753-1756

[STLAURENT98] Simon St. Laurent, Xml A Primer, MIS Press 1998

[STEVENS94] W. Richard Stevens, TCP/IP Illustrated - Volume 1, 1994

[WILLIAMS00] Kevin Williams, Professional XML Databases, Wrox Press, 2000

[WWW3C00-1] http://www.w3.org/TR/2000/REC-xml-20001006

[WWW3C00-2] http://www.w3.org/TR/NOTE-xml-schema-req

Betta Bookbinding & Printing
M & D Morrisey 02 42612998
26 Fields St
Kanabooka NSW 2530