A RAD Framework for Cross-Platform GUI Development of MVDBMS Client-Server Applications

John George Bend
john.bend@ITStuff.net

Dr. Samuel Sambasivam
ssambasivam@apu.edu
Computer Science Department, Azusa Pacific University
Azusa, CA 91702, USA

Abstract

Applications based upon the PICK family of Multi Value Database Management Systems (MVDBMS) have been serving their users needs for decades. However the majority of MVDBMS applications pre-date Graphical User Interfaces and Web application technologies. There is a recognised need within the MVDBMS development community to establish an uncomplicated development framework which continues the recognised benefits of PICK whilst helping MVDBMS applications compete in the current market place.

Keywords: Database, Multi-value, GUI, MVDBMS, SWT, Programming, Interface, API

1. INTRODUCTION

The PICK family of Multi-Value Databases Management Systems (MVDBMS) provides efficient and reliable data storage, manipulation and interrogation, together with a richly featured BASIC programming language. The unique features of MVDBMS enable developers to concentrate on the task of creating an application rather than the technology behind it. This translates to rapid application development and low software maintenance costs. Today there are many legacy MVDBMS applications still in use with development roots that predate today's graphical interface and n-tier architectures. Whilst these applications are still able to satisfy the user's needs they are text-based and perceived by the market as tired and uncompetitive when compared to modern applications. Thus vendors and developers recognised the need to move these applications over to graphical desktop or web-based clients. Unlike Relational Database applications, there is no established development framework to which MVDBMS developers can turn to. Working in isolation, MVDBMS vendors and developers have been obliged to engineer various competing development solutions. Typically these solutions lock the application into restrictive technologies, costly licences or a lengthy development process. This paper discusses an attempt to establish an MVDBMS development framework for developing client-server applications.

The PICK multi-value database model

"Multi-value is a database model with a physical layout that allows systematic manipulation and presentation of messy, natural, relational, data in any form, first normal to fifth normal. In other words: with repeating groups in a normalized (one- key and one-key-only) table." (DM Review Sep 2002).

The PICK database model can be traced back to 1965 when Richard Pick and Don Nelson developed an information retrieval language for a large US military contractor (Bourdon 1987). The Generalised Information Retrieval Language System (GIRLS) enabled remote terminals to locate rocket engine parts anywhere within the group. The US army took interest in the project and funded development of a data management system for the Cheyenne helicopter project. Since the project was funded by public money details of this data management system became readily available within the public domain, which allowed Richard Pick and Don Nelson to develop the system further and create a commercial version of the system in 1972. Since then numerous licenses and versions of the MVDBMS or PICK model have become established. Figure 1.

Multi-value databases are characterised by a number of features (Bourdon 1987, Sisk 1990, Tincat Group Inc. 2006) in that they:

- Have a hierarchical filing system which employs a hashing algorithm for storing and retrieving data. Data is not stored is 1st Nominal Form and is of variable length.
- Provide a Terminal Control Language (TCL) which provides an interface between the user and the multi-value database system.
- Possess an editor to enable online modification of data, system records and source code.
- Provide a "PROC" processor which executes scripts.
- Include a version of BASIC (Beginners All-purpose Symbolic Instruction Code) language. This is a compiled version of the original Dartmouth BASIC language, greatly extended to provide powerful data and string handling commands. The language implementation also supports un-typed data structures.
- Include a non-procedural database enquiry language (Access or English).
- Utilise an output Management System or SPOOLER. This provides and manages printer output and tape operations.
- System Accounting which provides a means of monitoring and controlling the system.
- Contain a Security System which enables and prevents access to information according to assigned privileges.
- Provide a document output processor (RUNOFF).

2. MVDBMS APPLICATIONS

Such databases were once the mainstay of the Health Service and local government, especially within the UK. Today, multi-value database software applications around the world are supporting user counts from tens to hundreds or users. Where established these multi-value database applications tend to dominate their vertical markets (DB Forums 2004). For example in the USA the market for car dealer software is dominated by two companies: Reynolds and Reynolds and Automatic Data Processing (ADP). The products from both these companies run on multi-value databases (Reynolds and Reynolds, ADP).

3. FRAMEWORK CONSIDERATIONS AND REQUIREMENTS

The aim is to establish rather than create a development framework underlining the intention to research and invent a workable development framework.

The framework is intended to support most MVDBMS offerings and a Graphical User Interface (GUI) running on popular graphical desktop platforms and embrace industry standards and protocols. It should continue to support MVDBMS rapid application development and provide developers with an application development environment which is MVDBMS vendor neutral.

The framework should support Rapid Application Development (RAD) of Graphical User Interface (GUI) clients capable of running on a number of popular desktop platforms and be firewall friendly in recognition of modern networking requirements.

Over and above the technical features the MVDBMS community insist that the most

outstanding feature of this family of databases is the support for rapid application development, allowing programmers to concentrate on the matter of developing database applications rather than the technology behind it. Although difficult to quantify, this principal should be continued by the development framework.

Cross-platform server
Multi-value databases are available on a wide range of server and workstation operating systems which includes Windows, Linux, Solaris and SCO. It follows then that any server component of the run-time framework (middleware) must also run on these operating systems.

Cross-platform client
Browser statistics gathered by W3Schools (W3Schools, January 2006) suggest that Microsoft Windows continues to dominate the desktop as we suspect. In 2003 Linux was already the second most popular desktop operating system and forecasts have place it's desktop adoption between 6% and 10% by 2007 (IDG News Service, August 2004).

There is a growing trend for businesses and organisations to consider the various Linux distributions as an alternative desktop platform.

> *Penwith District Council rejected Microsoft Windows and Office in favour of Sun Ray terminals running Open Office to save over £150,000 (Computer Weekly January 2002).*
>
> *West Yorkshire Police in the UK, working in conjunction with Netproject the Police IT Organisation (PITO), migrated 3,500 desktops to Linux. Plans exist to migrate the entire 60,000 desktop computers across England and Wales if the pilot project is successful. (OSI, June 2003). The reader should note that this decision was driven not by cost of ownership but for the need for greater security.*

Designing, developing, testing, documenting and selling a client-server application represents a significant investment in time and resources. Therefore, unless there are specific reasons for locking into a particular client platform, it is simply good business sense to develop software that will run on a choice of platforms rather than just one. We should also consider that MVDBMS application developed with the framework proposed by this paper are likely to be replacing legacy systems running on dumb terminals. The ability to deliver GUI clients on low cost PCs running Linux would avoid loading the overall cost of upgrading where the higher cost of a Windows PC could not be justified. The development framework should therefore support GUI clients capable of running on the most popular desktop platforms. These include Microsoft Windows, Apple Macintosh and Linux X-Windows.

4. TECHNOLOGY REVIEW

Before designing the development framework or embarking upon a review of programming languages, development tools and technologies it was necessary to identify the principal components of the development framework, together with the functional and non-functional requirements. Figure 2 (Appendix) illustrates a logical view of a MVDBMS client-server application where we can identify the fundamental components.

- The Application Client
- Middleware to enable interaction between the Client and the Server
- The Application Server

From this we can see that the development framework requires a development environment for creating GUI clients and middleware.

Client Development Environment
Whether right or wrong, customers tend to judge applications by the look and behaviour of the GUI. When creating a GUI client for any application then, it is important to strive for a high quality "look and feel". It is important that the development framework provides application developers with the maximum competitive advantage in this respect. The development framework GUI should have good performance and ideally provide a look and feel that is "native" to the graphical desktop hosting the application client.

A GUI can be created using a visual tool or may be crafted by low level code. As we might suspect there are arguments for and against both approaches.

Visual GUI tools	
✓	Allow rapid creation of screens and forms.
✓	Forms, screens and menus can be laid out by suitably trained technicians rather than skilled programmers.
✓	Allows rapid creation of prototypes which can be presented for customer approval.
✓	Quick and easy to effect minor changes with minimal impact on the client application code.
✓	Design phase can also be the build phase.
✓	Using a visual GUI editor can help force the application developers to keep the GUI code and client logic code separate.
×	Can frustrate attempts to create anything that is beyond limits anticipated by the authors of the tool.
×	Can result in verbose and perhaps inefficient code.

Hand coded GUI	
✓	By recognising component patterns the experienced programmer can create smaller and more efficient GUI programs through component reuse. (De Vlaminck, 2001).
✓	Allows fine grain control of the GUI.
✓	Intelligent use of layout managers can result in better behaved screens and forms.
✓	Code can be created which renders the client GUI in accordance with instructions received from the application server.
×	Can be time consuming to create and maintain.

On balance the speed and cost benefits of using a visual GUI editor far outweigh the negatives. An ideal GUI solution for this development framework would be to support both approaches for creating GUIs. Thus most GUI components could be created quickly and easily using a visual tool with complex GUI components coded by hand.

General Requirements

In considering the requirements with which to evaluate and select tools and technologies for the development framework three principal areas were identified. These are; development, functional and marketing as illustrated in Figure 3 (Appendix). The Development Requirements are simply that a client-server application development framework needs to support development of client-server applications. The Functional Requirements require the development framework to include run-time elements that support the continued running of the client-server application. The framework runtime must be capable of doing its job and support client-server operation over LAN or WAN. The Marketability (Non-functional) requirement dictates that the development framework should add value to the application. Developers need to be able to sell their finished applications.

Client (GUI development) Requirements

The requirements for the client GUI development environment have been identified as:

- Cross-platform run-time (MS Windows, Apple Mac and Linux
- High quality GUI with native look and feel
- Easy to use GUI development tool
- No distribution costs or royalties
- Allow hand coding of GUI if possible
- Support for internationalisation
- Based on or incorporates a well established programming language
- Well documented and supported
- Simple installation to client workstations (roll-out)
- Must co-exist with middleware technologies
- Cross-platform development would be a bonus

Middleware

The Carnegie Mellon Software Engineering Institute describes middleware thus:

> *"Middleware is connectivity software that consists of a set of enabling services that*

> *allow multiple processes running on one or more machines to interact across a network. Middleware is essential to migrating mainframe applications to client/server applications and to providing for communication across heterogeneous platforms. This technology has evolved during the 1990s to provide for interoperability in support of the move to client/server architectures." (Carnegie Mellon, Software Engineering Institute)*

Figure 4 (Appendix) illustrates the classic middleware services model as described by Bernstein (Bernstein 1996). Application clients interact with an Application Programming Interface (API). There can be any number of client implementations and in this paper we are assuming that these clients will be on dissimilar operating systems but only one version of the API. The API executes actions on the appropriate Application Server via interaction with a Platform Interface. The Platform Interface is usually specific to the Application server's resources and operating system. In this paper the Application Framework must ultimately support any of the multi-value databases available that support network communications. The author has established that each of these multi-value databases support network communications in quite different ways. It is the job of the API therefore to provide application developers with a unified, abstracted interface and mask the complex communications between itself, the Platform Interface and ultimately the Application Server.

Following the growing availability of high speed, low cost, reliable internet connections organisations are increasingly considering hosted applications. In this scenario Application Clients seamlessly interact with the Application Server over a high speed internet connection. This allows the Application Server to be located at almost any geographic location and allows fairly inexpensive application access to any location that is connected to the internet. Usually the server is located within a third party organisation that has the expertise to maintain and backup the application and data on the behalf of the users. It would therefore be highly desirable for the development framework middleware to support LAN and WAN access via the Internet. If the middleware could support WAN access via the Internet then it would be preferable to adopt protocols that do not require special handling by firewalls.

It is recognised that Internet technology continues to develop and grow quickly. With this in mind it would be prudent to use third party components wherever possible. Use of third party components would give leverage to the project development both in speed of development and helping to ensure that standards and protocols are correctly adhered to. Preference should be given, where possible, to third party components that possess a degree of maturity.

Middleware requirements

Thus we have identified the key requirements of the development framework middleware as:

- Cross-platform
- Support operation across LAN or WAN
- Firewall friendly
- Loose coupled (cannot depend on Server and Client having same platform or language)
- Interfaces with but is independent of the Client
- High availability
- Scaleable
- Provide the application developer with a familiar programming interface at the client
- Vendor neutral
- Well documented and supported
- Use established programming language
- Use third party components where possible
- Give preference to components with some maturity
- Use accepted protocols and standards where possible
- No distribution costs or royalties

5. SELECTION & EVALUATION

The selection process

After evaluating the component and requirements consideration was given on how to select and evaluate candidate technologies. The natural preference would be to identify a fully integrated

development solution which fully met all the criteria.

Fully integrated development solutions
Mono, Mono Project
A well supported project to create an Open Source (free) implementation of .NET. A legal implementation based on clean-room coding against Microsoft's published ECMA/ISO standards. It is intended to be fully cross-platform and whilst it is of great interest, alas it does not yet have a mature visual designer.
Revolution Studio, Runtime Revolution
British product originally released in 1997. Mature and fairly stable. Revolution Studio has a good visual designer and is cross-platform with web support. The principal problems are dependency on an obscure programming language, no support for MVDBMS. Royalty costs.
Omnis Studio, Raining Data
An exciting proposition from Raining Data (formally PICK Systems). Cross-platform, web support and forms based visual designer. Has obscure programming language, royalty costs and unreliable support for MVDBMS operations.

Since a fully integrated solution was not available the next preference was to identify candidate components that would work in concert. For convenience the search was divided into client GUI and middleware. It was understood that an iterative search would be necessary in order to identify a collection of technologies that worked with each other whilst having a "best fit" of the criteria.

Client technologies

The requirement was for a cross-platform graphical toolkit, together with appropriate programming language and visual designer. If a suitable graphical toolkit could be identified then a corresponding development environment would be searched for. The toolkit and programming language would need to work in conjunction with the chosen middleware technologies.

Fully integrated development solutions
QT Designer, Trolltech Inc.
Promoted as a "comprehensive development framework". Able, polished and well established. On which the KDE Linux desktop manager is based. The cross-platform licence is expensive at €5,260 and steep learning curve.
SWT and Eclipse, Eclipse Org.
A lightweight Open Source graphical toolkit for Java. Originated by IBM and the toolkit from which Eclipse was created. Mature, well supported and free. *SWT provides programmers with an abstracted interface which wraps native graphical components from the underlying desktop platform (Guojie 2005)*.

On balance SWT, Eclipse and Java offered the best match to the client criteria which left a need for the visual designer. A number of candidates were available but only Window Builder Pro from Instantiations Inc. was sufficiently mature enough to support end-to-end application development. This product was not expensive at $199 per development seat, is well supported and attracts no royalties. As an interesting aside Window Builder Pro supports two-way development between visual development and hand coding of the GUI. The Eclipse integrated development environment is mature and very well supported with add-ons such as code assistants, integrated help and documentation, packaging and CVS control. Deployment of SWT GUI applications does require distribution of the royalty free libraries but documentation suggests that this can be handled by Java WebStart. The selection of client technology had determined the choice of Java as programming language.

Selection of Middleware technologies

Research suggested that there were two real choices for loose-coupled, wide area network communications: Microsoft .NET and Java Web Services. It would be possible to

implement cross-platform middleware in Mono .NET and C#. This would increase the number of languages and technologies within the project. Java has many Web Service tools and technologies, such as Tomcat, from the Apache Foundation. The technology is mature, well supported, free and based upon the same language as the client.

Using Java and Tomcat it would be perfectly possible to write a Java Servlet that interacted with the Java Client API using Java network sockets. However this would not meet the criteria of firewall friendly or adopting established standards. Further research revealed that in order to meet this criteria the only choice would be to use Simple Object Access Protocol (SOAP). SOAP allows the loose-coupled interaction between two software technologies using a firewall friendly protocol based upon XML messages using the HTTP protocol (Snell, Tidwell, Kulchenko 2002). Though SOAP can be used in a number of ways, for the purposes of this paper in establishing a development framework it would be perfectly acceptable to use it only for synchronous Remote Procedure Calls (RPC). Further reading suggested that whilst SOAP would provide the communications interaction that met the framework criteria there would be a lot of programming work involved. SOAP is far from simple. Thankfully the Apache Foundation have a fairly established software component. Apache Axis provides Web Services programmers with the leverage to code SOAP interaction and masks most of the complexity. It can be run as a stand-alone Web Service within Tomcat or like Tomcat can be wrapped in application code to create a fairly secure, self-contained Tomcat Servlet. Axis was fairly mature (as Web Services components go) and seemed fairly well documented. One of the most important features of Axis was that it promised to support maintained state sessions which are vital to client-server applications. This is discussed in chapter 3.

At this stage a collection of technologies suitable for the development framework middleware had been selected. On the face of it all the criteria would be met. At each step in the selection and evaluation process candidates were uncovered, examined and where possible tested. It was now time to begin building the framework.

6. IMPLEMENTATION

Standing on the shoulders of giants.
(Isaac Newton 1676)

Implementation Overview

For the purposes of selecting technologies the development framework was divided into two categories; Client GUI and middleware. In the implementation phase it was appropriate to divide the framework further. Figure 7 illustrates a more detailed view of the framework.

From the illustration we can identify three high level components;

- Client
- Service
- Server

Note that the illustration is colour coded to identify components which are created by an application developer, components which are developed especially for the development framework and third party components that were uncovered during the selection and evaluation process. In most cases the Client would reside on the user's computer or workstation while the Service and Server would be hosted on a server. In certain cases, where no suitable Java is available for the MVDBMS host platform for example, it would be necessary for the Service to reside on a separate host.

7. CLIENT OVERVIEW

The Client is the entire portion of the application that resides and operates on the user's computer or workstation. The Client comprises of:

Client GUI

This is created either by the visual designer tool or the application developer as part of the application creation process. The GUI is the interactive part of application visible to the user and is created by the application programmer.

Client Logic

This is code created by the application developer to assist and control the logical flow and behaviour of the client screen. This code forms an interface between the GUI and the framework API. The Client logic is invisible to the user and is created by the application programmer.

Java Client API
The Client API is presented to the application developer in a library or Java package. Although a distinct part of the development framework the API draws upon certain functionality from the Apache Axis client library. The framework API presents an abstraction of the back-end MVDBMS functionality. The framework API is invisible to the user but is visible to the application programmer during the course of creating the application.

8. SERVICE OVERVIEW

It is difficult to assign a meaningful name to this component of the development framework. As the component draws its principal functionality from Web Services and is a Web Service from the perspective of the Client then we will call it the Service. It is the function of the Service component to provide location transparency (Schmelzer et al 2002). It hides the communication tasks between the Client API and the Server so that the Client can access methods and resources on the MVDBMS Server as though they were local to the client.

During the command/response cycle commands from the Client API are marshalled, encoded into SOAP, transmitted to the Server translated into MVDBMS actions and any results sent back through the same process. However in the normal course of operation this part of the framework would be invisible to users and application developers.

The Service comprises of:

Apache Tomcat
For the purpose of establishing the development framework Tomcat will be installed as a stand-alone server in accordance with the included instructions. In a public release of the development framework it might be appropriate to create a software wrapper to contain Tomcat together with the other framework middleware components. This would provide better security and avoid administration. However there may be occasions when the server cannot reside on the same host as the MVDBMS. In these cases Tomcat would be deployed on a separate host and used as a proxy to the MVDBMS. Therefore the development framework middleware would need to exist as a Tomcat Servlet.

Apache AXIS
Like Tomcat, AXIS can be used in two modes; either as a Servlet or as a function library for an application. Again for the purpose of establishing the development framework AXIS will be installed and run as a Servlet which will process SOAP messages and interact with the Java Web Application. In a public release of the development framework the AXIS library would certainly be wrapped in code to mask the install complexities and to provide better security.

Java Web Application
The Web Application provides an abstraction of the communication process between the Service Interface and the MVDBMS backend. Together with Apache AXIS this combination effectively translates a SOAP Remote Procedure Call (RPC) into an action on the MVDBMS host and returns the result to the Client API as a SOAP response.

During normal operation each client session would have its own Web Application instance which is enabled by Apache AXIS Sessions. Each Web Application instance is paired with a MVDBMS Specific Server instance on the Server.

As we have seen there are many MVDBMS versions from a number of vendors. Unfortunately whilst most MVDBMS share common language and filing features there is no common standard for implementing network communications. Vendors have implemented network communications differently on their MVDBMS. The Web Service component provides a unified and abstracted interface to the Client API with the MVDBMS specific communications handled by an instance of the Java Service interface and an MVDBMS Specific Server running on the MVDBMS.

Java Service interface

It is expected that a Java Service interface will be needed for the each of the different MVDBMS. In some cases, such as Raining Data D3, it will be necessary to supplement the Java Service interface with additional Java and MVDBMS BASIC code to enable reliable communications.

These MVDBMS specific Service interface versions would be distributed in a single framework package. The appropriate interface will be instantiated at the moment of connection according to the MVDBMS in use.

Server Overview

The Server is a logical view of the MVDBMS and its host. We should keep in mind that the many MVDBMS are available on many different server platforms. In certain cases a specific MVDBMS can have different behaviour on two different platforms. For example: D3 on Linux and D3 on Windows. Other MVDBMS brands such as UniVision have identical behaviour.

MVDBMS Specific Server

Although this component is illustrated by a single box in reality it could comprise of a number of MVDBMS BASIC programs residing on the Server. Whatever the implementation the function of the MVDBMS Specific Server is to execute Client API commands and return any results.

We should note that different vendor versions of MVDBMS whilst sharing a common core of functionality may handle some extended commands differently. A desirable design feature would be to wrap extended commands in order to always return the identical results from different MVDBMS. Equally it would be desirable to allow the application programmer to access MVDBMS specific features if that is required.

Application business logic

These are the MVDBMS BASIC programs hosted on the server which perform necessary business processing. The so called "business rules". This layer of code is supplied by the application developer and might very well be extracted from a legacy application.

Application data files

Naturally this is the data and files created for the application by the application developer. MVDBMS applications can host more than one data collection. In RDBMS parlance collections are referred to as a database but in MVDBMS they are known as an "Account".

Service Protocol

We have established that during the course of a session commands and responses are passed between the Client API and the MVDBMS Specific Server. Along the way these commands are translated to and from SOAP messages and are required to traverse a number of platforms, languages and network protocols. Fortunately since MVDBMS BASIC is an un-typed language the content of these messages will consist of (ASCII) characters. However it is still necessary to create a protocol and command syntax in order that both sides of the service can communicate.

Required Development

From this breakdown we can now identify the areas of development for the framework along with a number of interesting challenges:

Development

- Service Protocol
- Java Service Interface
- Java Web Application
- MVDBMS Specific Server
- Client API

Challenges

- How to develop a suite of interdependent programs based upon two different languages and four different programming technologies?
- How to maintain state in the stateless world of Web Services?
- How to maintain record locking from a remote client?
- What protocol can be used between Client API and Host Exec?
- How to implement file handles for a remote client?

- How to model multi-dimensional data in the two dimensional data world of Java? (XML was considered.)

Naming the framework

As design and coding had begun a project name was required. As the framework would deliver a Java API for Multi-value databases MVJAPI (Multi-Value Java Application Interface) was adopted. Though predictable and dull the name was available.

Development

There is insufficient space to discuss the entire development of MVJAPI. As the reader can imagine there was a lot of work involved. Instead perhaps it would be helpful to examine aspects of development that were a challenge or that were particularly interesting. It is hoped that these will provide an insight to the development process as a whole.

Development was undertaken on a single machine (laptop) equipped with Windows XP Pro, Sun Microsystems Java, UniVision, D3 and Eclipse. Since UniVision and D3 are capable of exposing their BASIC source code to the underlying operating system this allowed Eclipse to be used for all editing including the MVDBMS BASIC. Console sessions were opened to the appropriate MVDBMS to enable any required administration or running of test programs.

It was necessary to adopt an organic method of development. The author has many years programming experience at various levels with MVDBMS BASIC. However, only a modest grasp of Java and no knowledge of Web Services. These aspects of development would pose a fairly steep challenge.

Before any low level design could take place, key aspects of functionality were identified. These were isolated, researched and small test projects developed to investigate and establish the various working principals. Once small test projects were completed and working they were commented and put aside for later reference. Where appropriate larger projects were researched and created to establish the interdependency and interaction of the functional components.

These key areas of functionality include:

- Interfacing Java with UniVision
- Interfacing Java with D3
- Creating a trivial AXIS application
- Supporting "Sessions" in AXIS Web Applications
- Creating and running background process on D3
- Creating reliable network socket communications on D3
- Running a background process in a specific account on D3
- Preventing Java from translating high-bit ASCII codes to Unicode
- Implementing a Dynamic Array Structure in Java
- Translating a Dynamic Array message string into a Java Dynamic Array instance

Testing

Testing was a continuous part of the development phase. In most cases components were thoroughly tested before integrated development could progress. Test programs were devised, written and retained in order that components could be re-tested in the event of rogue behaviour occurring in an assembly of components. Three modes of testing were used depending on the nature of the component or assembly being developed:

- White box Testing
- Black box Testing
- Code Review

White box Testing

White box testing requires visibility of the source code and understanding of its internal function. During development of Web Service components the ability to view console messages is extremely limited. Various logging and data capture methods were temporarily written into the code to analyse component behaviour.

Black box Testing

Black box testing requires no visibility of the source code or understanding of the internal function. Components or assemblies are tested by an external

method according to their published behaviour. It is customary to pay particular attention to arguments and results that exceed or cross acceptable limits of input and output.

Code Review
Code review is simply the process of reviewing previous code. This is especially helpful in identifying message or obscure logic errors in code written a couple of days ago.

Service Protocol

The Service Protocol describes the message structure for passing commands and returning responses. After examining a number of protocols it was decided to adopt a simple character based protocol which comprised of a numeric command followed by any required arguments. Elements of the message would be delimited by a reserved ASCII character.

A number of tests were written to examine the communications between the Java Client and the MVDBMS BASIC back-end. It was determined that only 7-bit ASCII characters were needed. High-bit ASCII characters were passed without modification but low value ASCII characters, or control codes would not. MVDBMS uses the three ASCII characters 254, 253 and 252 as database record delimiters which would be passed without modification. This meant that MVDBMS Dynamic Arrays would be passed without any special handling needed. Thus ASCII character 250 was identified as being safe to use as the message delimiter.

Drawing on analysis of the MVDBMS BASIC command reference and many years of MVDBMS programming experience, a list of commands was created and assigned arbitrary codes. The actual codes assigned were not of great importance provided that the Client API and the MVDBMS Specific Server used the same codes. Table 1 lists the MVJAPI Service Protocol command codes.

Command	Code	Action
CONNECT	10	Connect to a specified MVDBMS.
DISCONNECT	11	Disconnect from current MVDBMS session.
OPENFILE	20	Open a file for read/write operations.
CLOSEFILE	21	Close a file and release file handle resources.
READ	30	Read an item (record).
READU	31	Read and item (record) with an update lock.
READV	32	Read a single attribute from an item (record).
READVU	33	Read a single attribute from an item (record) with an update lock.
WRITE	40	Write an item (record) and release any update lock.
WRITEU	41	Write an item (record) but maintain an update lock.
WRITEV	42	Write a single attribute to an item (record) and release any update lock.
WRITEVU	43	Write a single attribute to an item (record) but maintain an update lock.
SELECT	50	Execute an Access/English statement to generate an active select list.
READNEXT	51	Read next available value from the current select list.
KILLSELECT	52	Kill the current select list.
EXECUTE	60	Execute a TCL command or catalogued BASIC program.
CALL	70	Call an external catalogued BASIC subroutine.

OCONV	80	Perform an OCONV().
HANDSHAKE	90	Perform a handshake with version checking.

Table 1 - MVJAPI Service Protocol command codes

Following execution of a command the MVDBMS Specific Server would return a response which would comprise of an error code followed by any results. The first element of the reply is the appropriate error code with any error message or relevant details given in the following elements. Table 2 lists the MVJAPI Service Protocol error codes.

Error	Code	Meaning
ERROR	0	Error encountered whilst executing the last command.
OK	99	Success whilst executing the last command.
FILE.NOT.OPEN	100	Unable to open the specified file.
FILE.COUNT.EXCEEDED	101	Unable to open any more files.
ITEM.NOT.FOUND	110	The item (record) to read does not exist.
ITEM.LOCKED	111	The item (record) to read is locked for update by another process.

Table 2 - MVJAPI Service Protocol error codes

Client API

Conflict was inevitable in designing a Client API in Java which needed to represent MVDBMS BASIC operations.

- Java is an Object Oriented language whereas MVDBMS BASIC is a procedural language.
- Java demands strongly typed variables, but MVDBMS BASIC depends on context rather than typing to determine the nature of a variable.

There are architectural and syntactical differences between the two languages. However there are also important differences in design patterns. (Design patterns are the way of doing things in a programming language.) It is probable that adopters of the MVJAPI development framework are likely to be either MVDBMS programmers who have modest proficiency in Java or Java programmers with little or no experience of MVDBMS. It was important to conform to the established practice for Java programming and necessary to present MVDBMS programmers with an API that is both familiar and intuitive. It is hoped that experienced MVDBMS programmers will gain leverage from a familiar API which will compensate for their lack of Java skills, or that they can extend their understanding of MVDBMS operations to make better use of an establish Java programmer.

After consideration of the arguments between fat client and thin client the decision was made to give the application programmer the freedom of choice in this matter.

The minimum core functions were identified as necessary to the Client API:

- Connection
- Read/Write functions
- Dynamic Array manipulation
- Execution of commands and programs

After working through a progression of test models the overall structure of the Java API Classes was established:

MVJConnection	A class which represents a connection session together with methods and properties appropriate to an MVDBMS session.
MVJArray	A Java implementation of the MVDBMS BASIC Dynamic Array structure.
MVJFile	A class which represents read/write operations to and from MVDBMS files.
MVJ	A library of static definitions (including command and delimiters) and helper methods.

The MVJArray and MVJFile classes are worth discussing in further detail.

MVJArray: Dynamic Array Class

MVDBMS share a number of important features from the PICK model. Data operations are not constrained to the well known Database/Table/Columns/Rows model implemented in RDBMS. Instead data is represented by a textual structure known as an Item. Items are stored in Files and each Item has an identity or Item ID that is unique to that file. These Items have a data structure that consists of Attributes, which are analogous to RDBMS rows, Values which are analogous to RDBMS fields and Sub Values for which there is no RDBMS equivalent. If manipulating this structure as a raw string we find that these structural elements are delimited by single ASCII codes to determine their hierarchy. Attributes are delimited by ASCII 254, Values are delimited by ASCII 253 and Sub Values are delimited by ASCII 252. Whilst the RDBMS data model is analogous to a spreadsheet, the MVDBMS data model has a data structure similar to an XML document. (Raw XML documents can be stored in MVDBMS data files with no special handling or modification.)

Popular development languages such as C++, Java and C# have support for the Table/Query/Record Set components required for RDBMS. They do not support the multi-dimensional data model of the MVDBMS. In order to bring the MVDBMS data structure to the Java API it is necessary to create a collection of classes that model the MVDBMS Dynamic Array.

Dynamic Arrays are an important feature of the MVDBMS BASIC language. They are the mechanism by which multi-dimensional data is transported and manipulated. Dynamic Arrays grow and shrink as required and support operations to insert, delete, extract and assign values.

Consider the following program snippet:

```
SomeVariable = "Hello World"
```

Following execution we now have a variable called "SomeVariable" which contains the string "HelloWorld". However in MVDBMS BASIC we can go further:

```
SomeVariable<2, 1> = "Good morning"
```

Following execution of this line the string variable has now become a Dynamic Array which contains the two data attributes "Hello World" and "Good morning". Inspection of the raw data would reveal that a single multi-value data element appears exactly the same as a single attribute data element:

```
Raw data:
<1>    Hello World
<2>    Good morning
```

Continuing with the exercise:

```
SomeVariable<2, 2> = "Good evening"
```

Our Dynamic Array still contains two attributes but the second attribute now has two values which are "Good morning" and "Good evening". Now inspection of the raw data will show the multi-values appearing on attribute 2.

```
Raw data:
<1>    Hello World
<2>    Good morning]Good evening
```

Using the <> operators with Dynamic Arrays we can insert, replace, delete and extract

the three basic data elements of MVDBMS; attributes (AM), multi-values (VM) and sub-values (SM). This behaviour may seem strange but is exactly what the experienced MVDBMS programmer will expect. An important consideration when designing the Client API was how to implement a Java class which provided programmers who use this development framework with a familiar data model with which to work and how to implement this multi-dimensional data storage object within the two dimensional world of Java?

Exploration and experimentation of the Java language revealed that Vectors were the closest data model to MVDBMS Dynamic Arrays. "(Vectors) *implement a grow-able array of objects*." (Sun Microsystems 2006). Java Vectors provide indexing and iteration based upon having an element zero whilst Dynamic Arrays begin at attribute 1. It was therefore necessary to write a wrapper class which would either ignore element zero or to invisibly adjust the indexing. It was decided to ignore element zero and reserve it for future use. For example it could be used to hold an instance of a class which determines the type of Dynamic Array sub-structure being implemented; Attribute, Value or Sub-Value.

The MVJArray class implements MVDBMS Dynamic Array text based functions. Future development of the class should enable complex insert or replace operations where a Dynamic Array can be inserted or appended to another Dynamic Array. As far as possible the inner workings of the MVJArray class have been hidden from the application developer. However some aspects of the Java language such as strong typing of primitives were unavoidable.

It is anticipated that there will be occasions when application programmers may need to perform Dynamic Array operations separate from read/write operations. For this reason the MVJArray Dynamic Array class is available without needing to reference a parent class.

MVJFile: Read/Write Class

The MVJFile read/write functions are important operations in the Client API. These functions are responsible for retrieving and saving data. MVDBMS BASIC contains a rich array of read/write functions and naturally the approach for using these functions is appropriate for a procedural language. To translate these functions from MVDBMS BASIC to Java Classes was once again a challenge.

As an example of the programming differences let us consider the elementary programming pattern of updating a record from a file as illustrated the following code snippet.

```
1   OPEN "MYFILE" TO F.MYFILE ELSE STOP 201,"MYFILE"
2   READU MYARRAY FROM F.MYFILE, "MYRECORD" LOCKED
3   *  Code to handle record locked for update by another process
4     …
5   END THEN
6   *  Code to update the record
7     …
8   END ELSE
9   *  Code to handle record not found. Create new record?
10    …
11  END
12  WRITE MYARRAY ON F.MYFILE, "MYRECORD"
13  CLOSE F.MYFILE
```

In line 1 we open a file to a file variable. The file name is expressed as a literal string. Whilst custom dictates that file variables begin with "F." we can use any non-reserved word. If the file cannot be opened the program execution reports error message 201 with the file name and halts.

In line 2 the READU command attempts to read a record called "MYRECORD" into the given variable and lock the record against other simultaneous updates. The three possible outcomes to a READU operation are; LOCKED when another process already has an update lock on the record, THEN when the record is read and ELSE if the item does not exist. The programmer can insert statements to handle these events accordingly.

Finally on line 12 the updated record is written back to the file and any update locks release. The file remains upon until explicitly closed in line 13 or the program terminates.

There are additional points to note which are not immediately apparent from the example.

- MVDBMS BASIC does not require the programmer to declare or type variables.
- Programmers can use any name for variables provided the name does not clash with reserved names.
- All data is available within a program as text or strings. Numbers are stores as text. Any casting occurs in response to context (arithmetical operations) and is usually invisible to the programmer.
- Data within a variable becomes a Dynamic Array by virtue of having multiple attributes (fields), values (sub-fields) or sub-values (sub, sub-fields).

Error! Reference source not found. illustrates the same MVDBMS BASIC example as we might implement it a Java Client API. Whilst the flow of the Java code in the illustration is inappropriate for Object Oriented Programming it does allow a direct comparison between the programming approaches for MVDBMS BASIC and the Java Client API.

```
1  // Create a MVJConnection instance
2  MVJConnection conn = new
   MVJConnection(MVJLib.MVTYPE_UV);
3  try {
4    conn.connect(host, account,
   password, user);
5
6  } catch (MVJConnectionException e) {
7    // Code to handle connection failure
8
9  }
10
11
12 // Open file
13 try {
14   MVJFile myfile =
     conn.openFile("MYFILE");
15
16 } catch (MVJFileNotOpenException e) {
17   // Code to handle failure to open file
18
19 }
20
21
22 // Read record from file with update
   lock
23 MVJArray myArray = null;
24 try {
25   myArray =
   myFile.readu("MYRECORD");
26   // Code to update the record
27   …
28
29 } catch (MVJFileReadLockedException
   e) {
30   // Code to handle record locked for
   update by another process
31   …
32
33 } catch (MVJFileReadFailException e) {
34   // Code to handle record not found.
   Create new record?
35   …
36
37 }
38 myFile.write(myArray, "MYRECORD");
39
40 myFile.close();
```

In lines 1 to 9 we create an instance of MVJConnection. This is the root class which provides a path between the Client and the multi value database host. This step is not necessary in the multi value BASIC example since the code would already be running within an account on the multi value database server.

Lines 12 to 19 open a file on the multi value database host and make a handle available in the API. If the file cannot be opened, perhaps because the name is wrong or changed, then an exception is thrown and the file object will be null. The intention in binding the creation of subordinate API class instances, such as MVJFile, to the MVJConnection object that the relationship and dependence of subordinate classes to the parent object are reinforced in the minds of the programmer.

The READU operation posed an interesting Java code challenge. As discussed above the READU operation has three possible outcomes and is always expected to return a Dynamic Array even if it contains null. Lines 22 to 37 present an elegant Java solution using exceptions. MVJFile read operations always return an MVJArray Dynamic Array object, but throw an exception depending on the outcome of the read operation. The application programmer is then free to take whatever action is appropriate depending on the intentions of the operation. In this way the application developer is presented with a familiar and intuitive API whilst maintaining a programming approach appropriate to Java.

Java Web Application

The Web Application component provides a gateway into the MVJAPI Service. This area of the framework development was expected to present the least work since it draws largely on established, well documented, third-party components. However, this area of development nearly caused a failure to complete the project.

AXIS documentation suggests two approaches for handling Web sessions: One approach is to use HTTP cookies whilst the other approach is to use SOAP headers. Since the MVJAPI development framework is not browser based HTTP the appropriate choice was to use SOAP headers to maintain sessions.

The topic of Web Services in general together with Apache Tomcat and AXIS in particular was researched. Sources included the Apache documentation, text books and articles on the Internet. As Web Services is an area of rapid and continual development it was found that the available documentation and tutorials were usually lagging behind the current version of software and utilities. Nevertheless, development of Web Services test components was progressed in logical steps, starting with the ubiquitous "Hello World".

After much work it was discovered that the available documentation for implementing sessions within Apache AXIS was incorrect and that one of the software tools emitted an incorrect configuration file. These findings were reported back to the Apache AXIS author and the Web Services project development continued.

In order to implement sessions in AXIS it is necessary to specify the service scope as "session". This is done by adding a <parameter> statement to the <service> element in the WSDD deployment file.

```
    <parameter…
    <parameter name="scope"
value="session"/>
  </service>
</depoloyment>
```

Available documentation insists that it must be installed for both requester and response on the client and server sides.

ON THE SERVER:

- *The REQUEST is checked for a session ID header. If present, we look up the correct SimpleSession. If not, we create a new session. In either case, we install the session into the MessageContext, and put its ID in the SESSION_ID property.*

- *The RESPONSE gets a session ID header tacked on, assuming we found a SESSION_ID property in the MessageContext.*

ON THE CLIENT:

- *The RESPONSE messages are checked for session ID headers. If present, we pull the ID out and insert it into an option in the AxisClient. This works because a given Call object is associated with a single AxisClient. However, we might want to find a way to put it into the Call object itself, which would make a little more sense. This would mean being able to get to the Call from the MC, i.e. adding a getCall() API (which would only work on the client side)....*

- *When REQUESTS are generated, we look to see if an ID option is present in the AxisClient associated with the MessageContext. If so, we insert a session ID header with the appropriate ID.*

(Jax Systems LLC, 2005)

Thread safety is usually a concern in Web Services but not when implementing sessions. When using session scope the web service container (Tomcat) creates a new instance of the service for each request. Thread safety is an issue for "request" and "application" scopes where one instance of the web service is shared by all requests.

9. EVALUATION

Proof of concept

As stated above continual testing was a part of the development process. In this respect the MVJAPI application framework was evaluated and proven. However as a RAD tool, proof of concept required the successful creation of a trivial client-server GUI application.

The aim was to create a single screen GUI application with which to store basic book details on the MVDBMS server. An idea borrowed from the Microsoft Visual Basic "Northwind" demo application.

The demo application has four fields:

- ISBN
- Title
- Author
- Publisher

The ISBN is unique to each book and serves as the MVDBMS Item Id. The application also has four controls:

- Connect
- Next
- Save
- Exit

The Connect button connects the application to the MVDBMS resource using hard-coded details. In real-world applications a dialogue box would be generated in which the user would supply all necessary connection details. On clicking the Connect button the demo application performs the following tasks:

- Connects to the MVJDEMO database MVDBMS resource
- Disables the "Connect" button
- Enables the "Next" and "Save" buttons.
- Open the "BOOKS" file
- Executes a selection statement to select all the book records currently in the file
- Read the first ISBN from the active select list.
- If an ISBN was retrieved then read the record from the open file and populate the text box with the details. If no ISBN was retrieved then clear the form ready to accept new details.

The user may then input or amend the details as required then click the "Next", "Save" or "Exit" buttons.

- Clicking the "Save" button sends the current record details to the MVDBMS to write them to file.
- Clicking the "Next" button retrieves the next ISBN from the currently active select list. If an ISBN was retrieved then the details are read and the text boxes populated with the details. If no ISBN was retrieved then the form is cleared ready to accept new details.
- Clicking the "Exit" button disconnects from the MVDBMS resource and terminates the demo application.

In real applications it would be necessary to provide appropriate validation and error handling. These niceties were not required to establish that the development application framework works as required.

10. CONCLUSIONS AND IMPLACATIONS FOR FURTHER RESEARCH

The development framework presented here is one of many approaches that would meet the needs of MVDBMS developers needing to move their applications forward with a high quality GUI. With so much momentum behind software design reliant upon Relational Database it is doubtful that new adopters will find the solution attractive.

The design and selected technologies exceed the criteria listed above and it is difficult to imagine that this development framework is without commercial merit. It does however require further research and investment before it can be offered as a commercial or Open Source solution:

- Support is required for all suitable MVDBMS.
- The Client API needs features to handle Administration and Security.
- A reporting package is required. This could be implemented as an Eclipse "plug-in" using a visual designer interface that maintains a similar look and feel to SWT Designer.
- The Web Service component needs to be wrapped in a self contained Tomcat Servlet. This will simplify installation and provide better security.
- Examine the use of XML to transport MVJAPI Dynamic Arrays. This may enable Java and .NET programmers to use DOM and SAX libraries to manipulate the data if preferred.
- The Web Service should implemented over Secure Socket Layer (SSL) networking to provide better security.

11. REFERENCES

Bend, John George. (2006) *A RAD framework for cross-platform GUI development of MVDBMS client-server applications. Liverpool University.*

Apache Foundation. Tomcat http://tomcat.apache.org/

Apple Computers Inc, (2006). *Mac OS X released on Intel.* http://www.apple.com/macosx/

Automatic Data Processing Inc. (2006) http://www.adp.com/

Bernstein, Philip A. Middleware (2006) *A Model for Distributed Services. Communications of the ACM*

Bourdon, Roger J. (1987). *The PICK Operating System. A practical guide.* Addison-Wesley. ISBN 0201180553

Carnegie Mellon, Software Engineering Institute, *Software Technology Roadmap.* http://www.sei.cmu.edu/str/descriptions/middleware.html

Chappell, D. A., Jewell, T. (2002). Java Web Services. O'Reilly. ISBN: 0596002696

Computer Weekly. (January 2002) *Council Saves with open source.* http://www.computerweekly.com/SiteMapArticle/Articles/2002/01/31/c1056217/184897/Councilsaveswithopensource.htm

DM Review (2002) http://www.dmreview.com/editorial/dmreview/print_action.cfm?articleId=5736

Eclipse Org. *SWT graphical toolkit.* http://www.eclipse.org/swt/

Garrett. Jesse James. (2006) *Ajax – A new approach to Web Applications.* http://www.adaptivepath.com/publications/essays/archives/000385.php

Gnome Desktop http://www.gnome.org/

Instantiations, *SWT-Designer* http://www.swt-designer.com/

Guojie, Jackwind Li. (2005). *Professional Java Native Interfaces with FSWT/JFace*. Wrox. ISBN: 0470094591

Jax Systems LLC. (2005). *Docjar.* http://www.docjar.com/docs/api/org/apache/axis/handlers/SimpleSessionHandler.html

KDE Desktop http://www.kde.org/

Mono Project, *Mono* http://www.mono-project.com/Main_Page

Novel, *Ximian Desktop* http://www.novell.com/linux/ximian.html

OSI, June 2003. *West Yorks Police pilots Linux desktops* http://www.osix.net/modules/article/?id=238

Sisk, Jonathan E. (1990) PICK For Professionals. *Advanced Methods and Techniques.* TAB Books Inc. ISBN 0830601252

Snell, J., Tidwell D., Kulchenko P. (2002) *Programming Web Services with SOAP.*

O'Reilly. ISBN: 0596000952

Tincat Group Inc. (2006)

http://www.w3schools.com/browsers/browsers_stats.asp

12. BIBLIOGRAPHY

Bequet, H., Kunnumpurath, M.M., Rhody, S., Tost, A. (2002). *Beginning. Java Web Services.* WROX. ISBN: 1861007531

Brittain, J., Darwin, I.F. (2003). *Tomcat: The Definitive Guide*. O'REILLY. ISBN: 0596003188

Bourdon, R. J. (1987). *The PICK Operating System. A practical Guide.*

Addison-Wesley. ISBN: 0201180553

Chappell, D. A., Jewell, T. (2002). *Java Web Services.*

O'Reilly. ISBN: 0596002696

Deitel, H. M., Deitel, P. J. (2003). *Java. How To Program. Fifth Edition*.

PRENTICE HALL. ISBN: 0131016210

Deitel, H.M., Deitel, P.J., Nieto, T.R. (2000). *Internet & World Wide Web. How To Program.*

PRENTICE HALL. ISBN: 0130308978

Dougherty, D. (1989). *Pick ACCESS. A Guide To The SMA/Retrieval Language.* O'Reilly & Associates. ISBN: 0937175412

Gallardo, D., Burnette, E., McGovern, R. (2003). *Eclipse In Action.* MANNING. ISBN: 1930110960

Graham, S., Davis, D., Simeonov, S., Daniels, G., Brittenham, P., Nakamura. Y., Fremantle, P., Konig, D., Zentner, C. (2005). *Building Web Services With Java. Making Sense of XML, SOAP, WSDL, and UDDI.*

DEVELOPER'S LIBRARY. ISBN: 0672326418

Graham, S., Simeonov, S., Boubez, T., Davis, D., Daniels, G., Nakamura, Y., Neyama, R. (2002). Building Web Services With Java Making Sense of XML, SOAP, WSDL, and UDDI

SAMS. ISBN: 0672321815

Gurewich, N., Gurewich, O. (1997). *Teach Yourself Visual Basic 5 In 21 Days. Professional Reference Edition*.

SAMS Premier. ISBN: 0672311763

Guojie, J.L. (2005). *Professional Java Native Interfaces with SWT/JFace.*

WROX. ISBN: 0470094591

Hunter, J., Crawford, W. (2001). *Java Servlet Programming.*

O'REILLY. ISBN: 0596000405

Pick Systems. (1995). *Advanced PICK Reference Manual.*

Pick Systems.

Ray, J., (1999). *Special Edition Using TCP/IP.*

QUE. ISBN: 0789718979

Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D. F.(2005). *Web Services Platform Architecture. SOAP, WSDL,*

13. APPENDIX

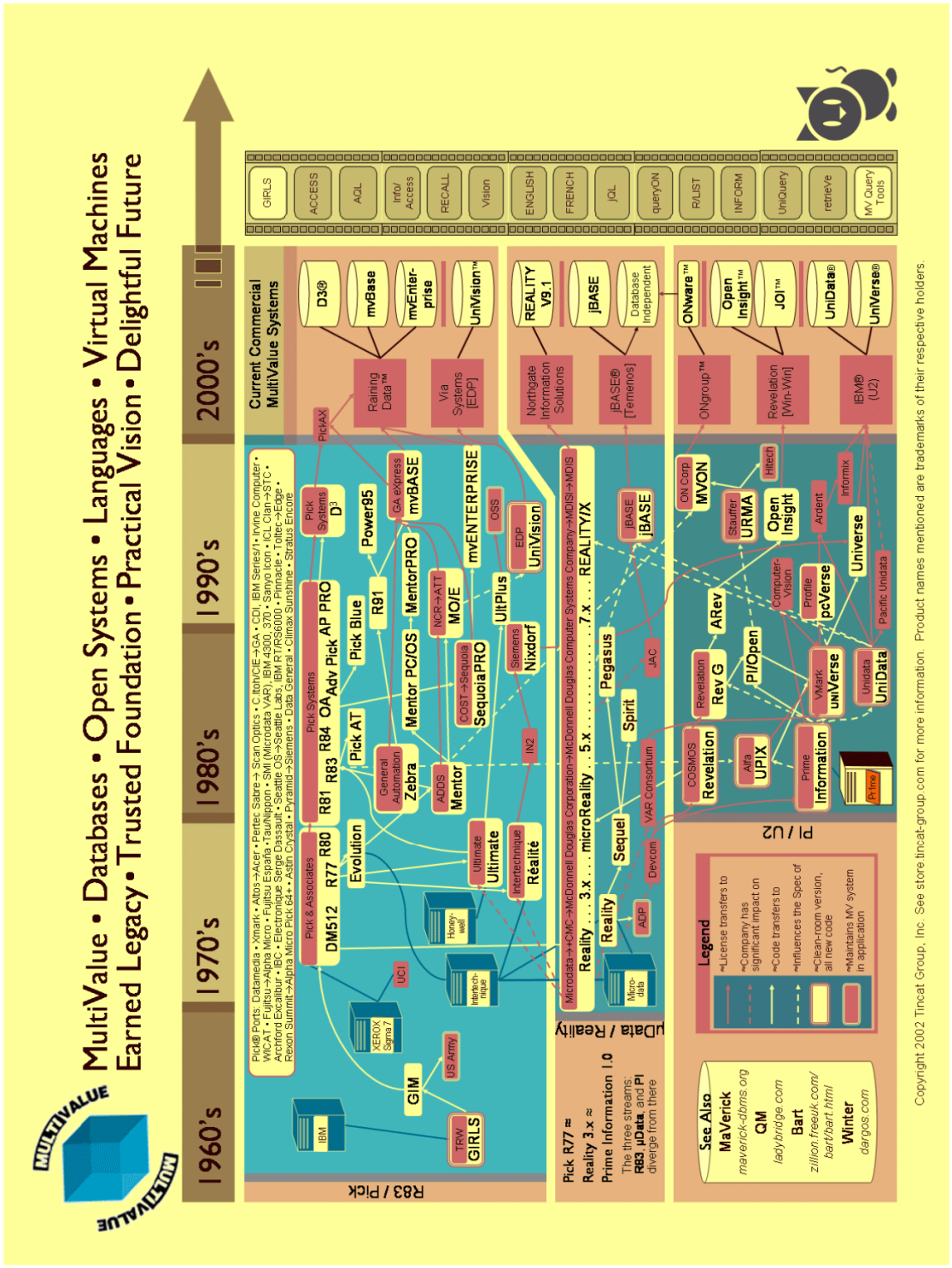

Figure 1 -Value database family tree (Tincat Group Inc., 2006)

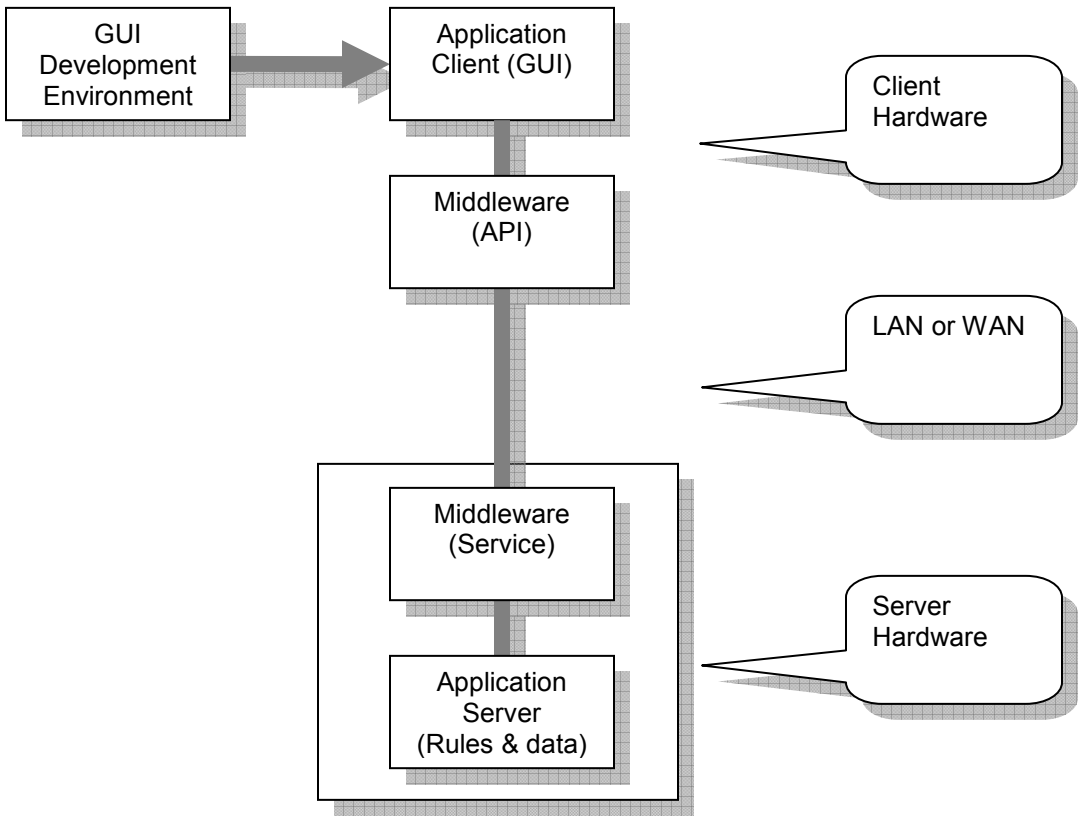

Figure 2 - Principal components of the development framework

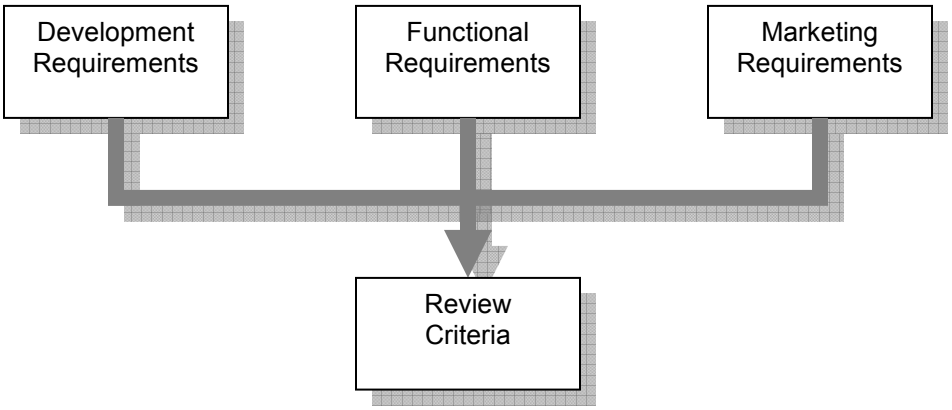

Figure 3 - General Requirements

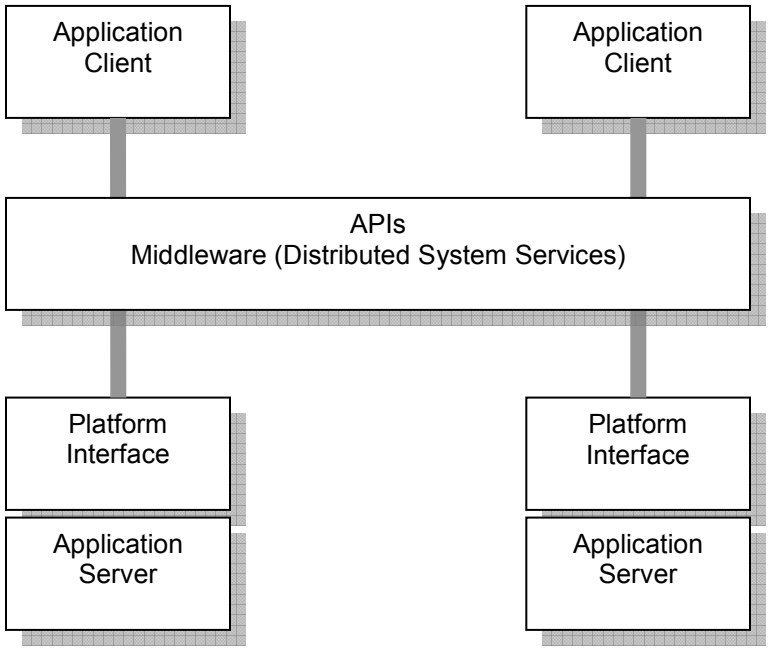

Figure 4 – Bernstein Middleware Services model

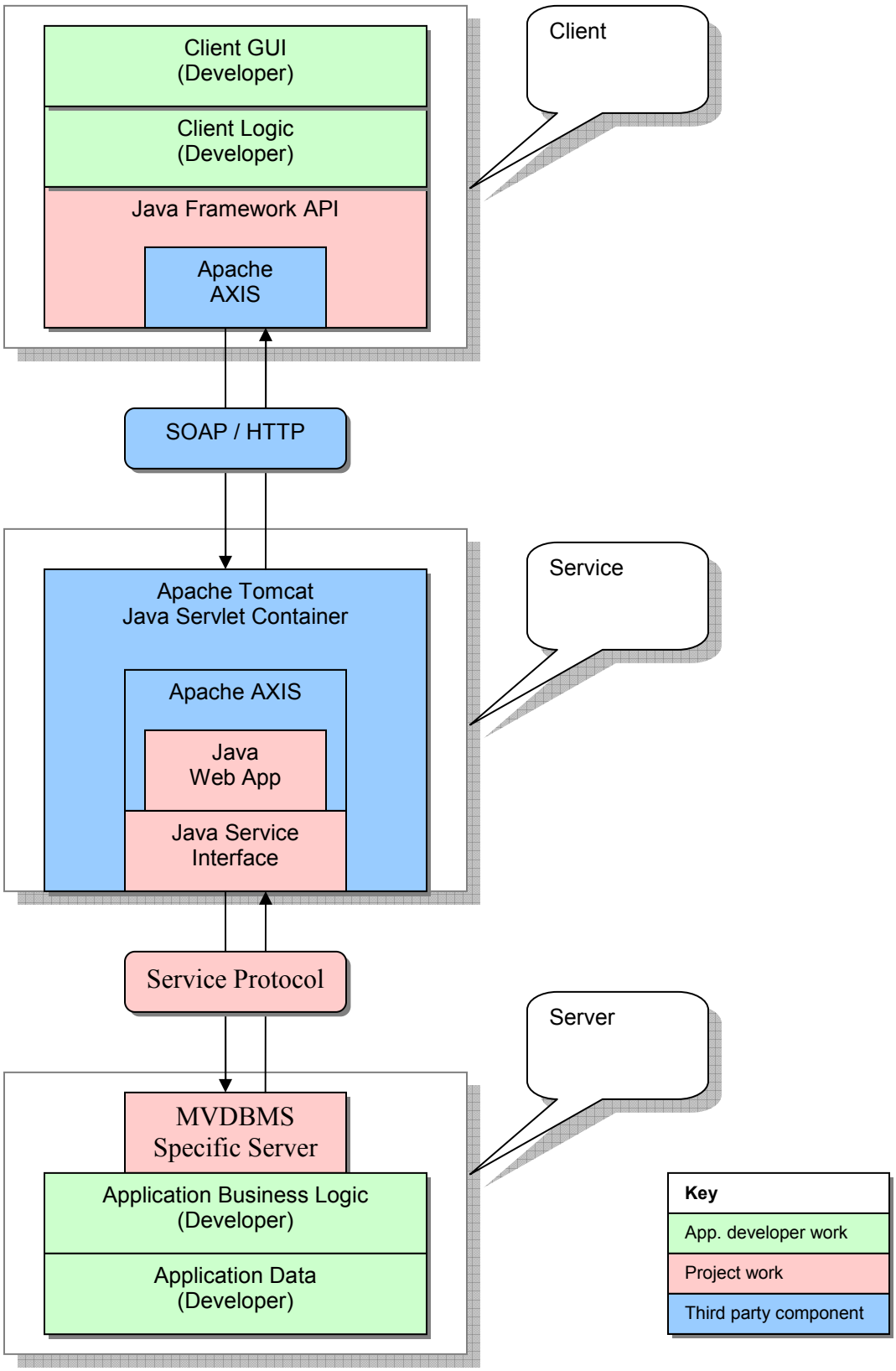
Client GUI
(Developer)
Client Logic
(Developer)
Java Framework API
Apache
AXIS
Client
SOAP / HTTP
Apache Tomcat
Java Servlet Container
Apache AXIS
Java
Web App
Java Service
Interface
Service
Service Protocol
MVDBMS
Specific Server
Application Business Logic
(Developer)
Application Data
(Developer)
Server
Key
App. developer work
Project work
Third party component

Figure 5 – Technological overview of the development framework