STEPHEN PAUL CLARK

IT SKILLS AND QUALIFICATIONS

Visual C++ *including*	C/C++; MFC (in-depth); Multi-threaded programming; Client-Server; Winsock; Win32 SDK; Enhancing and overriding MFC; ODBC; WinInet; Web Services through gSOAP; XML; XML Schemas; XML Parsing; Tibco Rendezvous; Websphere MQ; STL; VC 6; VS 2005; Windows Services
Less recently	IIS Extensions through ISAPI and MFC; ATL; COM; Winsock through MFC; Win32 Serial Communications; Owner-draw controls; Dual-platform (VeriFone terminal/Windows) coding; OLE Automation; TAPI; MAPI (Simple and full 32-bit)
Databases	Oracle; Ingres
Less recently	Microsoft SQL Server 6.5 and 7
Languages	C/C++
Less recently	Visual Basic; Pascal; Fortran; 80x86 Assembler; Sycero 'C'.
Operating Systems	Microsoft Windows variants – NT, XP, Server
Other Skills	System design; Data structure analysis & design, team lead, technical documentation; documentation design; software architecture

ACADEMIC QUALIFICATIONS

Degree	Lincoln College, University of Oxford. BA (Hons) Chemistry
'A' Levels	South East Essex Sixth Form College, 'SEEVIC', Benfleet, Essex. SMP Mathematics (A), SMP Further Mathematics(B), Physics(A), Chemistry(B).
'O' Levels	The Sweyne School, Rayleigh, Essex. Mathematics, Additional Mathematics, Physics, Chemistry, Biology, Music, History, German, English, English Literature.

WORK EXPERIENCE

Senior Developer, Technical Architect **BNP Paribas**	2001 to February 2011 Initially GUI developer on IRD trade capture system; then founder member of team to build new Credit Derivatives trade capture platform. Responsible for much of the design of the client-server system and became Technical Architect for the system. Moved into a purer architectural role as C++ specialist, then back into team lead to recover poorly performing project. Recently been designing a new client-server system. Full details attached.
"R& D Engineer" – ie, Software Developer **VeriFone Limited, a division of Hewlett Packard**	1999 to 2001. Developing applications that run in VeriFone's Omni2000 and Omni3300 ranges of payment terminals; also Windows based tools to configure the terminals. My work involved the development of a client/server, multi-lane, payments system through a proprietary framework that provides an ActiveStore interface to the application; this includes multi-threaded serial communications. Further details of projects at VeriFone are attached. I was a member of the German development team, but based in the UK.

Senior Developer **KAI Computer Services Ltd**	1993 to 1999. Writing business information software. Projects include Octopus (an activity management system) and various add-ons; Helpdesk; Retail Selling; Job Estimating; Workshop; On-line Recruitment. Project details given on further pages. Duties include the specification, design and implementation of product modifications and add-ons for specific client requirements. I have also been responsible for network design, implementation and support.
Chemistry Part II **Chemical Crystallography Lab., Oxford**	1992 to 1993. This was the final year of my degree, a research year that I spent writing software (in Fortran) to draw the voids found between the molecules in crystals.
Computer Room Officer **Lincoln College Oxford**	1989-90. During my second year at University, I supported the student computer rooms at College. Responsibilities including the installation and support of several standalone 8086 and 80286 machines.

HOBBIES AND INTERESTS

My main hobby is singing: I have lessons (and am taking exams) and sing with a local chamber choir. I am on the committee for the Welwyn Garden Concert Club, which stages professional chamber music concerts in Welwyn Garden City. I love to listen as well, and go to the opera regularly.

I also enjoy cooking. I am researching the food eaten in the past, particularly that described by Pepys in his Diary – with a view to cooking it! At some stage I will convince my garden to yield fresh vegetables for the pot.

I like to read; I'm interested in a wide range, from history and philosophy to classic fiction and literature. I enjoy travelling – I've recently been to China, Central Europe, Italy, Egypt and Jordan.

PROJECT DETAILS

BNP PARIBAS

DVS

June 2010 to February 2011

DVS is the risk generation platform used for Credit Trading. It drives the overnight price and risk batches for all credit products.

My task was to modify the trade snapshot to be stored in an Oracle database rather than a series of XML and SQLite files.

RISKCUBE / ON-DEMAND RISK

November 2009 to June 2010

In addition to the Façade project (described below) I was asked to produce the system design for a new server-based project – the RiskCube – which would allow us to define the Credit Risk measures calculated overnight by means of portfolio (at any level of the hierarchy), trade type, risk measure and scenario. This produces a four-dimensional data structure, but it is not an OLAP system as the Risk is defined at the highest level in the various dimensional hierarchies and overridden at lower levels as necessary. Another facet of the system is a live Dashboard to monitor the jobs running on the calculation grid, categorized by the same dimensions – a true OLAP scenario.

I have produced a system design for this, prototyped user interface and data imports, and analysed the data that would be manipulated. I discovered that the cube would be very sparse – using only about 10000 nodes out of a possible 48 billion. To store this efficiently I created an n-dimensional tree structure which can find the parents and children of nodes very quickly.

The RiskCube project has since transformed into the On-Demand Risk project, to allow traders to rerun part of the overnight risk at will. A new service – "On-Demand" – will provide the "command and control" structure for the other, existing, components in the system (the GUI, the calculation grid and the Reporting Engine). I am responsible for the overall system design, delegating the responsibilities of the various components, as well as the design and build of the On-Demand server. In effect, this is version 1 of the RiskCube.

The server design is fairly generic. Any server has a series of common components such as a listener, command router, log, cache and so forth. I have designed a series of interfaces in C++ that allow these to be implemented separately yet cooperate fully. The server can thus be built as a series of distinct components that interact in a well-mannered way; any component can be replaced at any time without requiring the remainder to be rebuilt. Any component can act as a command processor if so built, providing a full range of administrative capabilities, and there is no limit to the number of main command processors that can be attached to the system.

The sever is not limited in the number or type of network interfaces it can have, as the command processing knows little about them. You can attach many different kinds of listener such as HTTP (returning HTML), REST, XML-RPC, telnet, TibRv, Web Services if you wish. This allows a client application to use the interface that best suits it, rather than building a server that forces the client down a particular route. I am also working on an Excel plugin (XLL) that allows access to any of the server's functions within an Excel formula, but does not need to be updated to expose new functionality.

FAÇADE – DATA RETRIEVAL LIBRARY

January 2007 to Facade 2011

I was asked to take over the Façade project as team lead and architect.

The C++ library packages together access to a variety of data systems within the Bank and presents a common interface for them. This involves linking to a set of internal client libraries for some systems, and using open standards such as REST and XML-RPC for others. The library is used for data access mainly by the Credit analytics libraries and compute grid, and by several other teams within Credit IT. It has an Excel add-in interface that allows some of the functions to be invoked from Excel and the data to be returned directly into a spreadsheet.

When I took the project over, it had a poor reputation – features took some time to deliver and were then unreliable and slow, and it had a bloated memory footprint. There was also no documentation regarding the available functions or the data structures returned. I first created a proper test harness that allowed both interactive and scripted testing of the functions, and instituted an automated regression test for each formal build. The test harness also allows the data structures to be inspected.

One of the main causes of the speed problem was the unnecessary caching of data. This was removed and the caching layer rewritten so the data all goes into a central cache. This has since allowed the seamless addition of disk-caching (using Berkeley DB) and helps with data distribution around a Data Synapse based compute grid.

Reducing the memory footprint has created a challenge as it is mainly caused by a poorly designed data interface that cannot now be changed without rewriting all the client software. However, a good reduction was achieved by using a dictionary of field names rather than repeating them in every object. The data structures have some responsibility for the slowness as they use many small heap allocations that turn out to be expensive in terms of speed.

Another performance boost was achieved by replacing the four XML parsers in use with just one, developed by myself as the latest iteration of that developed for Focus (see below). By minimising the number of memory allocations used during the parsing, it produces a tree of nodes substantially faster than both MS-XML and Xerces. It also contains a simple XPath processor to extract data from the parsed XML. By taking care with the memory management and the C library functions used, it can then rebuild the XML at high speeds.

Replacing the ADO database access with ODBC not only simplified the code but improved performance by removing the COM overhead and the necessity to translate between BSTR and C strings. It also removed some inconvenient timeouts while waiting for complex queries to return. Creating database views simplified the most complex queries and provided another performance boost.

The software was ported from VC6 to VS2005, with dual builds for over a year.

Architectural research during this period included the development of flexible data structures, indexed by field names, that are both small and fast. Using a data-dictionary approach, they are designed so that serialisation of the data on and off the network, or in and out of a cache, requires very little extra processing. Another development was a server framework that separates the various layers of a server and makes it very easy to add or replace components, such as interfaces, caches or security, without rewriting the whole server.

Much of this was as background to a proposal to move Façade to a client-server application. This would have helped with distribution by having a wafer-thin client that did not need to be upgraded whenever new functions are added or data structures modified. It would also have enabled it to support many more client applications by having a variety of interfaces such as a closed, binary, C++ client for high-speed analytics applications, web-services for .NET and Java, Excel add-in, and so on.

Extra utilities included code to log a stack trace on access violation, rather than simply crashing, and to diagnose DLL load errors by analysing the DLL import tables.

Technologies: C++ (VC6 and VS 2005), STL, XML, HTTP, WinInet, Berkeley DB, ODBC, Oracle DB, Excel SDK, XML-RPC

Fixed Income/Transversal C++ Architect

September 2006 – December 2006

Following on from my time in Focus I was seconded to a role more focused on software architecture as part of a new team with responsibilities for architecture across a large part of the IT systems in the Bank. My particular sphere was C++, MFC and Windows development. This included

- assisting teams with questions,
- developing software to demonstrate principles,
- directly assisting teams with my time as required for investigating issues such as
    - supporting multi-threading environments in libraries that are not currently thread safe.
    - development of proof of concept applications
    - evaluation of compiler optimisations

FOCUS – Credit Derivative Trade Capture

September 2001 – September 2006

Focus is BNP Paribas's trade capture system for Credit Derivatives. Initially just handling long and short form vanilla default swaps, it has since been expanded to cope with exotic trades, the structures of which are designed at run-time. The system captures the data from the traders, passes it back to Middle Office for validation and then sends it direct to the Back Office settlement and documentation systems. It has external links for online trade confirmation, and can receive vanilla trades from the trading gateways. A further module allows the assessment of the impact of credit events on the trade portfolio.

Focus is a client-server product; the communications within the system are by Tibco/Rendezvous and those to other systems are by Websphere MQ. The system is completely written in C++. It runs internationally with a server and database in each location. Originally Focus used an Ingres database, but this was later migrated to Oracle. As we used ODBC to communicate with the database the migration was relatively painless.

I was a founder member of the Focus team and was responsible for the initial design and implementation of the "Deal Manager" (server) component: its data cache, threading model and the design of the wide area protocols to keep the system synchronised between locations. Thereafter my responsibilities included the addition of new features, maintenance and support.

Data are transferred within the MQ and RV messages as XML; to improve performance, I created an XML parser that tokenised the XML then deserialised directly into the objects rather than going via a DOM or using SAX.

Focus was initially conceived for vanilla trades but we needed to handle exotics as well. I designed and built a new framework for this called "Flex" that extended the application's object structure to include objects designed at runtime. A Business Analyst can use a drag-and-drop interface to create new objects from existing ones – both those built in to the system and others created in this way. This allows some quite complex structures to be built. After an object has been created it must be published for use; the system generates the database scripts for the initial creation and subsequent modification of the tables needed to store the objects' data. It also generates the bindings between the objects and the database.

Flex data types include the normal primitives and also enumerations and static lists that can be built and configured within the system. Lookups of centrally maintained static data are handled automatically when the appropriate object was dropped into the structure.

The user interface for a Flex trade is generated dynamically within the "Trade Ticket" tabbed dialog, following a layout configuration set up in the object definitions. Other configuration options include which fields are output in a compact XML data format sent to the risk and pricing systems.

For the last two years of my time on the team I was the Technical Architect of the system. During this time I designed, oversaw and implemented various technical enhancements to improve the code structure and the system's performance. These included improving the handling of Windows Services within the code, the development of a thread manager and increased support for multithreaded processing and support of a passive online backup Deal Manager. I conducted an architectural review and highlighted various aspects of the system that would need to be improved to handle the expected increase in trade volume then oversaw the first portion of the work.

Technologies: C++ in VC6, VS 2005; MFC; Tibco/Rendezvous; Websphere MQ; XML

MAD – MULTI-ASSET DERIVATIVES

March – September 2001

MAD is the trade capture system for Derivative trades within the Bank – Interest Rate, Equity and, initially, Credit (Credit Derivatives have now moved to a dedicated system). The software allows the user to build up a trade from a series of different instruments ("legs"); the trade can then be sent to the calculation farm to calculate price and risk.

My role was in the building and maintenance of the user interface. I added a new leg to capture data that was then priced outside the system, vastly reducing the development time when a new instrument was required. It was configured as a set of fields where the data could be string, numeric or selected from a list maintained within the system.

I also worked on enhancements for the Credit Default Swap leg. One allowed the production of printed trade tickets using a custom built form designer (report writers were not an option) and another generated term sheets as Word documents using OLE automation to drive a mail merge. Some basic STP from the Desk to Middle Office was added and automatic posting of simple trades to Back Office systems.

MAD is written in C++ using MFC.

VERIFONE

VERIFONE PAYMENTS APPLICATIONS

1999-2001

My role at VeriFone was the design and implementation of payments software – software that facilitates payments for goods by credit and debit card.

The project involved the customisation of a proprietary framework to provide a client/server payments system for a multi-lane ECR (electronic cash register) environment. This runs on Win32 platforms and is built in C++ using Visual C++. To provide best time to market, the existing payments software that runs on VeriFone's Omni3350 terminal was ported to Windows, principally by a library I wrote to emulate the Verix SDK that is used to program the terminals. Thus we have source-level portability of the software. The server side needed extensive work and, whilst working on this, I have had to develop a multi-threaded, multi-channel, serial communications server using Win32 serial communication APIs.

Both the client and the server side of the product are customised by the production of a set of plugins, which use COM to interface to the rest of the framework.

The terminal software, on which I worked prior to the Windows work, is written in C++ and is built and debugged using tools by SDS. The software is modular in structure, with the modules communicating across pipes. The task of the application is to gather bank card data from the user (either swiped or manually entered), package it up and transmit it with the amount and some other data to the authorisation host. This communication uses a variant of the ISO 8583 protocol.

I also created a Windows based configuration tool for the application, and my responsibilities included technical lead for modifications as well as creating and managing the schedules for the work.

KAI COMPUTER SERVICES

OCTOPUS

1996-1999

Octopus is an **activity management** system that takes contact management a step further and enables users to record activities against clients in the database, providing a full history of one's dealings with the client. I was the main programmer on the project with one or two others giving assistance as their own workload allowed. During this time I have rewrote most of the system to allow for modifications and amendments requested by clients. My full role on the Octopus project included specifying requirements (whether from a client or raised internally), designing suitable modifications and then implementing them.

Working on Octopus, I made use of the following technologies and skills to provide its multitude of functions.

- **ODBC** was used to link Octopus to its database, the API being wrapped with a thin class library. This made the code database independent and we had the system running, in a live environment, against FoxPro, Sybase SQL Anywhere and Microsoft SQL Server databases.
- **MAPI** and **VIM** linked Octopus to email systems such as MS Mail, cc:Mail and MS Exchange.
- **TAPI** provided links to phone systems enabling the autodial feature and Caller ID based data lookup.
- **COM** was used to encapsulate Octopus's word processor drivers. These, in turn, utilised **DDE**, **OLE Automation** and, when all else failed, sending keystrokes, to drive the word processors to perform a mail merge without further user intervention.
- The **Crystal Reports** engine is integrated into Octopus and was used to provide hard-copy from the database
- Octopus is fax-enabled. There is no device independent way to achieve this, so we built our own COM-based fax drivers that use each vendor's own APIs to provide automated faxing from Octopus.

- Octopus is written using **Visual C++**, making extensive use of the **MFC** class library. We extended the library to enable the use of **tabbed dialogs**, **wizards** and **forms** created from information stored in the database. **SourceSafe** was used for source-code control.
- The application is structured so that all database work is handled by a set of classes, one for each table. This provides high-level interaction between Octopus and the database, and removes any SQL from the main body of the source code. This makes the body of the logic and interface code much cleaner. It also means that alterations to data structures only need to be handled in one place.
- **Type-through** fields were introduced to speed use of the software, avoiding the necessity to use the mouse to select data from lists.
- Octopus grew over the years and contained over 300,000 lines of code spread across 12 DLLs. Several small applets provided database repair and migration utilities, and a wizard helped with system configuration.
- Octopus is **dual-platform** - 16- and 32-bit. This meant that I could not use the new style of **trees** and **lists** that came with Windows 95 and had to develop my own versions that worked in both environments. Other controls I developed for Octopus include a **diary** (similar to that used in Schedule+) and a **grid**.
- Octopus can be extended through its own API allowing the addition of forms, dialogs and menu items and the restriction of functionality.
- I wrote the documentation set for version 2 of Octopus.

When I left KAI, a complete rewrite of the activity sub-system was in progress, including a total restructuring of that portion of the database, to allow the introduction of a **workflow** module. I was also working on a **web-based** version of the product, running as an IIS extension with a very thin client in the browser.

HELPDESK
1995-1996

Helpdesk was an extension to Octopus - bolted on through the API - written in Visual Basic to extend Octopus's activity management functionality into a full helpdesk. The system allowed the logging of calls and then the threading of follow-ups until the problem was solved. Job sheets could be raised for site visits. A further part of the system allowed one to record an inventory of a client's equipment. A special function enabled the user to design job sheets themselves; this did not use a report writer or other custom control, but was implemented as part of the main application in Visual Basic.

RETAIL SELLING
1993-1995

The retail selling system was an add-on for the Tetra 2000 accounts system, written for one of our clients who were a wholesaler of greetings cards. The system extended the conventional order processing system to allow cards to be ordered from a "plan-o-gram", a chart representing the layout of the racks in which the cards are sold in stores. Which card was placed in which pocket of the rack could be modified, as could the layout, allowing the wholesaler to specify how the shop should display the stock. Reordering was eased by the use of barcoded stock tickets provided to the shops.

The system also provided extended reporting in addition to that provided by the accounts system including a best-seller list.

The system ran in DOS and was written using Sycero 'C', a 4 GL based on dBase that generated C code which we compiled using Microsoft C7 and linked with RTLink. The data format was C-ISAM, as this was the format used by Tetra 2000.

JOB ESTIMATING
1993-1995

The job estimating system was loosely linked to Tetra 2000 in that it used the same customer database. It allowed the client - a company that printed brochures, company reports, etc - to automate the estimating process for their orders. The actual application was straightforward, the complication came in the arithmetic required to compile the estimate.

When the estimate was accepted, relevant details could be posted to Tetra 2000's job costing system.

Another C-ISAM database, this was also developed using Sycero 'C'.

WORKSHOP

1993-1995

This application extended Tetra 2000 to provide workshop management and job tracking. The module was used by two companies, one that distributed CCTV equipment and another that repaired car stereos. It allowed them to track a repair job from the time the faulty device entered the workshop until the repair was complete or abandoned. The data were linked to the customer records in Tetra 2000's sales ledger, and an invoicing link was added.

Again, this was a C-ISAM database developed in Sycero 'C'.

RECRUITMENT DATABASE

1993-1994

The recruitment database was written for one of our clients, an IT recruitment consultants, who wanted their clients - particularly Oracle - to be able to dial-up and access an on-line database of contractors available for work. The database could be searched by skill-set and CV's were available on-line. This was before the popularity of the Web and so was written as a DOS program with pcAnywhere to provide the communications. The software was written using Oracle SQL*Forms 3 against an Oracle 7 database.