MISCOMP - An Information System Tool-Kit

Vicky White [1]

Fermilab, P.O.Box 500, Batavie, IL 60510, USA

MISCOMP, a business data information system, has been built with an object oriented design using a client-server framework. It uses both relational and object databases for persistent data storage. It has pioneered the use of an electronic data management and workflow system and has implemented part of the system in C++.

Key words: Databases;Object Server; C++; Java

1 Introduction

MISCOMP is an ambitious project, started in 1992, in order to re-engineer[1] the way the Computing Division carried out its operations, financial tracking, and planning. The goal was to provide a consistent 'enterprise-wise' object model of all the data used in everyday activities and to build a system which would support both the everyday functions and a management information system. An extensive requirements phase was carried out in 1992/93 and an overall architecture and vision for the future was determined. Formal modelling tools and methodologies were employed for this stage of the process to produce a complete breakdown of all aspects of the Computing Division's work by Functional View, Abstract business view, Structural view and Behavioral view - including details of all roles and organizations and all interfaces with lab-wide systems and processes. This analysis and the architecture of the system, defined in 1994, has served as a blueprint for the incremental implementation of the system from the early implementations in Oracle, through to the current use of an object data management system and the extensive use of the web for client applications interacting with the data repository.

[1] On behalf of the MISCOMP project - see Acknowledgements

2 Object Model and Overall Architecture

"The importance of object oriented programming is comparable to that of Whitney's interchangeable part invention, and for many of the same reasons"[2]. It was with this belief that the project proceeded with an overall business object model, although intending to use a relational database as the Database for much of the system. Recent additions make use of an object data management system in addition to a relational database.

Object Model :- Oracle's CASE tool was used initially to record the object model, although not ideal for the task. The Object Management Workbench,[3] a CASE tool implementing the Martin Odell [4] methodology was also used. Five highly abstract fundamental base classes were used in modelling the system - Actor, Provision, Claim, Request and Money. From the start it was understood that the system would be built incrementally and would have use legacy relational databases and data not under the control of the project. Business Data classes were therefore not assumed to conform to an OMG [5] model, but rather to each have a structural definition of itself and its "Aspects" (attributes, relationships, derived attributes and attributes (views) of related objects) as instances of classes which themselves form part of the persistent data repository.

Architecture Framework and Layers :- The overall framework of the system provides for a layered client-server architecture, whereby client applications interact with an object server through MISCOMP object services, rather than directly with a Database engine. Modification and creation of data in the repository is handled in this way, as well as many queries and simple reports on data. Complex reports, are not handled through the object model, but through database specific report-writing tools.

Object Persistency, Object Streaming and Commands :- Business Objects are managed, stored persistently and streamed out to client applications through a class specific Agent, which acts as both a proxy and a protocol adapter for a Business Object and through which requests for attribute data and actions are handled.

3 Implementation of Component parts

As the system has evolved over the past three+ years various implementations of parts of the architecture have been put in place or tried out. One can find many offerings in the marketplace for 'enterprise-wide' solutions, usually involving single vendor buy-in and platform restrictions. The decision whether to buy or build parts of the system has been a difficult one at each stage. Many emerging standards clearly also offer solutions for components of the architecture. We have chosen at this time not to use CORBA, for example, because the need for truly distributed objects (as opposed to a client-side cache merely for performance reasons) was not identified and because per-seat licencing costs for client applications need to be zero. For Server implementations we have however chosen and used some commercial applications and databases which provide the robustness,integrity, and management tools necessary for business data.

3.1 Database Repository Engines

- Oracle [7] Relational database with stored procedures and functions and SQL language interface
- Matrix System ^{TM} Object Database and Workflow from Adra Systems [6] with Matrix Object Query language (MQL) interface
- Aperture CAD drawing and Facilities management data management system [8] with (very recently and not yet in use at Fermilab) SmartPictures web interface and interface to ODBC compliant database
- Remedy Action Request System Helpdesk product [9]

3.2 Object Cache/Object Server/MISCOMP Object Services

- Server Application written in C++. Provides an in-memory object cache, command processing, martialling and streaming of objects and commands, reference counting of objects, persistence management, and notification to clients of modifications made through the in-memory cache.
- Perl non-cached-object server interacting with both Oracle RDB and the Matrix ODB and communicating to Web clients via CGI.
- Matrix Business Object Server [6] (in late beta test) providing interchange of database objects and Java client objects
- PL/SQL minimal object server built in a stored procedure layer of the Oracle data repository. This was an early implementation which nonetheless followed the architectural model and forced interaction with the database through "class" methods, rather than direct row/column manipulation of tables. It is used in major parts of the division's Equipment Database.

3.3 Client Application/Agent (Object Services adapter)/Local cache

- Web Browser application → Web Server acting as Agent for web client [→ CGI layer] → Object Server. Client local cache only in the case of Java
- Event driven C++ GUI applications, with asynchronous behavior and call-back, built using GUI builder tool (from XVT Software Inc.)[10] and MISCOMP-aware subclasses of the XVT GUI class library. Uses a Factory to instantiate these enhanced document, model, window, view and widget classes. An application consists of a much generated code with a custom Application and Document sub-class. Provides model-view-controller functionality between multiple views of the same business object data. Local

read-only Mirror Cache of objects
- Oracle Forms clients $\rightarrow$ PL/SQL object server database layer. Early implementations of GUI clients and some early non-GUI clients

Any interpreted (or other) language program which can easily bind object data and commands to GUI widgets and can handle parsing of ASCII streamed objects can be used to construct powerful client applications. Currently there are no working implementations of such applications, for example based on tcl,perl or python and tk, but some are under consideration.

3.4 Protocols/Streams/Client-Server Communication

Web clients communicate to their Agent/Object Server through http protocol. Servers and other client applications communicate via object streams, based on underlying TCP/IP socket communication. Tools.h++ and net.h++ from RogueWave [11] provide a major part of this functionality. The Matrix business object server[6] is a CORBA-compliant object request broker providing Java client object communication to the objects in the database, based on Orbix^{TM} from Iona [12].

4 The C++ Object Server and MISCOMP Object services

This is implemented as a set of C++ classes usable for both client and server code. Classes communicate and are managed through virtual methods of their base classes, although specific subclasses are instantiated, by an environment specific Object Factory,constructed based on the classes of business objects involved, their type of Persistence management (Cache), their DataSource (e.g. Oracle, Matrix), and the environment - primary server cache or mirror client cache. Rogue Wave [11] Collectables are used at all levels to manage collections of objects and RWHashDictionaries provide for multi-key access to business objects via their Proxy/Agent. Each instance of a managed business object has one or more class-specific Agent instances managing it, one for each client/client connection pair using it. Command objects, themselves collections of collections, pass between the various objects, initiating services as they pass, and between client and server Execution Environments. A number of design patterns were incorporated into the object services design including Abstract Factory, Bridge, Builder, Command, Facade, Iterator, Proxy,and Singleton patterns [13]. Adding new classes of data to the data repository is relatively straightforward involving specification of the classes, using the Persistence Software [14] tools, specification of the limits, validations and constraints on the attributes and relationships of the classes, coding of specific business logic (if any) in C++, and generation of code in support of server, and client where required. Ideally this process should be streamlined further and should proceed from one IDL, or extended IDL-like class definition, but this goal has some way to go to be realized, as indeed do potential CASE tools in support of such ideals. Although primarily intended to provide object services for objects stored persistently in a relational database the C++ object server can provide access, albeit less efficiently, to objects whose persistent data representation is in some custom format. For example, a complete set of objects could be streamed into a Cache, where the objects are managed and shared through the same Persistence generated accessors and mechanisms, but as non-persistent objects. This provides for an in-memory database of objects. The final state of the Cache of objects may then be written out to permanent storage, after some time interval, or at the end of some processing activity. The benefits of commercial database systems, with their transaction processing, rollback, history records and direct multi-user access to the actual persistent data, are clearly not present in such an approach and large collections of objects which could not be held entirely in memory would likewise not be candidates.

5 Conclusions

- Implementing Business information systems is difficult - caught between legacy systems, conservative approaches, disinterested or even downright uncooperative users, and rapidly changing technology[15]
- Although the overall framework and data model of the system has proved invaluable some of the other formal modelling and analysis which was carried out were of less certain lasting value and are hard to relate to over time as the people, management and roles change. Had sufficient resources been put on the project to finish it within 2 years (60 man years were estimated at the end of the analysis phase) the detailed analysis might have proved more relevant
- Object concepts and C++ development are difficult. People say you should plan to throw away one implementation, maybe two. We did. The shortened development time due to re-use of code has yet to be realized in this part of the project, but still the promise of it remains and may be almost in sight. Many pitfalls of designing C++ systems have been encountered, although many avoided [16]
- There are some powerful and effective commercial tools without heavy licence costs. The C++ code generated by both the Persistence Software tools and the XVT GUI builder has been invaluable. RogueWave tools have proved robust and functional, although probably much of the functionality can now be realized through the standard template library. Aperture, although originally a small MAC based drawing system appears to have evolved in the direction of the web and database access
- Code generation tools, including our own, are the way to go.

– The Object Data Management system from Matrix will allow for the complete integration of data and documents into the MISCOMP data repository and looks extremely promising for the workflow, sign-off and control of access to all types of data according to the state of the object and the roles and group affiliations of the user. A web-based Purchase Request processing system is about to be put into production use, based on this system and if successful will lead rapidly to electronic signature and processing of all types of requests in the Computing Division

Acknowledgement

Many people have contributed to the work of the MISCOMP project, including many who were part of the analysis and requirements stage, or early implementation, and have subsequently moved on or left the lab. The following people have made significant contributions and are currently engaged in implementation and support. Matt Vranicar - Project Manager (from Pioneer Systems), Barbara Angelos, Steve White, Cindy Wike (from UPP Systems), Shirley Jones, Norman Ho,Nelly Stanfield and David Ritchie.

References

[1] M. Hammer, J. Champy, Reengineering the Corporation, HarperBusiness, 1993

[2] B.Cox, Object Oriented Programming, Addison-Wesley, 1986

[3] http://www.intellicorp.com/

[4] J.Martin,J.J.Odell, Object-Oriented Analysis and Design, Prentice Hall,1992

[5] The Common Object Request Broker: Architecture and Specification, OMG Document Number 91.12.1, John Wiley & Sons, Inc.,1991

[6] http://www.adra.com/

[7] http://www.oracle.com/

[8] http://www.aperture.com/

[9] http://www.remedy.com/

[10] http://www.xvt.com/

[11] http://www.roguewave.com/

[12] http://www.iona.com/

[13] E.Gamma et.al., Design Patterns,Addison-Wesley, 1995

[14] http://www.persistence.com/

[15] T. Gale, J. Eldred, Getting Results with the Object-Oriented Enterprise Model, SIGS Books, 1996

[16] J. Lakos, Large-Scale C++ Software Design, Addison-Wesley, 1996