ICL TECHNICAL JOURNAL

Volume 8 Issue 3 May 1993

Published by
International Computers Limited
at
Oxford University Press

ICL

ICL

TECHNICAL JOURNAL

The ICL Technical Journal is published twice a year by International Computers Limited at Oxford University Press.

All correspondence and papers to be considered for publication should be addressed to the Editor.

The views expressed in the papers are those of the authors and do not necessarily represent ICL policy.

1993 subscription rates: annual subscription £60 UK and Europe and $120 rest of world; single issues £36 UK and Europe and $72 rest of world. Orders with remittances should be sent to the Journals Subscriptions Department, Oxford University Press, Walton Street, Oxford OX2 6DP.

Printed by The Charlesworth Group, Huddersfield, UK, 0484 517077
ISSN 0142-1557

ICL TECHNICAL JOURNAL
Volume 8 Issue 3

Contents

Editorial iii
French Translations of Abstracts v
German Translations of Abstracts x

OPEN*framework*

Foreword 349

An Introduction to OPEN*framework*
R.F. Brunt 351

The Evolution of the OPEN*framework* Systems Architecture
M.H. Kay 365

Creating Potential-for-Change
G.D. Pratten and P. Henderson 383

OPEN*framework* in Action at DEVETIR
I. Craig 398

Strategic Information Systems Planning: A Process to Integrate IT and Business Strategies
R. Thurlby 416

Describing Systems in the OPEN*framework* Integration Knowledge Base
S. O'Connor 438

Multimedia and Standards for Open Information
I.R. Campbell-Grant and C.R. Smethurst 453

VME-X: Making VME Open
P. Coates 473

Other Papers

A New Approach to Cryptographic Facility Design
J. Press 492

CHISLE: An Engineer's Tool for Hardware System Design
A. Jebson, C. Jones and H. Vosper 506

Distributed Detection of Deadlock
S. Hilditch and T. Thomson **520**

Book Reviews **546**

The ICL Computer Users Association **553**

Guidance to Authors **555**

Editorial Note

ICL's OPEN*framework* is the main topic of this issue of the Journal. Eight papers of very differing scope give a picture of an approach that ICL believes will put users in a better position to meet their particular needs by interlinking hardware and software components chosen from the most suitable sources whether proprietary to ICL or not. As stated in the Foreword by Mr Andrew Boswell, Technical Director of ICL, the approach embodies not merely a universal architecture – conceived of as independent of any vendor – but is based on a philosophy of how to go about this process of integration, starting from an analysis of specific business needs. OPEN*framework* also offers a substantial and systematically organised body of procedural advice and factual information.

Obviously such a wide subject cannot be covered in full detail in one issue of the ICL Technical Journal but ICL has sponsored a complete series of books, now published by Prentice Hall, about different aspects of OPEN*framework*. The introductory book in the series is reviewed in this issue by Annette Haworth. The full list of titles is given in the opening paper on OPEN*framework* by Brunt, in which he analyses the market trends which inspired the philosophy and architectural thinking of OPEN*framework*. The second paper by Kay is an account of how active and long-standing participation by ICL in international discussions within the industry and with users of standards for Open Systems, led (by an evolutionary process which is still going on) to the architectural concepts of OPEN*framework*.

Pratten and Henderson in their paper "Creating Potential for Change", show how to engineer systems to allow for change in the future, taking account of the increasingly complex "value chains" found today in the IT industry. Two papers follow that exemplify ways in which the OPEN*framework* approach may be applied in practice; Craig's paper is especially interesting in that it shows how common objectives can be established in a government authority (in this case in Australia) responsible for a range of activities that initially looked quite distinct. Thurlby describes a detailed analytical study of the business requirements of electricity distribution companies in the UK, though his conclusions are more general.

As already mentioned, OPEN*framework* expects substantial information will be required about the characteristics of products that may be chosen as components of an integrated system. Obviously the structure of this informa-

---

OPEN*framework* is a trademark of International Computers Ltd.

tion will need to be carefully designed to make it both easy to refer to and unambiguous in use. O'Connor proposes a suitable abstract structure for this purpose. Campbell-Grant and Smethurst review multi-media technologies and indicate how information represented in a multiplicity of media can be specified in standard ways, with particular reference to OPEN*framework*. Finally Coates describes VME-X, an extension to the established VME operating system supported by ICL's main frames to allow them to run UNIX-based software and so to work effectively in multi-user environments.

Three further papers discuss a novel approach to the design of cryptographic systems, an engineer's software tool to aid the design of hardware systems and two algorithms for the distributed detection of deadlock in distributed systems.

This issue concludes with reviews of three books, one being the Introduction to OPEN*framework* already mentioned and the other two about New Technology in Police Work and X/OPEN and Open Systems.

The Editor wishes to record his indebtedness and grateful thanks to Dr Michael Kay who, besides writing one himself, elicited and edited the other contributions on OPEN*framework*, to Colin Stretch for comments on those contributions, and to all authors for their papers and great readiness to respond to many editorial comments.

Résumés

R. F. Brunt
Manager, OPEN*framework* Systems Architecture, ICL, West Gorton, Manchester, Royaume-Uni
Introduction à OPENframework

OPEN*framework* répond aux besoins de simplification, de structure et de méthode des acquéreurs de technologie de l'information. Ces besoins découlent de trois facteurs interdépendants, à savoir l'état de maturité du secteur informatique, le déplacement de la fonction d'intégration des systèmes chez l'acheteur, et l'utilisation de l'informatique à l'avantage de l'entreprise. Cet article explique comment ces facteurs sont pris en compte par les divers aspects d'OPEN*framework*; il présente; ses perspectives, qualités, éléments et spécialisations; l'architecture d'entreprise, et les techniques d'étude des processus de gestion avec la base de connaissances.

Michael H. Kay
ICL Fellow, Reading, Royaume-Uni
L'évolution d'OPENframework

Cet article décrit la pensée qui a influencé le développement d'OPEN*framework*, l'architecture des systèmes informatiques d'ICL. Il explique l'évolution d'OPEN*framework* en quatre grandes phases, couvrant l'interfonctionnement, la portabilité, l'intégration des systèmes et, enfin, l'alignement stratégique avec la gestion d'entreprise. Cet article n'est pas conçu comme une énumération chronologique d'événements. Il est plutôt destiné à expliquer la signification historique d'OPEN*framework* pour ICL, au moment même où la société, d'un constructeur de gros systèmes propriétaires, devient un intégrateur, en utilisant des systèmes ouverts comme moyen de rendre la technologie informatique mieux adaptée aux besoins de ses entreprises clientes.

Graham D. Pratten
Company Architect, Potential-for-Change, OPEN*framework* Division, ICL
Peter Henderson
Department of Electronics and Computer Science, University of Southampton, ICL Visiting Fellow
Création du potentiel de changement

Le potentiel de changement est la capacité d'un produit à évoluer pour répondre aux demandes de nouveaux marchés et exploiter de nouvelles opportunités techniques. Cet article étudie comment l'industrie informatique confère cette qualité aux produits qu'elle développe et aux systèmes utilisateur dans lesquels ils sont intégrés.

Il concentre son attention sur les chaînes de valeurs et les processus qui sont utilisés dans les produits et systèmes informatiques en cours de développement et en évolution, et envisage comment il est possible de les orienter vers le changement. Il étudie brièvement comment les produits et systèmes proprement dits peuvent adopter des architectures orientées vers le changement, qui leur permettent d'évoluer de manière souple. Il termine par un regard sur les moteurs du changement dans les entreprises utilisatrices qui génèrent le besoin du potentiel de changement.

Ian Craig
Fujitsu Australia Pty. Ltd., Brisbane, Queensland, Australie
OPENframework en action chez DEVETIR

Les systèmes ouverts disposent du potentiel nécessaire pour offrir des avantages tangibles aux entreprises; toutefois, le passage de la théorie à la pratique se révèle une véritable pierre d'achoppement pour bon nombre d'organisations. A la fin de l'année 1991, ICL et DEVETIR, un département important de l'administration du Queensland en Australie, ont mis sur pied un projet unique de coopération sur trois ans pour développer une architecture de systèmes ouverts utilisant OPEN*framework* d'ICL. L'objectif est de lier directement des processus de gestion réétudiés à une architecture hautement répartie, dont la caractéristique essentielle est la capacité à mettre en oeuvre très rapidement des changements au niveau de l'organisation, des processus et de la technologie. Pour atteindre cet objectif, le projet a adopté des techniques de modélisation de processus, une technologie de support des processus et des systèmes de messagerie basés sur des répertoires. Les problèmes humains restent toutefois les facteurs de réussite les plus importants. Le projet bénéficie de l'appui et de la participation régulière de l'équipe de direction de l'entreprise. A cela se combine un investissement toujours plus important dans la formation, tant du personnel de gestion que du personnel informatique, pour les aider à travailler hors de leurs domaines de prédilection traditionnels.

R. Thurlby
Visiting Fellow, Brunel University, Uxbridge, Royaume-Uni
Planification des systemes informatiques strategiques: Processus d'intégration de l'informatique et des stratégies d'entreprise

Les techniques de planification des systèmes informatiques ont toujours été limitées par leur dépendance vis-à-vis d'une stratégie d'entreprise donnée. Des travaux récents de la direction du programme Nineties, décrits dans cet article, ont démontré que l'informatique est désormais un moteur suffisamment puissant pour que la stratégie d'entreprise et informatique soient devenues interdépendantes. De ce fait, de nouvelles techniques de planification des systèmes informatiques supportant la modélisation de stratégies interdépendantes ont été développées.

Cet article décrit une nouvelle méthodologie de planification des systèmes informatiques qui permet l'analyse et la modélisation dynamiques de l'interdépendance en tant que processus temporel. Cette méthodologie emploie les techniques de l'alignement stratégique et de la modélisation de processus de valeurs. Ces techniques sont décrites avec la théorie sur laquelle elles reposent. Cet article examine également comment il convient d'étendre les besoins d'analyse au-delà des limites de l'organisation en utilisant le concept d'invasion de processus.

L'article présente une étude de cas, qui décrit l'utilisation de cette méthodologie de planification des systèmes informatiques dans certaine compagnies d'électricité régionales au Royaume-Uni. Cette étude se concentre sur l'application de la méthodologie pour le développement et l'alignement de stratégies d'entreprise et de systèmes informatiques qui répondent aux opportunités offertes par la privatisation du secteur de l'électricité britannique.

L'article conclut en soulignant quelques problèmes soulevés par l'utilisation des techniques, et en expliquant comment ils seront résolus grâce aux recherches menées actuellement par l'auteur.

Stuart O'Connor
ICL OPEN*framework* Division, Manchester, Royaume-Uni
Description de systèmes informatiques à l'aide de "solutions" formelles

Le développement de systèmes informatiques plus complexes, rendu possible grâce aux progrès récents de l'informatique répartie et de l'intégration de systèmes, s'est traduit par une approche plus architecturale de la description des systèmes d'information. Cet article étudie une manière formelle de décrire les systèmes informatiques. L'élément fondamental proposé est une "Solution", qui comprend un ensemble fini de propriétés; un système d'information est constitué d'une combinaison de solutions. Les applications directes de cette théorie englobent: la conception structurelle d'une base de données pour le stockage d'informations d'intégration; le remplacement de méthodologies structurées par des outils d'analyse informatiques; et des listes de contrôle pour la capture d'informations à partir d'essais de vérification.

Ian R. Campbell-Grant
ICL Fellow, ICL, Bracknell, Royaume-Uni
C. R. Smethurst
Company Architect, OPEN*framework* Division, ICL, Kidsgrove, Royaume-Uni
Multimédia et normes pour une information ouverte

Cet article comprend deux grandes parties. La première présente les principales technologies multimédias et décrit quelques-uns des impacts sur la société de la mise en oeuvre d'une capacité multimédia réseau globale.

L'un des aspects technologiques essentiels concerne la définition de normes d'acceptation générale pour la représentation des informations multimédias. La seconde partie de l'article étudie ce point de manière plus détaillée, en le mettant en rapport avec l'architecture multimédia d'OPEN*framework* et en indiquant les domaines essentiels dans lesquels des normes sont nécessaires et en évolution.

Paul Coates
Computer Systems Division, ICL, Manchester, Royaume-Uni
VME-X: ouverture de VME

Conforme à la norme XPG4 version 1 de X/Open, VME-X fonctionne sur les serveurs ICL Series 39, qui supportent plusieurs environnements multiutilisateurs similaires à UNIX, en plus de la charge de travail normale d'une machine de la série 39. Cet

article comprend une présentation fonctionnelle brève de VME-X, suivie d'une description de sa structure et de sa mise en oeuvre. Etant donné qu'il fonctionne comme une application au-dessus d'un système d'exploitation général, ces dernières sont assez différentes d'un portage UNIX normal.

Jim Press
ICL Mid-Range Systems Division, Reading, Berks, Royaume-Uni
Nouvelle approche de la conception de fonctions cryptographiques

Cet article introduit les principaux concepts sur lesquels repose la conception du service CKMS (Cryptographic and Key Management Service), qui fait partie de la fonction CSF (Cryptographic Support Facility) utilisable dans d'autres produits ICL.

Le service CKMS est une nouvelle approche de la fourniture de services cryptographiques basée sur des techniques orientées objet. Il permet aux applications clientes d'être indépendantes des détails des algorithmes cryptographiques fondamentaux, ce qui contribue à leur portabilité. Il garantit également l'utilisation des algorithmes de chiffrage conformément à la politique locale en matière de sécurité et a été conçu pour que leur remplacement soit aisé.

A. Jebson, C. Jones, H. Vosper
Corporate Servers Product Group, ICL Corporate Systems Division, West Gorton, Manchester, Royaume-Uni
CHISLE: Un outil de pointe pour la conception de systèmes

Cet article décrit les méthodes employées pour formaliser l'approche du développement du matériel. Il présente une méthodologie basée sur la décomposition de la conception hiérarchique et sur la modélisation multiniveau mixte en utilisant un langage semi-formel "CHISLE" (Combined Hardware and Interface Specification Language for Engineers). Ce langage définit à la fois les fonctionnalités et les interfaces et est compilé en modèles exécutables. Cette approche permet de résoudre les principaux problèmes auxquels sont confrontés les concepteurs de matériel, en l'occurrence, comment spécifier la conception d'une manière concise, compréhensible et sans ambiguïté, et comment représenter le parallélisme et les relations temporelles inhérentes aux éléments complexes du matériel. Les résultats et réalisations à ce jour sont également décrits, avant que l'article ne se termine par la présentation de quelques possibilités d'améliorations futures.

Steve Hilditch
ICL Bracknell, Royaume-Uni
Tom Thomson
ICL Manchester, Royaume-Uni
Détection répartie d'un interblocage

La cohérence des données dans les systèmes de gestion de bases de données réparties est bien souvent mise en oeuvre par des transactions qui préservent séparément la cohérence. L'exécution de transactions concurrentes peut être assurée en plaçant des verrous sur les données et en utilisant une discipline de verrouillage à deux phases. Les transactions peuvent être contraintes à attendre que d'autres suppriment des

verrous. Il peut arriver qu'un cycle complet de transactions en attente provoque un interblocage. Dans un environnement à mémoire répartie, certains interblocages peuvent être détectés localement, c'est-à-dire sans utilisation du réseau de communication. Toutefois, le cas le plus difficile à détecter intervient lorsqu'un certain nombre de transactions sur différents noeuds du réseau s'attendent mutuellement, aucune ne pouvant alors continuer. Il s'agit dans ce cas d'un interblocage réparti.

Cet article présente deux algorithmes simples pour la détection d'un tel interblocage sur un système de base de données réparties sans partage. Ils sont conçus pour être évolutifs et minimisent le trafic de messages sur le réseau. Une approche orientée objet est utilisée et des preuves semi-formelles de la validité des algorithmes sont fournies.

Zusammenfassungen

R. F. Brunt
Manager, OPEN*framework* Systemarchitektur, ICL, West Gorton, Manchester, Großbritannien
*Einführung in das OPEN*framework

OPEN*framework* setzt sich mit den Bedürfnissen von IT-Anwendern nach Vereinfachung, Struktur und Methode auseinander. Diese Bedürfnisse entstehen aufgrund dreier zusammenhängender Faktoren: dem Entwicklungsstand der Informationstechnologie, einer Verschiebung der Systemintegrationsfunktion auf den Anwender und dem Einsatz der Informationstechnologie zur Erreichung von Geschäftsvorteilen. Der Artikel erläutert, wie die verschiedenen Aspekte von OPEN*framework* diese Faktoren berücksichtigen; dabei geht er auf Perspektiven, Eigenschaften, Elemente und Spezialisierungen, auf Unternehmensarchitektur, und technische Verfahren der Geschäftsabwicklung sowie auf die entsprechende Wissensbasis ein.

Michael H. Kay
ICL Fellow, Reading Großbritannien
*Die Entstehung von OPEN*framework

Dieser Artikel befaßt sich mit den Überlegungen, welche die Entwicklung von OPEN*framework*, ICLs Informationssystemarchitektur, beeinflußt haben. Er stellt die Entwicklung von OPEN*framework* in vier Stufen dar, zu denen Dialogfähigkeit, Übertragbarkeit, Systemintegration und schließlich strategische Ausrichtung auf die Unternehmensführung gehören. Der Artikel ist nicht als chronologische Darstellung der Entwicklungsstufen gedacht, sondern soll vielmehr erläutern, welche wichtige historische Rolle OPEN*framework* bei der Transformation von ICL vom Hersteller proprietärer Großrechner zum Systemintegrator gespielt hat. ICL setzt dabei offene Systeme als Mittel ein, um die Informationstechnologie mehr auf die Bedürfnisse der Unternehmen seiner Kunden zuzuschneiden.

Graham D. Pratten
Unternehmensarchitekt, Potential-for-Change, OPEN*framework* Division, ICL

Peter Henderson
Fachbereich Elektronik und Informatik der Universität Southampton, Visiting Fellow von ICL
Veränderungspotential schaffen

Veränderungspotential besteht bei der Entwicklungsfähigkeit eines Produktes, neue Marktbedürfnisse zu befriedigen und neue technische Möglichkeiten auszuschöpfen.

Dieser Artikel beleuchtet, wie die IT-Industrie dies durch die von ihnen entwickelten IT-Produkte sowie durch den Einsatz dieser Produkte in Endbenutzersystemen erreicht.

Er konzentriert sich auf die Werteketten und -prozesse, die bei der Entwicklung und Evolution von IT-Produkten und -Systemen eine Rolle spielen, und zieht in Erwägung, wie diese anpassungsfähig gemacht werden können. Er geht auch kurz darauf ein, wie man diesen Produkten und Systemen selbst eine flexiblere Architektur geben kann, so daß sie veränderungsfähiger werden. Abschließend wirft er einen Blick auf die Veränderungskräfte in Anwenderunternehmen, die das Bedürfnis nach Veränderungspotential hervorrufen.

Ian Craig
Fujitsu Australia Pty. Ltd., Brisbane, Queensland, Australien
*OPEN*framework *in Aktion bei DEVETIR*

Offene Systeme haben das Potential konkrete Geschäftsvorteile zu liefern. Für viele Organisationen ist das Umsetzen der Theorie in die Praxis aber problematisch. Ende 1991 schlossen sich ICL und DEVETIR, ein großes Ministerium in Queensland, Australien, zu einem dreijährigen Partnerschaftsprojekt zur Entwicklung einer auf offenen Systeme basierten Architektur zusammen, das ICLs OPEN*framework* benutzte. Ziel ist der direkte Anschluß individuell konstruierter Geschäftsprozesse an eine verteilte Architektur, deren Hauptmerkmal die Fähigkeit ist, organisatorische, prozessuale und technologische Änderungen sehr schnell durchzuführen. Um dieses Ziel zu erreichen, setzt das Projekt Prozeßmodellierungs- und Prozeßunterstützungstechnologien, sowie auf Dateiverzeichnis basierte Nachrichtensysteme ein. Die bedeutendsten und entscheidendsten Erfolgskriterien sind jedoch die menschlichen Aspekte. Das Projekt arbeitet mit starker Unterstützung und regelmäßiger Beteiligung des Managementteams des Unternehmens, verbunden mit ständig zunehmenden Investitionen in die Fortbildung von Mitarbeitern aller Geschäftsbereiche, um diesen zu ermöglichen, sich über die Grenzen ihrer herkömmlichen Arbeitsgebiete zu orientieren.

R. Thurlby
Visiting Fellow, Brunel University, Uxbridge, Großbritannien
Strategische Informationssystemplanung: Ein Prozeß der Integration von Informationstechnologie und Unternehmensstrategien

Die Verfahren der Informationssystemplanung (I.S. = Information Systems) sind historisch aufgrund ihrer Abhängigkeit von einer gegebenen Unternehmensstrategie eingeschränkt gewesen. Jüngste, in diesem Artikel vergestellte Arbeiten des Programms 'Management in den 90er Jahren' haben gezeigt, daß IT heute zu einem ausreichend wichtigen Aspekt geworden ist, um die Unternehmensstrategie zu beeinflussen. Folglich müssen neue I.S.-Planungssysteme entwickelt werden, die die Modellbildung sich gegenseitig beeinflussender Strategien unterstützen.

Dieser Artikel beschreibt eine neue Methodik der I.S.-Planung, die es ermöglicht, die Beeinflußung zu analysieren und dynamisch als temporalen Prozeß zu modellieren. Die Methodik bedient sich der Verfahren der strategischen Ausrichtung und Werteprozeß-Modellierung, und diese werden zusammen mit der ihnen zugrundelie-

genden Theorie beschrieben. Zusätzlich wird untersucht, wie die Analyse durch Prozeßinvasion über die Grenzen der Organisation hinaus erweitert werden kann.

Der Artikel zeigt die Anwendung dieser I.S.-Planungsmethodik anhand der Fallstudie eines regionalen Elektrizitätsunternehmens in Großbritannien. Die Fallstudie befaßt sich mit der Anwendung dieser Methodik auf die Entwicklung und Ausrichtung von Unternehmens- und I.S.-Strategien in Antwort auf die durch die Privatisierung der britischen Elektrizitätsindustrie eröffneten Möglichkeiten.

Abschließend erörtert der Artikel einige der Fragen, die sich durch die Anwendung der Verfahren ergeben haben, und zeigt, auf welche Weise der Autor diese Fragen in seiner gegenwärtigen Forschung in Angriff nimmt.

Stuart O'Connor
ICL OPEN*framework* Divsion, Manchester, Großbritannien
Erläuterung von Informationssystemen mit Hilfe formaler 'Lösungs'-Beschreibungen

Die durch neuste Fortschritte in verteiltem Computing und bei integrierten Systemen vorangetriebene Entwicklung komplexerer Informationssysteme hat zu einem stärker architekturbezogenen Ansatz bei der Beschreibung von Systemen geführt. Dieser Artikel befaßt sich mit der Frage, wie Informationssysteme formal beschrieben werden können. Die vorgeschlagene fundamentale Komponente ist eine sogenannte *'Lösung'*, die eine bestimmte Anzahl von Eigenschaften besitzt; ein Informationssystem setzt sich dabei aus einer Kombination von Lösungen zusammen. Zu den direkten Anwendungsbereichen dieser Theorie gehören u.a. das strukturelle Design einer Datenbank für Integrationsinformationen, das Ersetzen struktureller Methoden durch Computer-basierende Analyse-Tools und Prüflisten zum Festhalten der aus Verifizierungsversuchen gewonnenen Informationen.

Ian R. Campbell-Grant
ICL Fellow, ICL, Bracknell, Großbritannien

C. R. Smethurst
Unternehmensarchitekt, OPEN*framework* Division, ICL, Kidsgrove, Großbritannien
Multimedia und Standards für Offene Informationen

Dieser Artikel ist in zwei Hauptabschnitte unterteilt: Der erste stellt die Schlüssel-Multimediatechnologien vor und umreißt einige der Auswirkungen, die sich aus der Verwirklichung global vernetzter Multimediakapazitäten ergeben.

Eine Schlüsseltechnologie befaßt sich mit der Etablierung allgemein anerkannter Standards bei der Darstellung von Multimediainformation. Im zweiten Teil des Artikels wird dieser Aspekt näher beleuchtet, indem er zur OPEN*framework*-Multimedianarchitektur in Bezug gestellt wird. Außerdem wird auf die Schlüsselbereiche hingewiesen, in denen Standards benötigt werden und entstehen.

Paul Coates
Computer Systems Division, ICL, Manchester, Großbritannien
VME-X: Die Öffnung von VME

VME-X ist ein auf Servern der ICL Serie 39 laufendes S/W-Produkt, welches mehrere UNIX-gleiche Umgebungen für Multibenutzer neben der normalen Arbeitsbelastung der Serie 39 unterstützt. Es entspricht dem X/Open Standard der XPG4 Basisversion 1. Der Artikel beinhaltet außerdem eine kurze funktionale Beschreibung von VME-X, gefolgt von einer Beschreibung seiner Struktur und Implementierung, die sich – da VME-X als zusätzliche Anwendung zu einem allgemeinen Betriebssystem läuft – von einer normalen UNIX-Partierung unterscheidet.

Jim Press
ICL Mid-Range Systems Division, Reading, Berkshire, Großbritannien
Neuer Ansatz beim Design kryptographischer Einrichtungen

Dieser Artikel beschreibt die wichtigsten Grundkonzepte für das Design von Kryptographik- und Schlüsselwort-Managementservices (CKMS = Cryptographic and Key Management Service), einem Teil der Kryptographik-Unterstützungseinrichtung (CSF = Cryptographic Support Facility), die mit anderen ICL-Produkten verwendet werden kann.

CKMS ist ein neuer Ansatz bei der Bereitstellung von Kryptographik-Services, der auf objectorientierten Verfahren beruht. Er ermöglicht, daß Client-Anwendungen von den zugrundeliegenden kryptographischen Algorithmen unabhängig sind, und unterstützt somit deren Übertragbarkeit. Außerdem sorgt er dafür, daß kryptographische Algorithmen benutzt werden, die der örtlichen Sicherheitspolitik entsprechen, und ist mit Blick auf eine leichte Austauschbarkeit konzipiert worden.

A. Jebson, C. Jones, H. Vosper
Corporate Servers Product Group, ICL Corporate Systems Division, West Gorton, Manchester, Großbritannien
Chisle: Einen Schritt voraus im Systemdesign

Dieser Artikel stellt die Methoden zur Formalisierung des Ansatzes bei der Hardware-Entwicklung dar. Er beschreibt eine auf hierarchischer Design-Gliederung und gemischter Multilevel-Modellbildung basierende Methode, unter Verwendung einer halbformalen Sprache namens 'CHISLE' (Combined Hardware and Interface Specification Language for Engineers = Kombinierte Hardware und Interface Spezifikationssprache für Techniker). Diese Sprache definiert sowohl Funktionalität als auch Schnittstellen und ist in ausführbare Modelle übersetzt. Dieser Ansatz ermöglicht die Behandlung der wichtigsten Probleme von Hardware-Designern, d.h. eine knappe, deutliche und verständliche Beschreibung des Designs sowie die Darstellung der Abhängigkeiten und der Zeitverhältnisse, die komplexen Hardwareteilen eigen sind. Beschrieben werden außerdem die bis dato erzielten Leistungen, und der Artikel schließt mit einigen Verbesserungsvorschlägen für die Zukunft ab.

Steve Hilditch
ICL Bracknell, Großbritannien

Tom Thomson
ICL Manchester, Großbritannien
Dezentralisiertes Erkennen von Blockierungen

Datenkonsistenz in dezentralisierten Datenbankverwaltungssystemen wird häufig durch Transaktionen erreicht, die die Konsistenz separat behandeln. Eine simultane Transaktionsausführung kann durch Datensperrung und der Methode einer Zweiphasensperrung abgesichert werden. Transaktionen können gezwungen werden, darauf zu warten bis andere Transaktionen die Sperren wieder aufheben. Eine komplette Warteschleife von Transaktionen kann zur Blockierung führen. Bei einem dezentralisierten Speichersystem können manche Blockierungen lokal, d.h. ohne Einsatz des Kommunikationsnetzes, erkannt werden. Doch der am schwersten zu erkennende Fall tritt dann ein, wenn eine Reihe von Transaktionen in mehr als einem Netzknoten aufeinander warten, so daß keine ausgeführt werden kann. Man spricht dann von einer dezentralisierten Blockierung.

Wir stellen zwei einfache Algorithmen für die Erkennung von dezentralisierten Blockierungen in einem nicht gemeinsam benutzten dezentralisierten Datenbanksystem vor. Sie sind als meßbar konzipiert und minimieren den Datenverkehr des Netzes. Ein objektorientierter Ansatz wird benutzt, und eine semiformale Überprüfung der Richtigkeit der Algorithmen ist gewährleistet.

FOREWORD

The Open Systems movement deliberately set out to remove the proprietary barriers which stop a competitive market in component products developing. As the barriers fall, price naturally becomes a key factor. A classic commodity market results, in which product substitution is easy and which allows mixing and matching of the current best of breeds. As margins correspondingly fall under competitive pressure, suppliers find it increasingly arduous to bundle all manner of "no charge" services into their offerings. Indeed, the concept of single-supplier arrangements — formerly the norm — ceases to have much logic in a mix and match world. Furthermore, many new suppliers in the Open Systems market have never offered anything more than the equipment or software package itself.

Thus a new set of relationships between suppliers and customers is being formed. The Systems Integration function comes to dominate — it is Systems Integration which performs the vital role of converting the Open Systems components into the successful, operational, application solutions. Systems Integration has changed from being a hidden, bundled activity to being the key ingredient, where risk is taken and potentially large expenses incurred.

OPEN*framework* from ICL is a simple, certain method for undertaking Systems Integration in an Open Systems world. It equips its users to undertake these duties in a systematic, low risk way. The key lies in organising the subject in an understandable way, adding relevant information, providing checklists of points to be considered, offering methods for checking the acceptability of potential solutions and, finally, making accessible special integration technologies. OPEN*framework* in its various forms covers all these subjects and more. OPEN*framework* translates the undoubted, but often idealised, advantages of Open Systems into implemented, paying benefits for real businesses.

When OPEN*framework* was first launched in 1991, it was presented and gained acceptance as ICL's "architecture". This placed ICL in a similar position to that of its competitors. What were not revealed at that time were ICL's more far-reaching intentions for OPEN*framework*. OPEN*framework* was destined to become very much *more* than a prescriptive architecture in the manner of most other suppliers' architectures. OPEN*framework* has now been widened to provide for the IT Strategist, the IT Planner, the Application Provider, the Systems Integrator and other roles in real businesses.

The articles which follow illustrate the blossoming of OPEN*framework*. No reader will conclude from reading this issue of the Journal that "Open *framework* is ICL's marketecture". It is better to avoid this pre-conception. The advice to readers is to think of OPEN*framework* rather as an architecture for creating architectures – something living and evolving; not tied to any particular product or technology but adaptable; full of useful knowledge, practical know-how and down-to-earth advice; a powerful agent for the removal of risk and doubt.

This view of the architecture is appropriate in another dimension. Every organisation can employ OPEN*framework* to create its own unique business applications and IT infrastructure – its own example of OPEN*framework* architecture. OPEN*framework* has a valuable role to play in all organisations, ensuring that the benefits of Open Systems are realised without delay.

A.J. Boswell
ICL Technical Director

An Introduction to OPEN*framework*

R. F. Brunt
Manager, OPENframework Systems Architecture, ICL, West Gorton, Manchester

Abstract

OPENframework addresses the needs of information technology buyers for simplification, structure and method. These needs arise from three inter-related factors; the state of maturity of the information technology industry, the shift of the systems integration function to the buyer and the use of information technology for business advantage. The paper explains how these factors are addressed by the various aspects of OPENframework; the perspectives, qualities, elements and specialisations, and the business architecture, business process engineering techniques and the knowledge base.

1 The State of the Industry

For much of the last half century, developments in the information technology industry have been stimulated by governments and business. Governments, particularly as part of their cold war defence policies, have injected huge amounts of capital. Education in the United States, Europe, Japan and Oceania has created a large pool of skill. The drive for growth has caused businesses to seek ever greater efficiency and competitiveness through investment in information technology. The volume of research and the development of technology have been predictably alarming yet until the late 1980s the momentum was channelled by the strategies and capabilities of a small number of large computer suppliers.

The effect was the growth of a large and highly populated industry supported by the high selling price associated with pioneering technology. New norms for productivity in business were set and as long as this continued, the profitability of the information technology industry was maintained. The 1990s present a rather different face where a number of factors have combined to change the industry radically. The reduction in defence spending brought about by the end of the cold war and the global recession of the early 1990s have drastically reduced growth and have contributed to a halving in profitability within the computer industry over the five years from 1987 to 1992. (McKinsey Report, 1992).

On the technology front, the arrival of microprocessors and the consequent creation of a mass market for personal computers have led to a demand for standardisation. For business systems this demand has been intensified by the open systems movement. The creation of standards, *de jure* or *de facto*, is now driven by business decisions and in free market conditions this often results in multiple overlapping standards being established before their value declines. Under such circumstances, the creation of low cost product components sold in high volumes is bound to prove attractive, which in turn accelerates technological change.

The effects are to give free rein to diversity and to reduce the cost of market entry for a potential supplier. As a result there are many more viable suppliers than there were ten years ago. Competition is greater while margins suffer, compounding the underlying pressure on profitability brought about by the changed economic climate.

The industry is fragmenting into a greater number of smaller companies, polarised into technology manufacturers and technology providers. (Moschells, 1992). At the same time, standardisation means that there is no need for large numbers of different implementations of each technology. For most significant components, such as microprocessors, disk drives, operating systems, C compilers, relational databases, or word processors, three quarters of the world market is served by a dozen or so development teams. This eliminates unproductive duplication of activity, because companies like ICL that focus on the supply of complete information systems no longer need to develop each technology in-house. The result is market-led innovation in the application of information technology, as resource in the industry shifts from creation of technology to developing applications software and services more closely tailored to individual customers' needs.

In short, three effects can be clearly seen in the state of the information technology industry in the early 1990s:

- Growth in the information technology industry has declined in line with profitability, and many of the traditional large suppliers are contracting.
- The gradual shift of resources into software and services is encouraging the development of innovative applications of information technology, to match the need of businesses to gain competitive advantage as well as productivity.
- The fragmentation of the industry into a greater number of smaller companies is introducing diversity and complexity for the buyer. This is especially true for the many organisations who in the past have dealt almost exclusively with a single major supplier.

2 The Buyers of Information Technology

In the current state of the information technology industry, many buyers have emerged from the protective umbrella of their chosen supplier. They have done so because they could no longer find a complete solution to their

requirements from one vendor. Most buyers also seek multi-vendor solutions because they believe that a cost advantage will thereby accrue, which is certainly true for the basic equipment and software.

For most buyers this emergence is not a sudden occurrence; they have been moving gradually towards multi-vendor purchasing for some years. Nevertheless all buyers still face problems caused by the diversity and complexity of the modern information technology industry. The choices available are daunting, and in most cases the decisions have to be made by people who have little technical knowledge themselves, but who are deluged with advice from numerous sources inside and outside the organisation, little of it truly objective or well-informed. As a result, fashions such as object-orientation, client/server, and downsizing can take hold for reasons only loosely related to their technical merit.

Having decided to go for the benefits of multi-vendor supply, the buyer is next faced with the problem of systems integration. In the old single supplier (high profit) days, integration was a problem for the supplier; but it no longer comes for free in the new low margin, multi-supplier market. Initially systems consultancy companies responded to this need, but as more and more buyers assert their independence, integration has fast become a major market in which the traditional information technology suppliers also participate.

Buyers are unwilling to carry the full burden of systems integration: they discover that what they have saved on products, they spend on procurement and integration. There is a noticeable trend to limit this problem by buying integrated packages. This trend will not completely solve the integration problem but will raise it to a macroscopic level. In other words, the market is calling for bigger and bigger building blocks and hence for sharing the costs of integration between suppliers and buyers.

Business is now run by managers who cannot remember how the job was ever done without computers. They take current norms for productivity for granted, as a factor to be maintained but with limited scope for further improvement. They are now looking for other ways of using information technology to provide them with competitive advantage. The Management in the Nineties study (Scott Morton, 1991) identifies this trend and analyses its impact.

For many companies this problem is understood but the solution remains elusive: it implies the adoption of newly emerging technologies in areas such as distributed computing, object-orientation, process engineering, and knowledge-based systems; but it also affects the devolution of responsibility within the management of an organisation, and its attitude to innovation and risk.

We can thus draw three conclusions about buyers of information technology:
- Choice and diversity create daunting complications for many buyers.

- Integration of multi-vendor systems has become the buyers' problem. They will seek to purchase progressively larger building blocks to shift some of the cost of integration back to their suppliers.
- There is a drive to use newly emerging technologies to gain competitive advantage, as well as maintaining and improving productivity levels.

3 OPEN*framework*

3.1 What is OPENframework?

The definition in "Systems Architecture: an introduction" (Brunt and Hutt, 1992) says: "OPEN*framework* is an architecture which enterprises can use to create information systems that precisely meet their business requirements."

But it goes on to say that OPEN*framework* does not, like other architectures, pretend to define a single solution for everyone. Customers for information technology are all different, and their needs are constantly changing, so there cannot be a single answer that is right for all of them at all times. OPEN*framework* therefore provides a skeletal structure which is of completely general applicability, together with a process for specialising the structure to the needs of individual enterprises, to create specific architectures tailored to local needs.

This process starts with the definition of business architecture, which is to say a model of the business conducted by the enterprise, containing information such as the organisation structure, the strategic plan, asset models, core competences, competitive analysis, value chain analysis, and models of business processes. All these factors influence the information systems architecture.

The goal of the information systems architecture is to ensure that information systems are built that meet the needs of the enterprise in five key areas: availability, usability, performance, security, and perhaps above all, potential for change. These five qualities, together with cost of ownership, indicate whether or not an information system is delivering value for money. Deficiencies in an information system can always be attributed to a shortfall in one or more of these qualities, and thus either to a failure in the engineering process by which the information system was constructed or to an error in specifying its requirements. The five qualities in OPEN*framework* are illustrated in Figure 1.

If it is true that to get the information system right we must take these five qualities into account, then it is also true that to determine the requirements for these five qualities we must first study the environment in which the information system is to be used, that is, the people and the processes that will make the information system into a thing of value to the enterprise. A

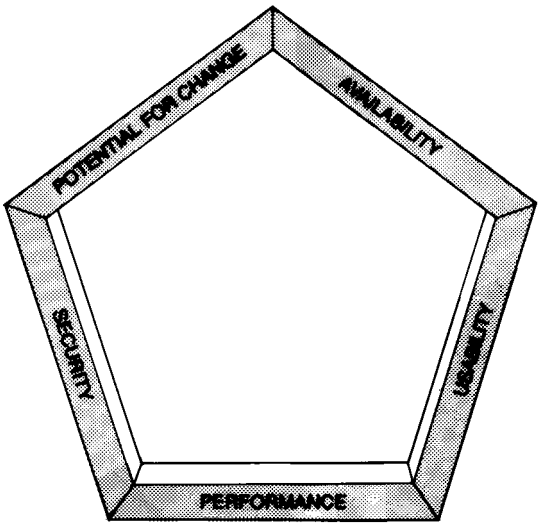


Fig. 1 OPEN*framework* Qualities

mismatch between the technology and the people it serves is the most common cause of failure in large information systems.

OPEN*framework* studies the people and processes in terms of four perspectives: the enterprise management, the users, the application developers, and the service providers. These last two roles perhaps deserve some further explanation, since OPEN*framework* stretches the conventional meanings of the terms:

- Application developers do not merely write the enterprise-specific applications, they are also concerned with every aspect of system design and construction. For example, they select components that will be procured as part of the final system, and they develop the documentation and training materials for users.
- Service providers, similarly, are concerned with every aspect of operation of the information system, whether it be capacity planning, installation of hardware and software, provision of training, management of faults, or running a help desk.

Application developers, clearly, carry the responsibility for designing a system that meets quality requirements, and this responsibility is discharged by using appropriate engineering disciplines at every stage of development. Service providers, equally, are responsible for measuring and monitoring the information system to ensure that agreed service levels are being attained, and taking corrective action when they are not.

The diagram showing the eight OPEN*framework* elements (Figure 2) is perhaps the most widely-recognised feature of the architecture. As the intro-

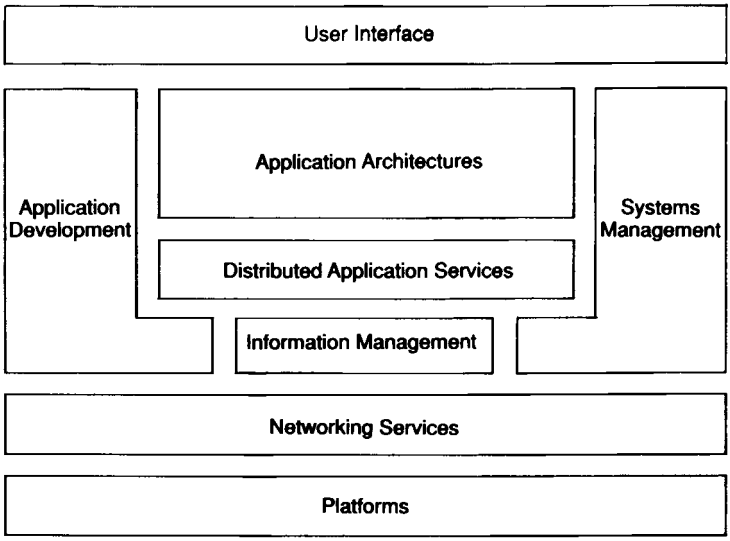


Fig. 2 OPEN*framework* Structure

duction to this paper shows, however, it is by no means its most important feature.

The reason for having eight elements with clean boundaries between them is to achieve potential for change. It allows the technology in each element to develop independently of the others. It also allows human skills to be partitioned: it becomes possible to employ database experts who know little about networking, and networking experts who know little about databases. This principle of modularity has been understood for many years, and is not unique to OPEN*framework*. As with other modular decompositions, the precise choices of which function goes in which element are essentially pragmatic, and in the case of OPEN*framework* are based as much as anything else on the structure of the industry supplying the components: it was observed many years ago that the structure of a system always corresponds to the structure of the organisation that produced it, which in this case is the information technology industry as a whole.

People seeing this OPEN*framework* structure for the first time sometimes react by asking what is different about it. The answer is, nothing: OPEN*framework* addresses the challenge of integrating open systems, and it could not achieve its aims if the technical model differed significantly from what the industry as a whole was doing. What is different about OPENframework is not the structural model, but the way in which it can be used as a management tool to deliver business benefits through the medium of the five qualities.

3.2 The OPENframework process

OPEN*framework* is designed to help organisations build their own architectures as a way of addressing the problems identified in the previous sections of this paper. Because OPEN*framework* starts by addressing the needs of multi-vendor systems integration, it cannot adopt the traditional architectural approach of imposing a uniform set of constraints on all buyers. Instead it recognises that the detailed policies and standards will vary from one organisation to another.

OPEN*framework* addresses the challenges discussed in the previous sections:

- Complexity of the technology
- Shift of the systems integration burden from suppliers to buyers
- New business uses for information technology

Because the requirement is to integrate systems from a multiplicity of vendors, OPEN*framework* has to address these challenges without making any assumptions about the specific products to be chosen. In particular, an architecture for integrating multi-vendor systems cannot demand that the buyer should be locked in to products from ICL or from any other single supplier. It must be a vendor-neutral architecture.

OPEN*framework* therefore addresses these challenges in a vendor-neutral way, by means of three techniques:

- Simplification: this is achieved through abstraction, by distinguishing the attributes of system components that have strategic significance from the mass of detail that is only of peripheral interest.
- Structured presentation: this is achieved by decomposing the complex mass of detail into subject areas that are mutually independent and that can each be analysed in a consistent way.
- Advice and guidance: by offering information, advice, methods and processes that can be used at every stage of system design, construction, evolution, and management to maximise the benefits obtained from the information system.

In the following sections, each of these concepts is examined in terms of its representation in OPENframework and its contribution to addressing the challenges identified above.

3.3 Simplification

We have seen that buyers are faced with a daunting variety of product choice. In making their investment decisions, they therefore have to simplify the problem. One of the principal objectives of OPEN*framework* is to show buyers how this simplification can be achieved.

The problem for OPEN*framework* is firstly to identify who needs what to be simpler, and then to distil a coherent, simple, and technically sound model from the mass of detail. To do justice to the capabilities of the technology and to be appealing to sufficient potential buyers, even this

simple model will be of some sophistication. Presentation of the model is a problem in its own right, not only because of its inherently abstract nature, but also because of the need to convince sceptical buyers that OPEN*framework* is a genuine attempt to help them solve their integration problems and not merely a pretext for selling branded products.

The analysis of the industry given earlier reveals the factors contributing to over-complexity: multiplicity of standards, diversity of products, reducing levels of control within any one organisation, and the accelerating pace of technological change. One source of simplification is through standards. The existence of programming language standards, for example, allows a two-step decision process: first choose a language, then choose a compiler. This process reduces the number of options that have to be evaluated and discarded. It also preserves a degree of freedom to change decisions subsequently.

As the value of standards has increased, so has the scramble to influence them, to intercept them, and to usurp them by market dominance. The fact that this race has been run differently in Europe, the US and Japan adds yet more confusion. The result has been an explosion of standards to the point where they are sometimes seen as part of the problem rather than as part of the solution.

There is no universal answer to this problem: the acceptance of standards will vary with geography and over time. OPEN*framework* recommends a simple policy for the selection of standards:

- Firstly, adoption of standards that are widely implemented in a variety of products is fundamental to sound system design.
- Secondly, provided this condition is met, standards that are defined by a rigorous process open to public scrutiny will generally prove more durable than those controlled by a single supplier.

Following these principles, in each area of OPEN*framework* a small number of standards are identified which correspond to the major areas for decisions in systems architecture. OPEN*framework* also has a liberal approach in the sense that it is not invalidated by any particular option. If unorthodox choices are made, its usefulness will degrade gracefully.

Diversity and choice exist beyond the area of standards, and this opens the way to yet more subjectivity. A liberal architecture like OPEN*framework* cannot *prescribe* the solution but it can demonstrate the best way to *arrive* at the best solution for a given user. The problem space is large but can be structured to allow the decision maker to examine one small area at a time with confidence that decisions taken in this area will not interact unacceptably with those affecting other areas.

OPEN*framework* achieves this by breaking the problem into three dimensions corresponding to the people involved (the perspectives), a value system

for the people to set goals (the qualities), and the technology (the elements), as described in section 3.1 above.

No organisation can afford to make a completely fresh start with its information systems, so all advances are by their nature evolutionary steps from a starting point towards a defined goal. Taking responsibility for systems integration means that organisations no longer have the luxury of implementing a planned and orderly progression of products from a major supplier. Careful planning and monitoring is required if the evolution of the system is to proceed in a controlled way, with separate decisions being mutually consistent, and without tactical changes being continually invalidated by larger strategic choices. Positive feedback in this process is provided by managing the OPEN*framework* qualities.

OPEN*framework* has a definite scope today, but is designed to accommodate future change. This flexibility is fundamental if it is to take account of advances in the industry, particularly in technology. Thus, OPEN*framework* identifies trends in all elements and qualities in order to anticipate likely advances in technology and its usage. A high rate of change can be catered for by ensuring that information systems are designed and constructed with change in mind. This is handled in OPEN*framework* by a specific quality, Potential for Change, discussed in detail in another paper in this issue (Pratten and Henderson, 1993).

3.4 Structured presentation

In presenting OPEN*framework*, as with any architecture, there is always a balance to be achieved between rigour and comprehension. OPEN*framework* is of little value if it cannot be understood. There have been formal approaches to systems architecture, for example ANSA (Herbert, 1987) and the Zackman framework (Sowa and Zackman, 1992) which provide a useful theoretical foundation, but OPEN*framework* aims to be of more direct value to the practitioner than these theoretical approaches.

OPEN*framework* incorporates much pragmatism and compromise but this does not prevent it achieving its objectives. In information technology as in the construction of buildings, architecture is part art and part science. It enables the planner to focus on the most important issues first. OPEN*framework* guides its users to concentrate upon the matters central to each element or quality from their own perspective rather than expending energy at the periphery of the problem. Effective presentation of the architecture thus has two aspects: how should the whole be partitioned, and then what is central to each partition.

The first level of partitioning is to split technology from values and from people. This yields seventeen discrete parts: the eight technology elements, five qualities and four perspectives described in section 3.1. This partitioning relies on a tight policy on terminology and a firm consistency of style. Each

part has limited value in its own right but is complete when related to the others. Examination of the reference architectures for the elements and the qualities, as described in the publications listed at the end of this paper, will reveal consistency of this nature.

Application of OPEN*framework* provides a short cut for an organisation aiming to define an architecture for its information systems. Rather than starting with a blank sheet of paper, the process starts by taking a comprehensive list of possibilities and eliminating those that are not appropriate. This process is referred to as specialisation, and results in the identification, and potential reuse, of a subset of OPEN*framework*. Such subsetting may take place for a number of reasons:

- A specialisation may define collections of technology that are appropriate to particular types of workload, such as transaction processing, multimedia document management, electronic data interchange, or business process enactment.
- A specialisation may define architectural options appropriate to the product sets available from a particular supplier or suppliers, for example, IBM mainframes surrounded by industry-standard personal computer LANs.
- A specialisation may be oriented to the needs of a particular industry, for example retail banking or hospital patient administration.

In this way OPENframework begins to address the need for integrating larger and larger building blocks. In the limit, applying OPENframework becomes a matter of customising a set of specialisations.

3.5 Advice and guidance

Despite its claim to be a way of simplifying problems and imposing structure on their solutions, OPEN*framework* as a reference source is still complex. Its value is realised by treating problems in small pieces and it would be incomplete without advice on how it is to be used, methods and processes for applying it, and comprehensive information to guide the user in making choices.

The Management in the Nineties study (Scott Morton, 1991) developed a Strategic Alignment Process which asserts that an organisation making best use of information technology will maintain a strong relationship between its business strategy, its infrastructure, and its information technology strategy and systems. These relationships are illustrated in Figure 3. Examples of the application of this process are given in two papers (Thurlby, 1993; Craig, 1993) in this issue of the ICL Technical Journal.

OPEN*framework* follows this philosophy by providing the means to achieve such alignment. The business architecture of OPENframework permits an organisation to express its business goals and constraints so as to facilitate the development of an appropriate information system architecture. The enterprise management perspective and the potential for change quality are

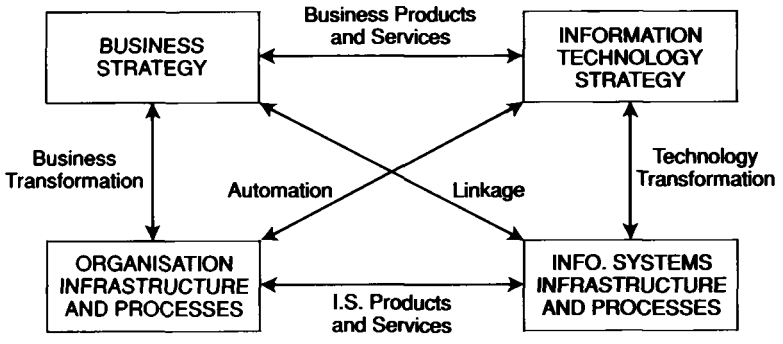


Fig. 3 The Strategic Alignment Process

common to both the business and the information system architectures. The application architectures element bridges the two. The relationships between these three aspects of OPEN*framework* are explored in another paper (Kay, 1993) in this issue.

The process for using OPEN*framework* starts with the business strategy, moves through definition of business processes, then through development of information systems architecture, and finally to procurement and construction of the desired system. This process avoids the discontinuities that can arise when methods from different sources are mixed. It is a macroscopic approach, giving an overall structure to the exercise but without prescribing the use of specific tools or techniques at lower levels.

The final stages of this process are the practical steps of selecting and integrating real products and monitoring the behaviour of real systems. To facilitate these steps, OPEN*framework* provides access to a wide-ranging information base of practical information aimed at advising users exactly what products work with what others and giving sound guidance, based upon practical experience, of how such products can be integrated. This is known as the OPEN*framework* Integration Knowledge Base (IKB), and is described by (O'Connor, 1993).

Much of this multi-vendor information is obtained by running real workloads on model configurations. This can be seen as a another step in increasing the size of the blocks to be integrated.

The process of translating business strategy into working information systems depends on people, and the definition of the process therefore includes a definition of these groups of people and their roles. These roles are described in terms of the four OPEN*framework* perspectives: enterprise management, users, application developers, and service providers.

Each of these groups of people participates in the process of change that causes the information system to evolve, and each has responsibilities to discharge if the system is to evolve in a way that increases its value to the business. In particular, each group makes decisions that contribute to the achievement of each of the five OPEN*framework* qualities.

This description simply formalises what happens naturally in many organisations, but nevertheless an understanding of these roles and their relationships helps to avoid the errors that have caused so many projects to fail to meet their objectives.

4 Summary

This paper is written some two and a half years into the development of OPEN*framework*. Much documentation has been published and a significant base of knowledge has been collected. Most of the ideas in OPEN*framework* have been used successfully somewhere, and a few organisations have adopted the OPEN*framework* approach as the principal plank of their information system planning.

The paper has concentrated upon the macroscopic level, because OPEN*framework* is addressing the challenge of engineering-in-the-large. The problems it sets out to solve are large, and to solve them the first essential is to understand the overall picture. This is not to say that the task of developing the detail of each element and quality is minor, rather the opposite: the details are so significant and so complex that the high-level framework is essential to enable the detail to be controlled and comprehended.

We have shown that buyers of information technology face problems in respect of:

- Complexity and diversity
- Migration of the systems integration responsibility
- New business uses for information technology.

OPEN*framework* responds to these needs by recognising:

- Need for simplification
- Need for a systematic approach and clear, structured presentation
- A sound and consistent method of application, using a source of reliable practical information

OPEN*framework* addresses these through a structure of business architecture, perspectives (people roles), qualities (a value system) and elements (technology) which can be customised by a user organisation to produce its own architecture. Specialised architectures are produced not from scratch, but by specialising what exists already.

All this is part of the process of aligning information systems with business strategy. This process is facilitated by such techniques as business process engineering, and by the provision of information about the integration of products from multiple suppliers that has been proven in practice.

References

BRUNT, R.F. and HUTT, A.T.F. 1992: see book title below.

CRAIG, I. OPEN*framework* in Action at DEVETIR. *ICL Tech. J.* 8(3), pp. 398–415, 1993.

HERBERT, A. The Advanced Network Systems Architecture project. *ICL Tech. J.* 5(4) pp. 638–651, 1987.

KAY, M.H. The Evolution of the OPEN*framework* Systems Architecture, *ICL Tech. J* 8(3), pp. 365–382, 1993.

McKINSEY. *The* 1992 *Report on the Computer Industry*, McKinsey & Co Inc., 1992.

MOSCHELLS, D. The restructuring of the information technology industry. *IDC*, May 1992.

O'CONNOR, S. Describing Systems in the OPEN*framework* Integration Knowledge Base, *ICL Tech. J.* 8(3), pp. 438–452, 1993.

PRATTEN, G.D. and HENDERSON, P. Creating Potential-for-Change, *ICL Tech. J.* 8(3), pp. 383–397, 1993.

SCOTT MORTON, M. *The Corporation of the 1990s.* OUP, ISBN 0–19–506358–9, 1991.

SOWA, J.F. and ZACKMAN, J.A. Extending and formalizing the framework for information systems architecture. *IBM Systems Journal*, Vol. 31 No. 3, 1992.

THURLBY, R. Strategic Information Systems Planning: A Process to Integrate IT and Business Strategies, *ICL Tech. J.* 8(3), pp. 416–437, 1993.

Further reading

The following books describe the OPEN*framework* systems architecture in further detail. All are published by Prentice Hall in 1993, and may be ordered either from ICL or through bookshops.

Overview

BRUNT, R.F. and HUTT, A.T.F. *The Systems Architecture: an introduction.* ISBN 0–13–560186-X.

Qualities

SMETHURST, C.R. *Availability.* ISBN 0–13–630948–8.
HUTT, A.T.F. *Usability.* ISBN 0–13–630930–5.
SUTCLIFFE, S.E. *Performance.* ISBN 0–13–630666–7.
FAIRTHORNE, S.B. *Security.* ISBN 0–13–630658–6.
PRATTEN, G.D. *Potential for Change.* ISBN 0–13–630617–9.

Elements

HUTT, A.T.F. *User Interface.* ISBN 0–13–630591–1.
BRENNER, J.B. *Distributed Application Services.* ISBN 0–13–630518–0.
KAY, M.H. *Information Management.* ISBN 0–13–630500–8.
BROWN, G.H. *Application Development.* ISBN 0–13–630484–2.

GALE, A.C. *Systems Management.* ISBN 0–13–630450–8.
DEIGNAN, F. *Networking Services.* ISBN 0–13–630393–5.
McVITIE, D.G. *Platforms.* ISBN 0–13–630385–4.

Specialisations

BANKS, R. *Transaction Management.* ISBN 0–13–630377–3.

Biography

Ron Brunt

Ron Brunt joined ICT in 1964 with a degree in electrical engineering from King's College, London. He has had a broadly based career in information technology development. For several years he was based at ICT, Stevenage, working on the development of ICT 1900 systems and software. Since 1968, working at Kidsgrove, he has been involved in systems and software design at a strategic level including ICL's VME operating system. In the 1980s he was responsible for strategic product planning of ICL mainframe systems.

He has spent two periods overseas; in Europe with the European Commission and in the US West coast software industry.

In 1991 he became the leader of ICL's systems architecture group with the task of creating OPENframework. The group has now published fourteen books on this subject and is providing consultancy to a growing number of major organisations.

The Evolution of the OPEN*framework* Systems Architecture

Michael H. Kay
ICL Fellow, Reading, UK

Abstract

This paper describes the thinking that influenced the development of the systems architecture of OPEN*framework*, ICL's vision for open systems integration. It explains the evolution of OPEN*framework* in four broad stages, covering interworking, portability, systems integration, and finally strategic alignment with enterprise management. The paper is not intended as a chronological record of events, but is designed rather to explain the historical significance of OPEN*framework* to ICL as it transforms itself from a manufacturer of proprietary mainframes to a systems integrator, using open systems as a means of making information technology more responsive to the needs of its customers' enterprises.

1 Introduction

The evolution of ICL's approach to open systems can be seen as one of gradually widening scope, through four broad stages:

- During the first stage, 1980–1983, the primary objective was to allow interworking between ICL machines and, where possible, with non-ICL machines. This led to the introduction of Information Processing Architecture (IPA), a networking architecture spanning all ICL's machines and designed to intercept OSI standards as they became available.
- During the second stage, 1983–1987, the scope expanded to include portability of software across ICL machines and (again, where possible) non-ICL machines. This stage included the formation of X/Open and its publication of a Common Application Environment.
- During the third stage, 1987–1990, previous work was expanded and consolidated to define an integrated systems architecture, in the sense of a coherent distributed computing infrastructure with consistent facilities in areas such as systems management, application development, user interface, messaging, and security. The culmination of this stage was the announcement of OPEN*framework* in May 1991.

- During the fourth stage, which is still underway, the emphasis is on using the information systems architecture within an enterprise, to achieve more effective alignment of information technology strategy with business strategy. This stage focuses on the engineering techniques needed to cope with rapid changes in technology and with increasing variety of both products and standards; it recognises that all information systems evolve over many years and incorporate hardware and software components from many different suppliers.

These activities were cumulative, in the sense that each stage built on the results of previous work. They were also interdependent and mutually reinforcing: ultimately, none of the objectives could be achieved without all four strands being present.

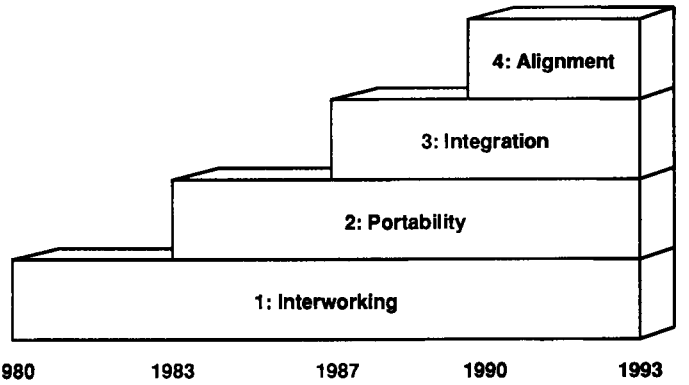


Fig. 1 Four stages of architectural evolution

The progression of these four stages is illustrated in Figure 1.
The following sections analyse the principal objectives, activities, and outcomes of each of these four stages.

2 Stage One – The introduction of IPA

IPA (Information Processing Architecture) has been described in (Kemp and Reynolds, 1980; Brenner, 1983).

The principal motivation for the introduction of IPA was the need to provide a coherent set of interworking services between ICL machines. Interworking with non-ICL equipment was recognised as a requirement, but was less urgent. Work on open standards for interconnection had already started (Houldsworth, 1978; Brenner, 1980; Houldsworth, 1992) and the decision was made early on that IPA should be based on OSI standards as they became available. In the meantime, however, it was necessary to provide some level of interworking capability that recognised the need to coexist with currently installed networks (which at that time usually meant terminals

connected to a main-frame via front-end terminal concentrators) while also providing an evolution path to OSI standards later.

With most of the company's engineering effort in the 1970s engaged on completing the development of the New Range (2900 series) and its VME operating system, development of networking services in ICL was behind the rest of the industry.

According to (Campbell-Kelly 1989):

> *"ICL only began to develop a coherent networking strategy in 1979, but for once its tardiness proved almost an advantage, since it was able to adopt the OSI international standards for networking. IBM and the other manufacturers, having started earlier, were locked into their own proprietary standards. ICL's Information Processing Architecture (IPA) was finally announced in 1980. The adoption of OSI standards was to become a major competitive weapon in the 1980s."*

An important objective of the IPA work was the need for smooth migration. Although ICL might have lacked a coherent networking architecture in the past, that did not mean our customers had no installed networks. The paper by (Kemp and Reynolds 1980) shows how IPA can be seen as an evolution from the earlier Full XBM protocols (also known as ICLC-03). The OSI 7-layer reference model was seen as providing the clue to how this migration might be achieved; in particular layer four, the transport layer, was recognised as a critical interface.

The migration strategy to move customers to OSI has been described in (Houldsworth 1988). The strategy was to encourage customers to install OSI bottom-up, starting with an OSI transport service. Initially this would run the existing interworking applications, so it could be installed with no impact on existing applications or users; subsequently new OSI applications could be added over the same transport service.

By and large this strategy was successful, thanks to a policy, maintained consistently over a long period, of providing OSI as the first choice product and Full XBM and other protocol suites as alternatives only for users who really needed them.

The result was that by the late 1980s, ICL's networking products, and most of our customers' networks, were based solidly around formal international standards.

However, the industry as a whole did not move towards OSI standards as rapidly as some had hoped. The momentum towards OSI was disturbed in the late 1980s by the increasing popularity of portable system software such as UNIX and Novell's Netware, which provided an alternative way of achieving interworking between heterogeneous platforms. These products gave pre-OSI standards such as TCP/IP a new lease of life. Proprietary

standards such as IBM's SNA also refused to go away, despite the fact that many large enterprises were making a strategic commitment to OSI.

This problem was not entirely unforeseen: to quote from (Brenner, 1980):

> *"It is clearly apparent that the idealistic outcome of everybody worldwide converging quickly onto one unified set of OSI standards is highly improbable. ... Predictably, there will be some competing and incompatible standards from different sources, and certainly some lasting incompatibilities."*

IPA's approach to this problem was to define a kernel based on OSI standards, and a number of secondary architectures based on *de facto* or proprietary standards, using the OSI reference model to define the points at which interoperation was possible. This approach stood the test of time, and is largely retained in OPEN*framework*.

3 Stage Two – Common Application Environment

In 1982 many of ICL's customers were still using the various 24-bit regimes (1900, 2903, ME29, TME and CME), but transition to VME had started in earnest. In addition, the first DRS machines based on microprocessor technology had been introduced, and there was also a substantial base of users on the System 25 platform launched as a successor to the Singer System Ten. There was an urgent need to make it as easy as possible to move software between all these platforms with the minimum of change.

At the same time it was recognised that an increasing number of purchasing decisions were application-led, and that ICL would lose sales unless its hardware was capable of running the user's chosen application package. This too implied a need for application portability.

There had been many attempts in ICL to define internal standards for application portability. Cross-range specifications for COBOL, IDMS, and TP were defined around 1980, as was a common set of programming interfaces to the two operating systems VME/B and VME/K. But these had not been very successful: porting applications was still notoriously expensive.

A renewed attempt to address this problem started with a conference held at Cirencester in 1982, and was known as the Cirencester programme. This led to a number of working parties attempting to identify portability standards in areas such as database, programming languages, and transaction processing.

In general it is true to say that where standards have been defined outside ICL, they have been adopted successfully within the company, but ICL has found it very difficult to impose internal standards. Divergence from such

standards can always be justified on the irrefutable grounds that customer requirements take precedence.

For example, ICL's Normalised Document Format (NDF) standard was enhanced in the mid 1980s to allow improved interworking between the OFFICEPOWER office automation system and the ICLFILE document store; but NDF implementations on the ME29 and System 25 machines were never upgraded because there was no prospect of ever recovering the development cost.

Sometimes changes like this are made for good reasons, and with a well-designed standard like NDF they can be made in a way that does not jeopardise interworking. Sometimes they are made for bad reasons, simply because an individual customer is convinced the enhancement is necessary when in fact alternative solutions could be found. Either way, this kind of creeping divergence is less likely to happen with an external standard because the consequences are more visible to management and users alike.

Cirencester attempted to establish a set of internal standards for software portability within ICL, and as such it failed. But the project had an invaluable outcome, which was the management realisation that a common application environment was needed, and that it could not be proprietary to ICL.

A few strategists in ICL had started to become interested in the potential of UNIX as a common application environment as early as 1982, and by 1984 they had convinced the company that this was the way forward. It was recognised that application portability would always be limited if the operating system interfaces were different, and a portable operating system therefore offered the best solution to the problem. It was important though to establish some control over the specification of these interfaces – UNIX in those days was notorious for the informality of its documentation and change control. This led to the formation of the BISON consortium (Bull, ICL, Siemens, Olivetti, and Nixdorf), a group of European vendors who all shared similar views. With subsequent expansion, the BISON group became X/Open.

The formation and subsequent development of X/Open is described in (Taylor, 1987, 1991, 1992). To quote:

> *"During the first half of 1984, ICL approached the other major European computer manufacturers with a view to ensuring that there would not only be a single standard at the UNIX level, but also that a complete Common Application Environment should be defined covering the basic operating system, data management, integration of applications, data communications, distributed systems, higher level languages, internationalisation, and all the many other aspects involved in providing a comprehensive interface for portable applications".*

X/Open succeeded quickly in defining a common application environment. The first round of specifications were completed as early as May 1985. By

the late 1980s ICL had completely redesigned its mid-range product line around X/Open standards, adopting UNIX System V.4 as the underlying operating system. In addition, in 1991 ICL became the first supplier to achieve X/Open conformance for a non-UNIX mainframe system, with the introduction of VME-X (Coates 1993). This finally provided an acceptable solution to the problem of application portability first addressed ten years earlier.

4 Stage Three – Delivering Integrated Systems

As the 1980s progressed it was recognised that the move towards open systems would significantly alter the shape of the information technology industry, with standardisation leading not only to portable interworking applications, but also to widespread availability of commodity components: everything from microprocessors and operating systems to compilers, database management systems, and graphical user interfaces. It thus became increasingly clear that ICL's engineering resources were best employed not on the development of individual hardware and software components, but on integrating these components together to produce total systems.

However, although the phrase systems integration was much used, the process was not well understood. Much of the development of OPEN*framework* was concerned with improving this understanding.

For some years ICL had attempted to address the challenge of producing an integrated product line through a number of technical strategies, each under the control of a senior architect. Most of these strategies spanned products in a number of different development divisions. The strategies included:

- Interconnection
- Interworking
- Message Handling
- Terminal Connection
- Systems Management
- Information Management
- Human Computer Interface
- Security
- Knowledge Engineering

These technical strategies were not product programmes as such. Each was backed by a modest budget, which was used to stimulate activities such as standards work and technology transfer. Product programmes were assessed for conformance with the technical strategies through a phase review process, but ultimately individual product managers (or their customers) had the final say.

There were also strategies maintained within individual development divisions that served a similar role; these included Office Automation, Management Support, Application Development Tools, Transaction Processing, and many others. These were easier to manage because any conflicts of interest could be resolved locally. In many cases such a strategy was complemented by an architecture, a programme plan, and a budget, but in other cases the strategy formulation and execution did not align neatly.

There were several difficulties with this situation:

- It was not always easy to translate strategy into action, especially when those responsible for action had many tactical objectives and constraints to reconcile with the strategic direction.
- It was difficult to understand how all the strategies related to each other: how could one determine whether there were gaps, overlaps, or inconsistencies? Even the attempt to group strategies into those providing standard infrastructure and those producing technical differentiators had not been notably successful.
- Because the strategies were required to take a long-term view, they tended to focus on predictable trends rather than on discontinuities in technology or in the market-place. Many of the important new product introductions (such as ICL's entry into the personal computer market-place, or the relational database programme) happened without any input from the technical strategy process.

IBM's launch of Systems Application Architecture (IBM 1987) did not go unnoticed, either by ICL or by its customers. We realised that we did not need a direct equivalent to SAA, since that role was more than adequately filled by the X/Open CAE, but we did recognise that we needed some kind of unifying framework to tie together the variety of product lines and strategies being pursued, and to establish some coherence.

During 1989 there were a number of attempts by different parts of the company to define a top-down view of the world into which all these separate strategies, architectures, and programmes could be positioned. A significant influence was an internal paper written by a marketing team, defining what ICL needed to do to achieve its aspiration of becoming a leading systems integrator. This led to a programme code-named Lisbon. Lisbon rightly recognised that architecture was only one part of the solution, and most of the effort went into organisational and cultural projects rather than technical proposals. Thus Lisbon, like Cirencester before it, failed to make any great technical impact, but succeeded in establishing the management climate that made OPEN*framework* possible.

Our architectural thinking was still being developed under the IPA umbrella. IPA had gradually expanded in scope from its original role as a networking architecture to a comprehensive set of standards and profiles covering every aspect of information technology. For example, it now included standards

for user interface styles and for usability engineering. But in 1989 there was an urgent need for an overview describing how all this fitted together; it was impossible to find out what IPA was without reading several thousand pages of specifications.

Work to produce an accessible description of the architecture started in earnest with the creation of a Systems Integration Division early in 1990, which brought together many of the architects around the company as a single team. The architecture was originally referred to as IPA-X, reflecting its evolution from IPA. But market research revealed that there was little general understanding of what IPA had become; we even discovered ICL publications that used the term IPA to refer exclusively to the proprietary ICL networking protocols that predated OSI. The absence of an accessible overview of IPA made this hardly surprising.

The name OPEN*framework* came from a quite different source: it was introduced in 1990 by ICL's UK sales division as the unifying theme of a product launch. The name was subjected along with other candidates to market research, and was selected because it registered high levels of recognition and understanding.

Our architectural work was not done in isolation, of course. It was influenced by many other activities in which we participated with other companies. The Advanced Network Systems Architecture (ANSA) collaboration, for example, (Herbert, 1987) provided some of the theoretical underpinning that influenced our thinking, and our work on the ISO Open Distributed Processing (ODP) committees taught us how to make some of the rigorous concepts in ANSA more readily accessible, no doubt in a way which its originators found distressingly pragmatic. Other influential activities included work on the BSI DISC Framework for User Requirements, the CCTA Framework for Open Systems, the Object Management Group (OMG) Architecture, and committees in the European Computer Manufacturers' Association (ECMA), the European Workshop on Open Systems (EWOS), and the US National Institute of Standards and Technology (NIST).

If we didn't need our own version of SAA, because we already had the X/Open standards, what did we need? We needed an understanding of the process of systems integration.

Key to understanding this process is an appreciation of where costs are incurred and where value is added. Competitive advantage comes from being able to integrate new components very rapidly, from being able to manage variety, and from having an inventory of reusable assets that can be harnessed to produce customised solutions to a wide variety of problems.

This kind of systems engineering depends on the existence of standards, but standards alone are not enough: there need to be an overall architecture

showing how the components relate to each other functionally, and an engineering discipline for ensuring the quality of the final system in terms of, for example, its performance, reliability, and usability.

In developing the architecture, an important realisation was that the various strategies and architectures already in existence could be divided into three categories:

- those concerned with the overall picture as seen by a particular class of user, for example the application developer or the system manager;
- those concerned with a particular assembly of technical components into a functional whole, for example message-handling systems;
- those concerned with the achievement of particular system attributes such as security and usability.

It seemed that one could look at systems from the outside in by considering first the various people and their perspectives, then the system attributes or qualities as measures of the extent to which the system met these people's requirements, and finally the technical structure of the information system.

This led to the development of OPEN*framework* perspectives, qualities, and elements. These are illustrated in Figure 2, which shows the eventual four perspectives surrounding the five OPEN*framework* qualities, which in turn surround the structure of eight elements.

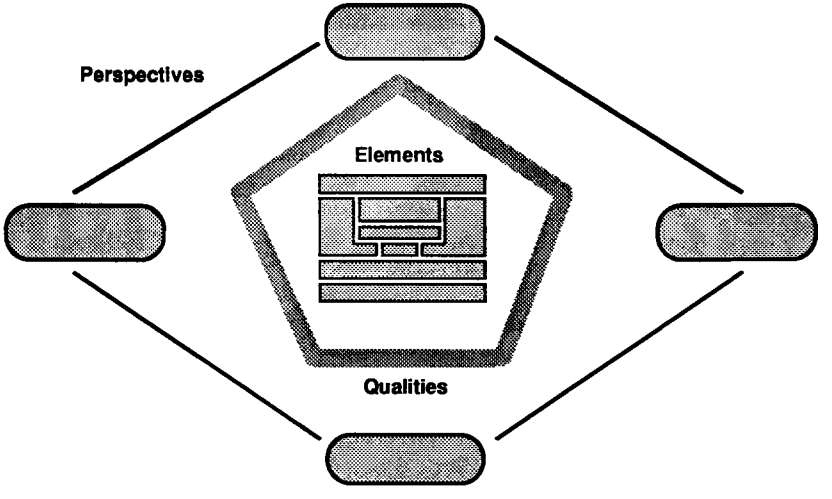


Fig. 2 Perspectives, qualities, and elements

One aspect of architectural thinking that was very prevalent at the time was the so-called "three-ring circus", variously represented as a classification of applications into corporate, departmental, and personal, or a classification

of machines into mainframe, mid-range, and workstations, or more crudely as a division of ICL's product line into VME, UNIX, and DOS. This was sometimes illustrated as shown in Figure 3.

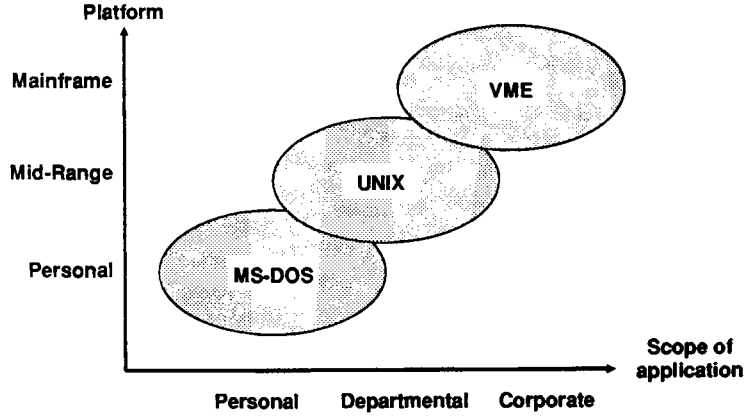


Fig. 3 The three-ring circus

We consciously avoided making this trinity a major part of OPEN*framework*, for a number of reasons:

- Although the spectrum from personal to corporate might be real enough, we felt that there was no intrinsic reason to believe that there should only ever be three noteworthy points on this spectrum.
- The hypothesis that the organisational scope of an application is related to its demand for computer platform resources is untenable: there are many examples of applications that are narrow in their organisational impact yet hungry in machine resources, and the opposite is equally true.
- We wanted to encourage people to think top-down, starting with user requirements and leaving technology choices as late as possible. This meant that the discontinuities between regimes such as MS-DOS, UNIX, and VME, real though they were, should be recognised as arbitrary and architecturally undesirable, rather than being promoted to the status of fundamental architectural boundaries. In fact, as demonstrated by (Coates 1993), a major objective of some of the current product programmes is to demolish this architectural boundary.

During this stage it was also recognised that OPEN*framework*, as ICL's vision for open systems integration, needed to be much more than an architecture. It needed to include methods, tools, and information to enable real systems to be built from real products. These wider aspects of OPEN*framework*, however, are outside the scope of this paper.

5 Stage Four – Information Systems supporting Enterprise Strategy

The final stage in the evolution of OPEN*framework*, which is still underway, is in applying the architectural framework to the challenge faced by enterprises in planning their information systems strategy. In this stage we move from a static architecture, one that describes the state of the system at a given point in time, to a dynamic architecture, one that explains how the users' system evolves to meet the changing needs of the business.

This stage is particularly concerned with the development of one of the OPEN*framework* perspectives (enterprise management); with one of the qualities (potential for change); and with one of the structural elements (application architectures).

Enterprises are faced with a paradox. Their business is changing rapidly; the technology is changing rapidly; yet their information systems are extremely inflexible. They want to be able to exploit technical innovation to achieve competitive advantage, yet in too many cases information technology seems to inhibit change. An example from the Financial Times, 9th March 1992:

> *"Sabre attempted a merger last year with Amadeus. The plan foundered because the computers were incompatible."*

An important insight at this stage is that architecture is not about limiting variety and change, it is about enabling it. The unspoken aspiration of architecture and standards work, despite the caveat quoted earlier from (Brenner, 1980), has often been that one day all the world would converge on a standard network, a standard operating system, a standard programming language, a standard relational database, and a standard user interface. But to think this way is to forget that business is all about change.

The realisation that architecture is there to enable change suggests the answer to another problem. Since enterprises of any size buy information technology from a number of suppliers, how can any one supplier define an architecture for the information system as a whole? This problem is stated clearly in the introduction to (CEC, 1990), a report by the European Commission defining its own informatics architecture:

> *"Most architectures have been designed by the computer manufacturers to match their present and future product ranges. Such architectures are, generally speaking, mutually incompatible even where standards are used. A vendor-independent architecture can only be developed by the user organisation being the customer of the IT industry – but no single customer has the power to impose a given architectural design on industry. Consequently, such an architecture must emerge from the ongoing process of supply and demand."*

ICL too realised that information systems architecture depends on the interplay of market forces, and that we were in no position to impose specific products and standards on our users. The solution is an architecture that starts at a very high level of abstraction, and that can be incrementally specialised and instantiated with standards and products selected at every point in the value chain, from any supplier – ultimately, by the user enterprise itself. The way in which OPEN*framework* incorporates these ideas is described in a companion paper (Pratten and Henderson, 1993) in this issue.

We believe we have reached the point where OPEN*framework*, far from being designed to match ICL's current or future product range, could in principle be used to design systems that included no ICL products at all. Much of the development of the architecture during 1992 has been concerned with this separation between the abstract, vendor-neutral architecture, and its concrete realisation in ICL products.

5.1 The Enterprise Management Perspective

ICL's thinking in this area is heavily influenced by the Management in the 90s (MIT90s) research programme (Scott-Morton, 1991), which investigated the factors that would make enterprises successful in the 1990s, and in particular the way in which successful enterprises would exploit information technology.

Some examples of the application of this research are given in other papers (Craig, 1993; Thurlby, 1993) in this issue.

An important concept that emerges from this work is the idea of strategic alignment: that is, the relationship of business strategy to information technology strategy, and the relationship of the enterprise structure to the information system structure. The principal discovery is that it is no longer adequate to think of information technology following in the wake of decisions about business strategy. Instead information technology can be used to re-engineer the internal structure of the enterprise, to establish new trading relationships, and ultimately to redefine the scope and nature of the business itself. This is illustrated in MIT90s by Figure 4, which depicts five levels of business transformation wrought by information technology. The transition between level 2 and level 3 marks a watershed: at this point information technology is no longer being used merely to improve the efficiency of the business, but to change the nature of the business.

It follows that from the enterprise management perspective, the most important quality of the information system is potential for change. This requires flexibility to incorporate new technology, and flexibility to meet new business needs. Open systems are important not because they enable interworking or because they reduce cost, but because they provide this flexibility. (See also (Gray, 1991).) It follows that variety and heterogeneity within the system are not disasters to be avoided at all costs, but symptoms of a healthy ability

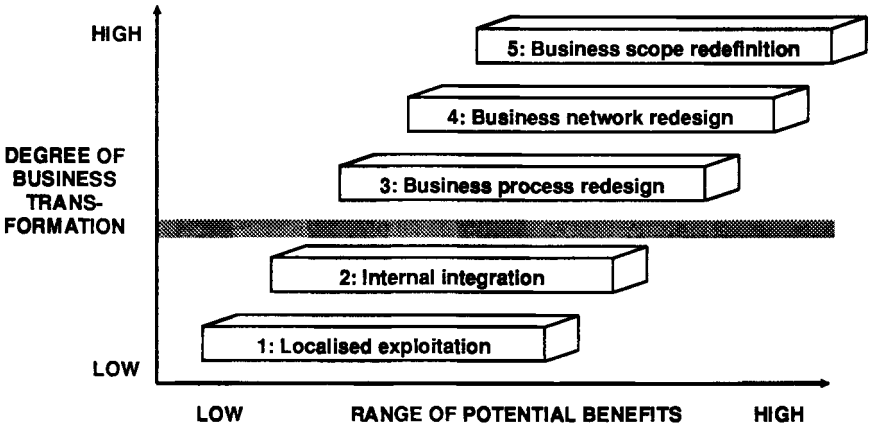


Fig. 4 Role of information technology (after Fig. 5.3 on p. 127 of Scott–Morton 1991 loc. cit. By kind permission of Oxford University Press, New York).

to absorb change. Therefore, the objective of an information technology strategy should not be to eliminate variety, but to ensure that variety is affordable.

5.2 *Potential For Change*

OPEN*framework* identifies five qualities that must be achieved in an information system if it is to meet user requirements:

- Availability
- Usability
- Performance
- Security
- Potential for change

All of these require engineering attention at every stage of system design, construction, and management, and in each case there is a repertoire of techniques available to the systems engineer to ensure that the system meets its goals. An example is the engineering process for achieving performance requirements, described in a forthcoming paper by Baker *et al.* in this journal.

But in one respect potential for change is different from the other qualities. The other four qualities are static attributes of the system, that can be measured or evaluated at a point in time. Potential for change is dynamic, and is very difficult indeed to measure.

Variety and change go hand-in-hand. An enterprise installs functionally-similar products from a number of suppliers either as a consequence of having an organisation that is empowered to make local decisions so that it can change rapidly; or as a result of business change in the form of mergers

and acquisitions; or as a deliberate strategy to remind suppliers that it is able to change its sources of supply. In other words, variety either results from change or is introduced to enable change.

The main way in which standardisation enables change is that it allows one component of the system to be modified or replaced without affecting every other component of the system. For example, the adoption of the X/Open common application environment has made it possible to introduce RISC processors without affecting existing applications; the existence of an SQL standard makes possible innovations such as ICL's relational database accelerators.

But standards themselves are subject to change and variety. The number of standards in use is doubling every three or four years. There are several reasons for this:

- Different standards impose different trade-offs among the OPEN*framework* qualities. For example, the C and PROLOG programming languages have different usability and performance characteristics. So there is rarely a single standard that will meet all requirements.
- Standardisation initiatives usually start within a particular community, but then expand in scope to embrace the requirements of other communities. Thus different standards start to overlap. For example, in the field of character coding standards the ISO 6937 standard emerged from the data communications community while ISO 8859 emerged from the data processing community.
- While a given standard enables technical innovation within its scope, it also inhibits more radical innovations, and these will eventually cause it to be superseded. For example, the C programming language might eventually be superseded by languages that are better suited to parallel processing.

This diversity means that few enterprises of any size can afford the luxury of imposing a single standard across all their information systems, whether it be a standard programming language, a standard networking protocol, or a standard user interface style. If they attempt to do so, they are almost certainly inhibiting innovation to an undesirable extent. But if users cannot achieve homogeneity within a single enterprise, a supplier like ICL certainly has no chance of imposing a single set of standards on all its users.

The existence of a variety of standards, like the existence of a variety of products implementing each standard, is essentially a healthy phenomenon in the industry, because without it innovation would be stifled and the scope for enterprises to gain competitive advantage would be reduced. Each standard has its own strengths and weaknesses, it makes different trade-offs among the OPEN*framework* qualities. It is therefore entirely proper that different standards should coexist.

Because multiple standards will always exist in an information system, it follows that standardisation cannot be the whole answer to achieving potential for change.

Standardisation can be seen in terms of an abstract type system. A standard is a type definition; products implementing that standard are instances of the type. Innovation and variety are examples of polymorphism: different products implement the same specification in different ways.

Other aspects of abstract type systems and object-oriented systems are also relevant to potential for change:

- Encapsulation establishes system boundaries that ensure that innovation behind an interface is always possible. The OPEN*framework* structure of eight elements is a top-level decomposition of an information system into encapsulated objects, allowing each to change independently of the others.
- Subtyping allows subsets and supersets of interfaces to be defined, again providing scope for incremental system change. This is represented in the standards world by levelling, for example the three conformance levels of SQL defined in ISO 9075:1992.
- Late binding allows system changes to be introduced without rebuilding existing components. Early binding is a major inhibitor for change. For example, this occurs when a database product is delivered to users with TCP/IP networking code already built into it: substitution of OSI networking code can only be achieved by the database supplier and not by the user, even though the OSI code offers the same programming interface as the TCP/IP code. The architectural reference models in OPEN*framework* indicate the points at which late binding is desirable.
- The process of systems integration which OPEN*framework* supports is essentially a process of object re-use. At every stage in the value chain there is a combination of top-down and bottom-up activity: analysis of market requirements (that is, the requirements of the subsequent stage in the value chain), combined with selection of re-usable objects from the previous stage in the value chain.

5.3 Application Architectures and Specialisations

The way in which OPEN*framework* supports variety and change is particularly evident in the concepts of application architectures and specialisations.

Application architectures refine OPEN*framework* by adding functionality. Specialisations refine OPEN*framework* by adding constraints. Together, these correspond to the two aspects of inheritance in object-oriented systems:

- Extending behaviour by adding operations and attributes
- Limiting possible states by adding constraints on class membership

In many cases these will go hand-in-hand; for example Retail OPEN*framework* adds retail applications to the generic infrastructure, but also constrains the choice of platforms, networking services, information management and user interface technology.

The process of specialisation is illustrated in the OPEN*framework* introduction (Brunt and Hutt, 1992) by the diagram in Figure 5. The darkened areas indicate aspects of the architecture where additional constraints have been imposed. The projecting areas indicate functionality that has been added to the basic architecture.

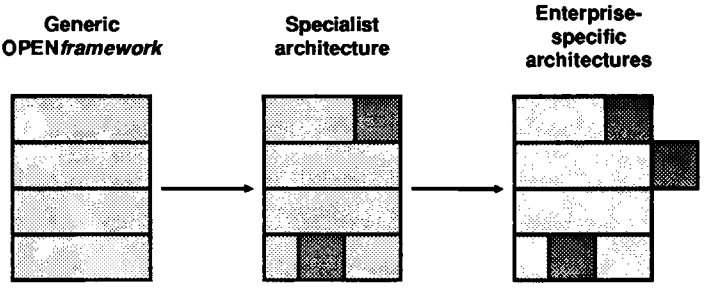


Fig. 5 Specialisation of the architecture

The idea of an application architecture is that at each stage of the value chain something (an application) is added to the system, but at the same time constraints are applied to the environment that the application will run in. For example, it might be constrained to run on specified platforms, to use particular networking infrastructure, and to use a particular user interface style. The sum total of these constraints is referred to as an OPEN-*framework* specialisation.

Specialisation is a recursive process. The first stage of specialisation identifies particular classes of application, for example multimedia applications, or transaction processing applications, and defines constraints appropriate to these areas. Further specialisations can be defined for specific industries or other communities.

6 Summary

The paper has identified four stages of architectural thinking that influenced the development of OPEN*framework* in its present form:

- The need for a coherent networking architecture
- The need for a common application environment
- The need to produce integrated systems from externally-sourced components

- The need of enterprises to achieve competitive advantage through innovative use of information systems, leading to a requirement to enable variety and change

The aspects of OPEN*framework* that contribute most to this final stage are the enterprise management perspective, the potential for change quality, and the concepts of applications architectures and specialisation.

Throughout this period the concept of an open system has evolved considerably, but remains a key theme within the architecture.

This description of a neat historical progression is of course a simplification. Most of the ingredients of OPEN*framework* are the result of a long tradition of architectural thinking, and some of them (such as the security model) developed along lines that do not fit neatly into these phases. There were other activities not described in this paper that proved ephemeral, such as the attempt to define a converged architecture for computing and telecommunications in the early days of the ICL-STC merger. But the overall conclusion remains valid: OPEN*framework* has emerged from a tradition of architectural thinking through a number of phases of gradually increasing scope, all designed to deliver integrated systems supporting business requirements.

Acknowledgements

The ideas described in this paper have been stimulated, and in some cases generated, by my colleagues among the OPEN*framework* Company Architects and the ICL Fellows: particularly (but not exclusively) Andrew Hutt, Barry Pearson, Graham Pratten, Brian Warboys, and Peter Wharton.

Dave Hollingsworth provided useful ideas on the structure and content of the paper, and John Brenner, Jack Houldsworth, and Colin Taylor contributed points of detail.

References

BAKER, C.E.J., PICTON, R.D., SANDERS, K., and SUTCLIFFE, S.E. OPENframework in action: Engineering for Performance at the Inland Revenue. (To be published)

BRENNER, J.B. Using Open System Interconnection Standards. *ICL Tech J.* 2(1) pp. 106–118, 1980.
BRENNER, J.B. IPA Networking Architecture. *ICL Tech J.* 3(3) pp. 234–249, 1983.
BRUNT, R.F., and HUTT, A.T.F. (eds). *OPENframework. The Systems Architecture: an introduction.* Prentice Hall, 1992.
CAMPBELL-KELLY, M. *ICL, A Business and Technical History.* OUP, ISBN 0–19–853918–5, 1989.
COATES, P. VME-X; Making VME open. *ICL Tech J.* 8(3) pp. 473–491, 1993.
COMMISSION OF THE EUROPEAN COMMUNITIES (CEC). *Guidelines for an Informatics Architecture.* Fourth Edition, ISBN 92–826–0275–3; Catalogue number CB-58–90–667-EN-C, 1990.
CRAIG, I. OPEN*framework* in Action at DEVETIR *ICL Tech J.* 8(3), pp. 398–415, 1993.
GRAY, P.A. *Open Systems: a Business Strategy for the 1990s.* McGraw-Hill, ISBN 0–07–707244–8, 1991.
HERBERT, A. The Advanced Network Systems Architecture project. *ICL Tech J.* 5(4) pp. 638–651, 1987.
HOULDSWORTH, J. Standards for open-network operation. *ICL Tech J.* 1(1) pp. 50–65, 1978.
HOULDSWORTH, J. OSI Migration. *ICL Tech J.* 6(1) pp. 88–106, 1988.
HOULDSWORTH, J. Open Networks – The Key to Global Success. *ICL Tech J.* 8(2) pp. 179–197, 1992.
IBM. *Systems Application Architecture: an Overview.* First edition, May. GC26–4341–0, 1987.
KEMP, J., and REYNOLDS, R. The ICL Information Processing Architecture IPA. *ICL Tech J.* 2(2) pp. 119–131, 1980.
PRATTEN, G.D., and HENDERSON, P. Creating Potential for Change. *ICL Tech J.* 8(3), pp. 383–397, 1993.
SCOTT-MORTON, M. *The Corporation in the 1990s.* OUP, ISBN 0–19–506358–9, 1991.
TAYLOR, C.B. The X/Open Group and the Common Application Environment. *ICL Tech J.* 5(4), pp. 665–679, 1987.
TAYLOR, C.B. X/Open – from Strength to Strength. *ICL Tech J.* 7(3) pp. 565–583, 1991.
TAYLOR, C.B. *X/Open and Open Systems. X/Open Company* Limited, ISBN 1–872630–55–3, 1992.
THURLBY, R. Strategic Information Systems Planning. *ICL Tech J.* 8(3), pp. 416–437, 1993.

Biography

Michael H. Kay

Michael Kay read Computer Science at the University of Cambridge, staying there to complete a Ph.D. in database management.

He joined ICL in 1977 to work on the development of VME IDMSX, subsequently becoming chief designer on that project. He led the design team for the ICLFILE document retrieval system, subsequently marketed within the OFFICEPOWER suite under the name POWERFILE, and acted as chief architect for the cross-divisional programme implementing the relational database INGRES.

More recently he has been involved with the development of a new dictionary product, and a prototype object-oriented database designed to underpin it, both described in recent articles in the ICL Technical Journal.

Dr. Kay was appointed an ICL Fellow in 1989, and is currently a member of the OPEN*framework* architecture team, as Company Architect for Information Management.

Creating Potential-for-Change

Graham D. Pratten
Company Architect, Potential-for-Change, OPEN*framework* Division, ICL
Peter Henderson
Department of Electronics and Computer Science, University of Southampton
ICL Visiting Fellow

Abstract

Potential-for-change is the ability a product has to evolve to meet new market demands and to exploit new technical opportunities. This paper considers how the IT industry is providing this quality in the IT products it develops and in the end-user systems which contain these products.

It concentrates its attention on the value-chains and processes which are used in developing and evolving IT products and systems and considers how these can be made change-oriented. It briefly considers how the products and systems themselves may be given change-oriented-architectures which make them amenable to change. It finishes with a look at the forces-for-change in user enterprises which are generating the need for potential-for-change.

1 Introduction

The IT industry currently exhibits a strange contradiction. On the one hand the market demands on the industry and the technical opportunities available to it are as strong as ever. On the other hand the IT industry seems unable to move fast enough to cope with existing demands and opportunities let alone new ones. IT vendors find they cannot improve their products fast enough. End-user enterprises find it difficult to evolve the IT systems into which these products are integrated. They complain that it takes longer to change their IT systems than it did to change the manual predecessors of these systems.

Thus there is a strong forward impetus on the IT industry coming from market-pull and technical-push but a strong drag back on the industry coming from the legacies of its history.

The IT industry can build into its products and systems the ability to evolve to meet new market demands and to exploit new technical opportunities. A number of phrases or words have been used to characterise this quality, such as future-proofing, changeability, enhanceability, evolveability and flexibility. This paper uses the phrase potential-for-change which was coined for this quality in ICL's OPEN*framework* architecture.

Other industries are changing the way they develop products (Womack et al., 1990) in order to achieve potential-for-change and other desirable characteristics like quality, time-to-market, flexible manufacturing, low inventories and just-in-time. These changes are now beginning to affect the IT industry. Companies involved in the development of IT products and systems are becoming linked in complex value-chains (Porter, 1985). Each stage in a value-chain maintains its own asset-base of products, components and skills. It receives products from earlier (supplier) stages, incorporates them into its asset-base, adds value to its asset base by engineering, integrating and reengineering, and passes products on to later (customer) stages. The supplier and customer relationships between companies in value-chains are being modified to achieve potential-for-change and other desirable characteristics.

Sections 2 to 4 of this paper consider how the potential-for-change quality can be catered for within the value-chains and processes of the IT industry, what architectures can be given to IT products and systems to make them amenable to change, and what forces-for-change are coming from the enterprises which use IT systems. In practice, of course, it is the change in enterprises that drives the change in IT products and systems and this in turn drives change in the value-chains and processes used in their development. The material is presented in the reverse order because, for presentation purposes, this order is more natural.

2 Change Oriented Processes and Value-chains

2.1 New Style of Development

In the last ten years the IT industry has fundamentally changed its approach to development. The change can be characterised by the following trends:

technical-push on industry→market-pull on industry

single vendor supply→multi-vendor supply

collaboration within companies→collaboration within value-chains

support within companies→support across value-chains

general purpose companies→teams specialising in products or markets

small teams→large teams→collaborating small teams

serial flow of development→concurrency in development

developing products in their entirety→building them from smaller products

developing new →evolving existing

innovation everywhere→innovation based on standards

importance of components→importance of integration technology

profit from products→profit from services

In moving into its new style of development the IT industry has made a move of significance, similar to that made by the car manufacturing industry when it moved from an engineering-garage to a manufacturing-line development process.

Unfortunately, the IT industry has no model of its processes which takes it beyond the simplistic waterfall and V-diagram models (Boehm, 1981) which served an earlier age and only has methods and tools emphasising these models. As a result, it is rich in methods and tools for requirements analysis, design and programming but lacks methods and tools for supplier analysis, re-engineering, reverse-engineering, reuse, integration, customisation and tailoring. It has customer-care courses but no supplier-care courses. The industry has evolved into a new style of development process but has not developed a way of modelling, understanding, controlling and supporting this new style.

The next few sections look at the new style of development hinted at above. If the industry is to create potential-for-change in its products and systems it must do it within the context provided by this new style as this style becomes predominant within the industry.

2.2 Life Cycles of Products

The new style of development process revealed itself clearly in the development of the IBM compatible PC product. Figure 1 below illustrates the lifecycle of this product.

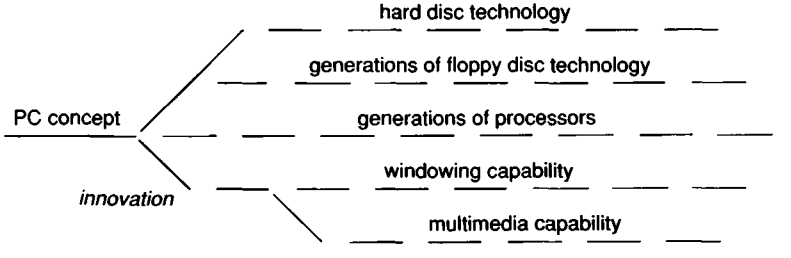


Fig. 1 Lifecycle of PC product

The definition of a standard for PCs began with IBM's definition of its own standard based on the existing de-facto PC standards. It defined or adopted standards for some of the components within the product such as MSDOS, discs and processor code. It started to organise the straightforward relationships with its suppliers which it had had for earlier products. Some components, such as the processor, settled down to a steady series of evolutionary steps with a conventional relationship between PC-product developer and component supplier. However other manufacturers began to produce clones of the whole product. This allowed some components, such as those for windowing, to gain a life of their own with their suppliers taking independent and innovative positions. This in turn took the whole product off in new revolutionary directions. These new developments eventually settled down into steady evolutionary progress. However by then the new developments such as Windows had become more important than the original product. One is even seeing the relationship between MSDOS and Windows being shifted by the supplier. Now MSDOS and its applications are products which can be run under the control of Windows rather than the other way round.

The PC product has reached the point where no vendor controls its future. It evolves forward with a complex mixture of competition and collaboration, innovation and standardisation. The same approach is being applied to the development of UNIX and other products. It is even being adapted by some end-users enterprises for the evolution of their IT systems. (This has been called just-in-time acquisition, end-user systems evolving forward exploiting whatever the market throws up). To some extent this new style of development can be orchestrated by open systems consortia like X/Open (Gray, 1991) and OSF but it can never be controlled by a single vendor. The new style is gradually being recognised and formalised as the way products and systems will be developed and evolved in the future.

There are some lessons for the potential-for-change quality to be learnt from the history of the PC-product apart from those already listed in Section 2.1.

Developers must know where their products, systems and components are in their life-cycles, whether they are innovative and in their infancy, or mature and ripe for standardisation, or somewhere in between (Foster, 1985). For instance in the mid-80s the processor code design in the PC product was mature and standardised whilst windowing was in its infancy and innovative.

Many products in the past have proved to be unscaleable and so become redundant because deep within the product there was an assumption about the size of some resource (probably represented in a one byte field) which eventually proved to be too conservative. Developers can predict the effects of continuous changes in their products and systems and prepare for these changes. For instance it is possible for PC developers to predict and plan

for the continuous improvement in processor speed and store size ten years ahead.

Developers can try to predict discontinuous changes in their products and systems. For instance with some foresight PC developers in the early 1980's saw the importance of windows developments. Now they can predict the importance of developments in multi-media and group working and prepare for their arrival.

The history of the PC also gives insight into the interplay of standardisation and innovation. Standardisation of the PC product and its components, like earlier acts of standardisation (Hawke, 1989), have not cramped innovation. Innovation is alive and well, the annual number of patents has increased ten-fold in twenty years. A general rule seems to be that standardisation restricts innovation where it is applied but more than makes up for this by stimulating innovation in surrounding areas. Developers can look for parts of their product which can be standardised and identify the innovative possibilities this will unleash.

2.3 V-diagram Model

The traditional view of development is represented in the various versions of the V-diagram model in Figure 2 (McDermid, 1991) (Boehm, 1981) (DTI, 1987).

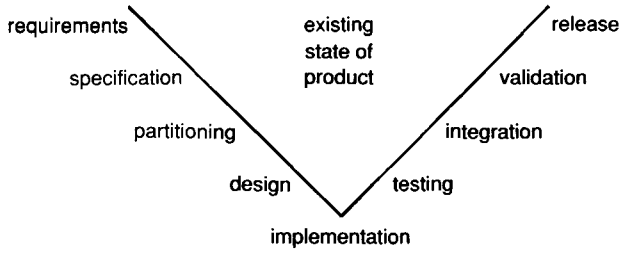


Fig. 2 V-diagram

It assumes that development begins with a requirement for a product or system, this is then subdivided into requirements for components of the product or system, each of these is then designed, implemented and tested separately, the components are then integrated into a release of the product or system satisfying the original requirement.

Of course products and systems are not implemented once; they evolve in a sequence of releases spread over many years. The V-diagram model can be modified to cope with this; the modification is often referred to as the spiral model (McDermid, 1991) (Boehm, 1988). Each loop round the spiral (or pass through the V-diagram) creates one release. The starting point for each loop is a release produced by an earlier loop plus some requirements

for changes to that release. Releases are allowed to develop in parallel with each other. For instance one release may be at the requirements analysis stage whilst another is at the integration stage.

The V-diagram style of development emphasises the requirements analysis, specification, design, implementation, testing and integration activities.

Products were designed to make them easier to change within the context of the V-diagram style. An early move was the concept of modularity to allow components to be independently designed, implemented, compiled and tested before being bound together. Another early development was configuration management to manage the configuration of versions of a system or product from versions of its components.

An important design concept was separation-of-concerns which said that features with different change characteristics ought to be kept separate from each other within a design, ought to be implemented in different components, and should be bound together late in the development process. Over the years people have gradually improved their understanding of the phrase "different change characteristics". It includes reason for change, frequency of change, skill level required for change and skill type required for change.

Components within systems and products were also designed to facilitate change. For instance the number of parameters or order of parameters were made changeable so that the capabilities of a component could be enhanced without propagating change into all the components using its earlier capabilities. Or for instance each version of a component was designed so that it could be configured with a number of versions of other components.

2.4 Concurrency Within the Process

From a potential-for-change point of view a major constraint is the time taken for one loop in the spiral. With a large product or system developed by a large team it can take several years to pass round one loop. Over the years attempts have been made to encourage incremental development within this style, to encourage frequent loops each achieving comparatively modest changes. This may or may not have been augmented with incremental delivery of the changes to users.

Another problem from a potential-for-change point of view is the level of concurrency between releases, the fact that there will be a number of different releases at different stages in their development, one or more already in use, one being designed implemented and integrated, and one still in the requirements analysis stage. A requirement will tend to be defined against the release already in use but will in fact be satisfied in a much later release, perhaps months or even years later. Some changes will be capable of passing round the loop more quickly than others so changes may initially be planned for integration in one release but may then need to be moved to earlier or later releases.

The initiatives on concurrent engineering in the USA (Reddy et al., 1992) are concentrating on this problem of concurrency from another standpoint. The aim is to improve time-to-market from manufacturing processes in the USA in order to make them more competitive. The initiative recognised the serial nature of existing processes, products passing through a series of steps (such as those in the V-diagram) on their way from concept to realisation. It recognised that time-to-market could only be improved if concurrency was maximised within the process. For instance a car manufacturing plant can be designed and set up, or the marketing launch of the car can be planned, whilst the car is being designed and does not have to wait until it is designed. In order to achieve this they recognised they wanted:

1. *Virtual team support.* The people involved in the development process can be given electronic support which enables them to cooperate as a close knit team even when they are dispersed physically and organisationally. This cooperation is required across the development process between different types of engineer e.g. mechanical, electronic, IT, etc. engineers and down the development process between designers, marketeers, manufacturers, etc.
3. *Shared models of products.* If different people within the development process are to be able to communicate with each other they need to have shared understanding, i.e. shared models, of the products they are developing. These models have to be shareable across and down the development process.
4. *Design for marketing, manufacturing and maintenance.* Engineers can design their products so that they are easy to market, manufacture, validate and maintain. Marketeers, manufacturers, and maintenance people can be involved in the design activity at an early stage and share objectives for the products they are developing
5. *Operator in the loop.* Early in the development process the existence of a powerful simulation of the product enables the product to be validated by its future operators (i.e. users). This allows the users to be represented in the design activity as well as the developers
6. *Quality control.* The development process can be designed so that it detects errors at all stages (not just the end and the beginning) and modifies its behaviour appropriately

These ideas are as applicable to the V-diagram model of development (and the models of development shown in later sections) as they are to USA manufacturing processes.

2.5 Integration Chain

During the second world war the number of components in a bomber declined markedly. The appearance and use of the bomber did not change greatly; the objective was to simplify and accelerate the production of bombers. A similar thing has happened in the car manufacturing industry (Kumpe et al., 1988). Car manufacturers are simplifying their development

problems by reducing the number of components in their cars. They are forcing manufacturing problems back on to their suppliers. The car manufacturer sees a smaller number of components and these components are no more complicated for them than the earlier components. Much of the complexity of the car has been shifted into its components and is now the worry of the suppliers. The car manufacturer and its suppliers and their suppliers are organised in an integration chain as in Figure 3.

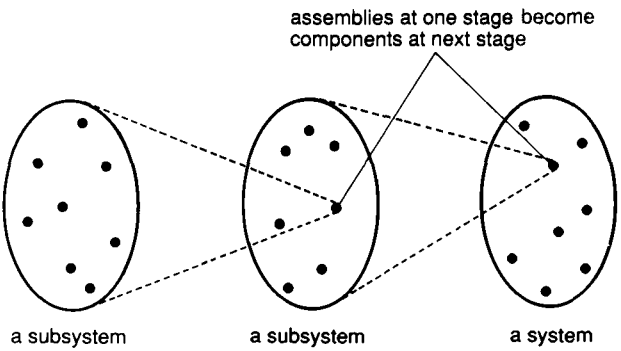


Fig. 3 Integration chain

Each stage in the chain only has to worry about a reasonably small number of components; what is a simple component to one stage in the chain will be an assembly of components to an earlier stage. This means that each stage in the chain can move round its development loop more quickly because the complexity of its problems has been contained at a reasonable level. Hence it is better able to respond to demands for change in its products.

The integration chain style of development is now being applied in the IT industry. It could be represented as a chain of V-diagrams with requirements and releases passing between the stages in the chain. However to be truly effective the integration chain has to introduce new types of development activity not included in the V-diagram model. For instance one stage in the chain may receive components from an unreliable supplier. It may have to re-engineer or reverse engineer the components it receives. It may be capable of taking the same type of component from multiple sources. A better model of the typical node in the integration chain is required. This is given by the asset pumping model shown in Figure 4.

It sees each stage in the integration chain as maintaining an asset base of products and components. It continually receives components from its suppliers re-engineers them, reverse engineers them and integrates them to form new assets in its asset base and new products to pass on to its customers.

Thus in the integration chain style of development supplier-care, supplier choice, re-engineering and reverse engineering become as important as the

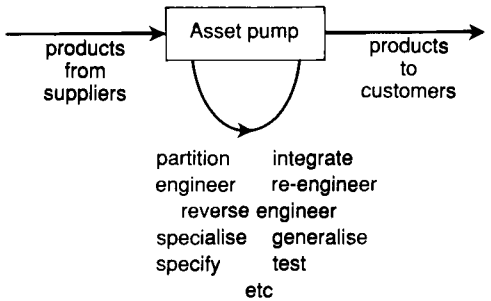


Fig. 4 Asset pump

traditional development activities introduced in Section 2.3. There is more concurrency in development. Different companies will be developing systems and products and their components in parallel with each other. Designs of systems and products have to be able to adapt to exploit new components as these become available from their suppliers.

Suppliers will not normally be suppliers to just one company. Companies cannot expect to control components coming from their suppliers as much as they did when all components were developed in-house under the V-diagram style of development. In the worst case they will have to cope with commodity components which are used so widely that the companies using the components have little influence on the suppliers of the components.

2.6 Customisation Chain

The style of development described in the last section came about because of the demands of the developers. It eased the manufacturing process for complex systems including cars and IT systems. There is a variant on this style of development which is coming into existence to satisfy market demands. This is the customisation style of development shown in Figure 5.

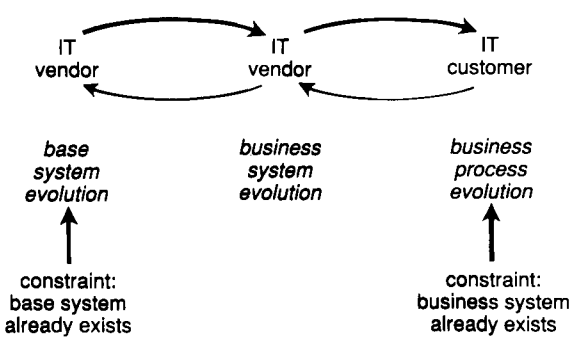


Fig. 5 Customisation chain

A company creates a generic product, for instance a retail point-of-sale device. It then customises it in stages to different market sectors such as DIY or food, different regions such as Australia or France, different customer enterprises and ultimately different sites within each enterprise.

This style of development adds to the picture given in the previous section. It places the emphasis on additional development activities, specialisation and generalisation.

At the end of the customisation chain is the customisation done by end-users. Hence the emphasis at this point in time on user enhanceable systems, systems which can be readily tailored to their requirements by end-users who are IT illiterate (though possibly skilled in other disciplines).

2.7 *Value-chains*

From the previous sections it can be seen that the new style of development involves many different types of vendor collaborating and competing in value-chains (Porter, 1985), (Johnston, 1988). Different stages in the value-chain will concentrate on different types of development activity, design, implementation, integration, reverse engineering, re-engineering, specialisation, generalisation, etc. Figure 6 shows a typical value-chain. It includes independent software vendors (ISVs), value added resellers (VARs), facility managers and systems integrators.

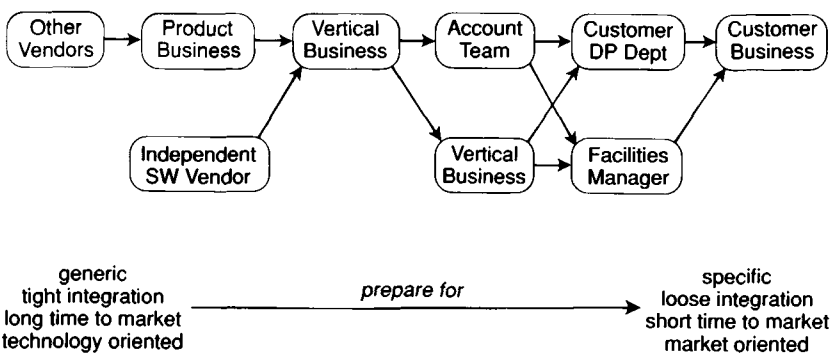


Fig. 6 Value chain

IT products and information systems pass through these complex value-chains on their way to eventual use in customer enterprises.

Each stage in the value-chain takes products, or components thereof, from earlier stages in the value-chain, adds value by integrating, generalising, specialising, re-engineering, reverse engineering, reusing, partitioning, etc. and passes on products to later stages.

Development approaches vary across the value-chains. Products may take years to reach their ultimate market from early stages in the value-chain, months from later stages. Skills in the early stages will tend to be technical oriented, skills in later stages business oriented. Tight, difficult and efficient approaches to integration can be used in the early stages, whilst loose, easy but possibly inefficient mechanisms are required in the later stages. Time-to-market pressures mean that as much responsibility as possible for change must be forced down to the end of the value-chain. Integration done in early stages must be done with later stages in mind; for instance the early stages should not bind features tightly together if later stages will need to separate them. Earlier stages only exist to make change easy in later stages, they provide the generic change mechanisms which can be used later.

The industry is recognising the different development approaches required at different stages in the value-chain and is providing suitable support for them.

Towards the end of the value-chain the problem of change is a problem of scale. The sheer number of machines and packages which have to be changed in an enterprise, the problems encountered with multiple versions of products and interworking between them, the problems of phasing change across people and machines so that it does not disrupt the work of the enterprise. These can be supported by change management capabilities such as those described in the systems management element of OPEN*framework*.

Earlier in the value-chain there are also problems of scale, the large number of components and versions of components used in developing products and systems, the configuration of components into larger components and products, the reuse of components to maximise their value, the reverse engineering and re-engineering of components obtained from many suppliers. These are supported by application development capabilities such as those described in the application development element of OPEN*framework*.

Development units within the value-chains will in future nurse their suppliers and collaborators quite as assiduously as they do their customers. The IT industry is now recognising development as being supply driven as well as market driven.

3 Change-Oriented-Architectures

In the early days of the IT industry people were cheap compared with IT resources. Performance of IT systems was the dominant problem. Systems were structured to achieve IT performance. Now IT resources are cheap compared with human resources so systems can be structured for other criteria. So for instance secure architectures have been developed which emphasise the security quality in systems. The industry is now beginning to develop architectures which emphasise change as the major consideration, that is to say change-oriented architectures.

Pragmatic approaches to change-oriented-architectures are already being adopted. Flexible frontends are being put on to existing, difficult to change, systems. This is being done using office systems, knowledge engineering systems, HCI systems and process support systems. These will continue to be important over the next few years.

However it is now possible to develop a formal approach to change-oriented-architectures. The key principle is that of "separation-of-concerns". Aspects of the system which have different change characteristics are separated from each other so that they can be changed independently. These change characteristics include the reason for change, frequency of change, skill type required to make the change and skill level required. So for instance features which have to be changed frequently by unskilled users are separated within the architecture from features changed infrequently by IT experts. Features changed by database experts are kept separate from those changed by HCI experts. The industry will continue to develop and use these change-oriented-architectures.

If these change-oriented-architectures are to be effective, products have to be engineered or re-engineered according to the rules of the change-oriented-architectures. They have to be made suitable as components within the architectures, in other words made into change-oriented-components. For instance products must not be produced with HCI-driven interfaces which have no equivalent in procedural interfaces. Other products have no way of calling on the services provided by such products as they cannot manipulate the HCI controls as a human and cannot invoke the products via a procedural interface. It is already known that such products are difficult or even impossible to integrate with other products. Products can be given interfaces which are easy to tailor or extend, for instance with a variable number of parameters or with identified rather than positioned parameters. Products can be given object-oriented structures. In its encapsulation mechanisms object-orientation provides good support for separation-of-concerns. It also allows a good match between the real world supported by an IT product or system and the model of that world held within the product or system. Client-server architectures also give good separation-of-concerns in the design of systems. Thus rules for change-oriented-components can be identified and formalised.

There are mechanisms at many levels which can be used to plug products together within the change-oriented-architectures. The interconnect mechanisms at the network level are already well developed. Good interworking mechanisms at the application level are emerging. More human-oriented interworking mechanisms will emerge in the next decade. These will provide IT support for sophisticated interworking within human teams. The industry will continue developing these higher level interworking mechanisms. The industry has, over the years, developed layer upon layer of interworking mechanisms, tending to treat each layer as the last. It is now developing a more analytical approach to the problem. The state of the art in this area

is described in the Distributed Applications Services element of OPEN*framework*.

A subject of increasing importance is user enhanceable systems. How can change mechanisms be made so easy to use that they can be used by relatively unsophisticated users? Something like Visual Basic gives a glimpse of what is possible here.

4 Forces-for-Change

In the 1980s MIT in the USA set up a project called Management in the 1990s to look at the way enterprises would operate in the 1990s and the way they would use IT. The results of this programme (Scott-Morton 1991) are now having a major impact on ICL and other companies just as quality programmes affected them in the early 1980s (Crosby, 1979).

The most important conclusion of the Management in the 1990s programme was that the cost performance of IT is improving ten times as fast as any other technology and so must periodically cause major change in any other product incorporating it and in any enterprise using it.

This is happening in the design of planes, cars and televisions and in banks, building societies and manufacturing lines. IT is becoming embedded at the core of these products and enterprises.

IT is also having an impact on the organisational structure of enterprises. A similar thing happened 100 years ago when telecommunications technology made possible the large hierarchic organisational structures seen today.

IT products and systems are now making possible another major change in organisational structures. These new organisational structures are already being characterised by phrases like cluster working, network organisations, flat organisations, globalisation, concurrent engineering, empowerment and electronic trading.

Management in the 1990s summed up this trend by saying that the business of any enterprise and its IT are now tightly coupled. Neither can advance without the other.

And yet enterprises are finding that they cannot evolve their businesses to meet new market demands and new technical opportunities because they cannot evolve their existing IT products and systems fast enough.

So the end objective is to give potential-for-change to the businesses which use IT. Building potential-for-change into IT products and systems is simply seen as the most important way of achieving this objective.

The forces-for-change described above are forces affecting the enterprises which use IT. However they also affect the enterprises which develop IT. They are bringing about the changes in development processes referred to in section 2.

5 Conclusions

This paper has looked at the ways development processes within the IT industry are changing and the way they are or can be made to facilitate change in the products and systems developed by the industry. It followed this with a brief look at the way change can also be facilitated by change-oriented-architectures and ended with a glance at the forces imposing change on the industry.

The paper noted the new style of development process emerging now and involving a mixture of collaboration and competition, innovation and standardisation. Products are now developed via complex value-chains involving many different types of development unit concentrating on integration, customisation, re-engineering or whatever. The actions required to facilitate change vary down the value-chain. Problems of scale in the later stage of the value-chain can be supported by change management capabilities; in the earlier stages by asset management. Reuse, re-engineering, reverse engineering are increasingly important aspects of development within the value-chain.

The paper looked at ways of structuring systems so that they are amenable to change using change-oriented-architectures, change-oriented-components and integration mechanisms and noted the trend towards user enhanceable systems.

The paper looked at the forces imposing change on IT products and systems, the continuing speed of change in technology and the close coupling between change in IT and change in businesses.

Management in the 1990s treated potential-for-change as the major demand of businesses in the 1990s. ICL's OPEN*framework* is responding by building potential-for-change into the IT products and systems which support these businesses.

Acknowledgements

This paper has drawn on material developed in conversations with Michael Kay, Alun Roberts, Bob Snowdon, Peter Wharton and Brian Warboys and with the OPEN*framework* architects.

References

BOEHM, B.W. *Software Engineering Economics.* Prentice Hall Inc, Eaglewood Cliffs, NJ 07 632 1981.

BOEHM, B.W. A Spiral Model for Software Development and Enhancement. *IEEE Computer*, volume 21, number 5 May 1988.
CROSBY, P.B. *Quality is Free: The Art of Making Quality Certain.* McGraw Hill LC 79–89296 1979.
DTI. *The STARTS Guide*, 2nd edition, NCC Manchester 1987.
FOSTER, R.N. *Innovation: The Attacker's Advantage.* Pan Books ISBN 0–33–029–925–5, 1985.
GRAY, P. *Open Systems, A Strategy for the 1990s.* McGraw Hill ISBN 0077072448, 1991.
HAWKE, D.F. *Nuts and Bolts of the Past: History of American Technology 1776–1860.* Harper and Row (NY) ISBN 0–06–091–605–2, 1989.
JOHNSTONE, R. and LAWRENCE, P.R. Beyond Vertical Integration: the Rise of the Value-Adding Partnership, *Harvard Bus. Rev.* July–August 1988.
KUMPE, E. and BOLWIJN, P.T. Manufacturing: The New Case for Vertical Integration. *Harvard Bus. Rev.* March-April 1988.
MCDERMID, J. *Software Engineering Reference Book.* Butterworth-Heineman 1991.
PORTER, M.E. *Competitive Advantage: creating and sustaining superior performance.* The Free Press ISBN 0–02–925–090–0. 1985.
REDDY, Y.V., WOOD, R.T., and CLEETUS, K.J. The DARPA Initiative in Concurrent Engineering. Concurrent Engineering Research In Review. *CERC, Volume 1*, Winter 1991/2 carriger@cerc.wvu.wvnet.edu, 1992.
SCOTT-MORTON, M.S. (Ed) *The Corporation in the 1990s – Information Technology and Organisational Transformation.* Oxford University Press (NY) ISBN 0–19–506–358–9, 1991.
WOMACK, J.P., JONES, D.T. and ROOS, D. *The Machine that Changed the World.* Rawson Associales ISBN 0–89256–350–8, 1990.

Biographies

Peter Henderson

Peter Henderson is a graduate in mathematics from Manchester University with MSc and PhD in computer science from Newcastle University. He was Lecturer in Computer Science at Newcastle University, visiting scientist at CalTech. visiting research fellow at IBM New York, Lecturer in Computing at Oxford University and Professor of Information Technology at Stirling University and is now Professor of Computer Science at Southampton University. He wrote a seminal book on Functional Programming and a number of important papers on functional programming, formal specification and software engineering and will shortly publish a book on Object-Oriented Specification. He led a number of Alvey and Esprit projects and is currently a member of the DTI/SERC Software Engineering committee. He is a visiting ICL Fellow working closely with Retail Business, OPEN*framework* and other parts of ICL.

Graham Pratten

Graham Pratten is a graduate in mathematics (with part III) of Cambridge University with 29 years experience in the computer industry. He was a programmer on Leo III systems, teamleader for file systems and virtual store systems on the EMAS project at Edinburgh University, database systems and operating systems designer in early ICL 2900 development, designer of CADES CAD system, ICL technical leader in PISA, IOPT and other projects, strategist for STC systems engineering business and an STL Chief Research Fellow and is now Company Architect for Potential-for-Change in ICL's OPEN*framework* team and a member of the Chief Engineer's Office in OPEN*framework* division.

OPEN*framework* in Action at DEVETIR

Ian Craig
Fujitsu Australia Pty. Ltd, Brisbane, Queensland

Abstract

Open Systems have the potential to deliver tangible business benefits; however, putting theory into practice is proving a major stumbling block for many organisations. Late in 1991 ICL and DEVETIR, a large Government Department in Queensland Australia, formed a unique three-year partnership project to develop an Open Systems based architecture using ICL's OPEN*framework*. The goal is to directly link re-engineered core business processes to a highly distributed architecture, the primary attribute of which is its ability to implement very rapidly organisational, process and technology changes. To achieve this goal the project has adopted process-modelling techniques, process support technology and directory-based messaging systems. The human issues however are the most significant critical success factors. The project has the strong support and regular involvement of the corporate management team. This has been combined with an ever increasing investment in the education of both business and IT staff in order to help them to operate outside their traditional zones.

1 Introduction

The DEVETIR/FUJITSU Partnership Project commenced in December 1991. This article was written in November 1992 when the project had completed Phase One and was near to completing Phase Two and therefore reflects the progress made during this period and the lessons learned. The project is currently planning to deliver a business focused project complying with the new architecture by the end of 1993.

All IT architectures will inevitably reflect the differing needs of individual organisations so the purpose of this article is not to describe the architecture proposed for DEVETIR in detail but rather the manner in which it was derived, the issues which arose and the current view of the solutions to these issues.

2 DEVETIR; The Organisation

The reason for DEVETIR being an ideal case study for the development of an OPEN*framework* architecture is implied by its name. DEVETIR stands

for the Department of Employment, Vocational Education, Training and Industrial Relations. The Department is an aggregation of several former Departments now known as Divisions of DEVETIR. It is a $A900m (£400m) per annum operation employing approximately 6,500 people involved in vocational training and support services. DEVETIR is one instance of a major push by the Queensland Government to achieve economies of scale by reducing the number of Government Departments. This trend is itself a reflection of public pressure for more efficient Government and of the influence of information technology.

The rationale for combining the specific Divisions of DEVETIR into one Department is that they all have a focus on workplaces. DEVETIR is responsible for the promotion of safe, healthy, equitable and appropriately skilled workplaces in Queensland. There are six Divisions, each of which contributes to the achievement of DEVETIR's goals in different specialised ways. Their responsibilities are as follows:

Labour Market Reform: assisting employers and employees to produce a flexible and harmonious *industrial relations environment.*

Employment and Training Initiatives: *matching* unemployed *workers to job opportunities.*

TAFE-TEQ: *training* current and future employees *in vocational skills* appropriate to the emerging requirements of employers.

Workplace Health & Safety: *prevention of workplace accidents* by assisting employers to implement self-management systems for health and safety issues.

Workers' Compensation Board: *compensation and rehabilitation* of injured workers.

Corporate Services: provision of *support services* to DEVETIR as a corporate organisation.

3 Project Background

DEVETIR has long recognised that it would be difficult to achieve a cohesive Departmental approach unless the supporting IT systems are integrated in some way. A leading IT consultancy firm undertook in 1990 an IT strategic planning exercise for DEVETIR, which identified a need for a corporate database to provide a coordinated business focus on employers and an Open Systems migration plan to facilitate interworking between the disparate Divisional IT systems. Ultimately DEVETIR also recognised that it would need access to additional skills to undertake a complex project of the kind required to achieve these two goals.

During the last half of 1991 ICL launched OPEN*framework* as its methodology for developing Open Systems IT architectures. DEVETIR recognised the potential for OPEN*framework* to address its requirements. As a result late in 1991 a three year partnership agreement was signed which committed both parties to invest equally in the development of an Open Systems architecture and a corporate database based on the OPEN*framework* methodology. DEVETIR's objectives are reasonably clear-cut. ICL's objectives are to refine the methodology and develop a reference site for further OPEN*framework* initiatives in Australia.

4 Initial Situation

It is widely recognised that in the 1990s there is little possibility of a return to an environment in which little changes. In fact there is ample evidence that the rate of change will accelerate.

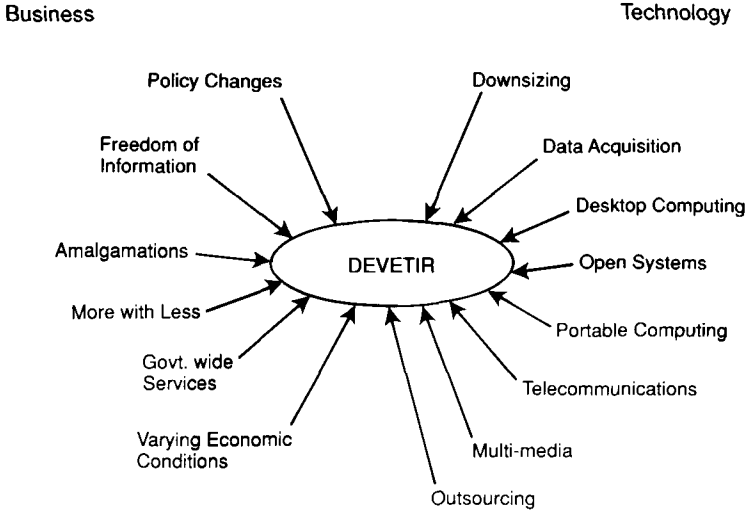


Fig. 1 Pressures on DEVETIR

Figure 1 illustrates some of the issues concerning DEVETIR at this point in time. Neither the project staff nor DEVETIR believe there to be a universal panacea which will address all these issues. Equally, while the list may be different in five years time it is unlikely to be any smaller or any less problematical.

The key point is that management of change is the one pressing and enduring requirement for modern businesses.

The volatility of the situation is illustrated by the following sequence of events: TAFE-TEQ, the largest Division, joined DEVETIR as the project started; the Labour Market Reform, and Employment and Training Initiat-

ives Divisions were formed from parts of TAFE-TEQ and the Industrial Relations Division in August 1992; Fujitsu Australia Limited and ICL (Australia) merged to form a single company in June 1992; the Director General of DEVETIR changed in August 1992; the Government Minister responsible for DEVETIR changed in October 1992.

On the technology side DEVETIR recognises that information technology is increasingly entering areas of business management which are much more significant than simply processing records on databases. Such records, while valuable, can often be secondary in importance to knowledge, and are often in the form of subjective opinions expressed by experienced people, provided and managed through media such as voice, image, graphics and even simple text reports. This is of great importance to DEVETIR in moving to proactive interactions with clients directly at their places of work.

This recognition has a significant impact on architecture design because it demands an open and distributed approach to accommodate a "horses for courses" use of current and emerging technologies.

DEVETIR have a complex and widely diversified IT environment. The IT employed by each Division reflects its "heritage". Workers' Compensation Board (WCBQ) are a fully funded trust fund (i.e., similar to a mutual insurance company) handling claims running to approximately $250m per year. WCBQ runs a claims processing application using TPMS/IDMS on an ICL 3980/2 mainframe currently being upgraded to an SX520.

TAFE-TEQ on the other hand run 32 colleges throughout Queensland and have adopted a distributed approach to their major application, the College Administration System (CAP), being based on 33 Prime Unix systems running INGRES. Workplace Health and Safety, Labour Market Reform and Corporate Services each employ a number of UNIX systems running a mixture of ORACLE and INGRES. The networks employed range across the whole spectrum of TCP/IP, OSI X25, point to point asynchronous links and UTP LANs.

The IT situation at DEVETIR with its inherent incompatibilities is familiar to many large organisations. There are, however, two important business issues which also need to be recognised and addressed in conjunction with the IT issues. The first is that computer systems have an impact on the business processes of which they are a part. Computer systems, such as those at DEVETIR, which were not designed to maximise their potential for change, can enforce inflexible business processes. This creates significant difficulties if these processes are optimised for one part of an organisation but are being redesigned in terms of their Departmental optimisation. Secondly, the merger of several separate Divisions into one Department resulted in business processes, whether manual or computerised, which were duplicated, incompatible or overlapping. This situation significantly limits the opportunities for improving client service and using resources more effectively.

The recognition of these two business issues was to have a fundamental impact on the nature of the Partnership Project since it led to an increasing emphasis on Business Process Redesign and on a flexible architecture which could support the consequential high rate of change.

5 The Project; Phase One

The Partnership agreement gave rise to three initial projects, known collectively as "Phase One", which ran from November 1991 through to May 1992. The projects were the Open Architecture Design Project (OPAD), the Business Process Modelling Project (BPM) and the Systems Audit Project (SAP). The BPM and SAP projects acted as information "feeder" projects under the control of the OPAD project.

The primary objective given to the project teams by the corporate management group was the creation of an IT environment to facilitate management of change and provide a corporate information base to assist greater Departmental coordination.

The initial expectation of the Partnership Agreement was that Phase One would produce an Open Systems Reference Architecture specific to DEVETIR which would reflect the input of the BPM and SAP projects and provide the inputs necessary to specify future partnership projects aimed at "implementing the architecture" and developing a corporate database. As part of the process of defining the project plans for Phase One it became very clear that there were two issues which made such an expectation unrealistic. The resolution of these issues was a fundamental part of Phase One and had a large impact on the nature of Phase Two and beyond.

5.1 What is an Open IT Architecture?

The first issue to be resolved was the establishment of a common understanding of the nature of an OPEN IT Architecture. There are a great variety of definitions which focus on rules, guidelines and standards; however such definitions lose the essence of what is essentially an abstract concept.

An Architecture is a statement of behaviour and responsibilities when viewed across well defined interfaces. It tells designers and implementors the common purpose to which all parties work and the manner in which these parties must work together.

For instance in the case of a house, the architecture may define the boundaries of a room and the placement of the doors and windows so that they are consistent with the style of the rest of the building. It does not dictate the colour of the wallpaper in the room; this is the responsibility of its occupant. Furthermore, once the overall architecture is agreed, the builder of one room should be able to work independently of the builder of another room knowing that services delivered to it by other parties (eg electricity, water, etc) will be provided in a standard manner.

In the context of Information Technology, an "Open Architecture" would not result from simply mandating "open" standards for all elements of the information technology environment because there is no implicit structure or bounding of the areas of responsibility involved in this approach. Open standards are certainly advantageous in lowering the ongoing development, maintenance and support costs of IT systems through ease of communication and some degree of reusability; but, in themselves, they do not address the complex issues associated with achieving the business goals of an organisation.

An OPEN IT architecture is a defined set of business policies with regard to the use of information technology which in turn define domains of responsibility, information domains, interfaces between domains, engineering structures and, finally, current and projected implementation standards for IT components and methods.

The degree to which domains can be added, modified or removed with minimal impact on other domains is the key determining factor of the "openness" of an IT architecture. The degree to which the *engineering aspects* of one domain conform to those of another is simply a balance between the need for economies of scale provided by commonality and the need for specific functionality within a domain which, in some cases, may be achievable only with "non-standard" components.

In the view of the project, "open system" standards are a means of ensuring everybody is speaking the same language but architecture is a means of ensuring that everybody is talking in the same context, about the same subject and with an agreed understanding of their individual and collective responsibilities.

At an early stage it was clear therefore that OPEN*framework* Reference Architecture for DEVETIR would be a major undertaking and could not be achieved within the six months allotted. The Phase One OPAD plan therefore specified an output covering the whole spectrum of architectural design at a high level rather than of one area in great detail. Phase Two was allotted the task of filling in the detail.

5.2 What is the Linkage Between Business and Architecture?

The second issue was the methodology to be used by the Business Process Modelling Project.

It is widely accepted that the use of IT should be closely linked to the needs of its specific business owner; however in many organisations there is no strong link. ICL's OPEN*framework* is based on the results of the "Management in the Nineties" research program carried out by a consortium of 12 major US and European corporations and the Sloan School of Business

Management. One of the conclusions reached by this research is illustrated in Figure 2.

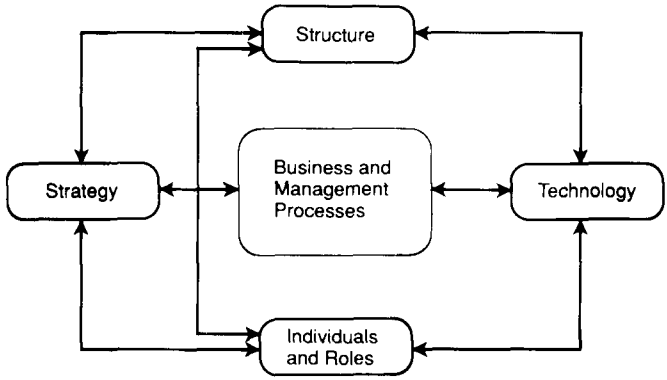


Fig. 2 MIT90s "Paradigm"

The MIT90s model illustrates that a business involves a complex set of interactions between different viewpoints, the central influences being business and management processes. No single viewpoint is likely to provide a sophisticated mechanism to guide the evolution of a business-focused IT environment.

In the view of the OPAD project, most traditional techniques of IT strategic analysis make a direct but insubstantial connection between business strategy and IT operational systems, resulting in a rapid divergence between the business and IT strategic plans.

The term *business process* in its simplest form means "a series of actions which produce a change". At DEVETIR there is a specific focus on *corporate processes* in which the coordinated actions of several Divisions are called upon to produce the best business outcome for a client. This is important, as it means that any IT systems will be developed with a cross-Divisional perspective, rather than on the basis of narrow operational needs of the Divisions.

The OPAD project believes that data- or information-centred analysis techniques are flawed because they tend to ignore the wider issues of large-scale corporate business processes. They also have an inbuilt assumption as to the technology – namely screen based on-line transaction processing (OLTP) – on which the business processes will be implemented. Such technologies may not be suitable for implementing long-running business processes in a widely dispersed and diversified organisation.

Initially the BPM project elected to use a DEVETIR in-house standard for business modelling. However a review of this methodology by the OPAD team found that, because of its focus on the development of operational

OLTP systems based on the current organisational structure, it was unsuitable when the organisational structure was not stable.

Therefore, prior to making any major decisions about the nature of the architecture, the OPAD project actively sought methodologies which: focused on business processes; had a means for identifying and focusing on core high value business processes; could accommodate solutions which were a mixture of technology and business changes; and derived the key business information objects from the process analysis. The outcome was to adopt methods for Business Process Redesign, based on the outcome of MIT90 and provided by ICL's IT Partners, and of Structured Analysis and Design Techniques (SADT) supported by the PC based Design/IDEF tool for process analysis and documentation.

The process of moving from the initial modelling methodology to the approach eventually adopted by the project highlighted the fact that IT people, while keen to recommend radical change to business people, are often very inflexible with regard to changes in their own environment.

The resolution of these two issues during Phase One is largely reflected in the current position of the project as described below. The key outcomes, however, were a commitment to an education program across a broad spectrum of the organisation, the addition of a prototyping and demonstration project to assist in the education process and to validate the architectural approach, the appointment of a DEVETIR business person as the manager of a revamped BPM project and the adoption of a Business Process Redesign approach to linking business and IT strategies and plans.

6 Current Position

At the time of writing (11/92) the project is operating to a plan covering the period up to the end of 1993. The major components of that plan reflect the OPEN*framework* methodology as tailored for DEVETIR by the project team (see Figure 3).

7 Business Focus

Every business has a "style", deriving from many attributes of the business, such as geographical dispersion, public accountability and posture (i.e. whether aggressive or conservative). An architecture must reflect the style of the business and therefore architectural design must start by identifying the focus of the business. Having determined the primary focus of the business, its associated high value processes must be identified.

The selection of this primary focus and the associated high value processes does not necessarily *dictate* the shape of the architecture. However it enables the need to invest in the architecture to be demonstrated by the business benefits that will accrue from that investment.

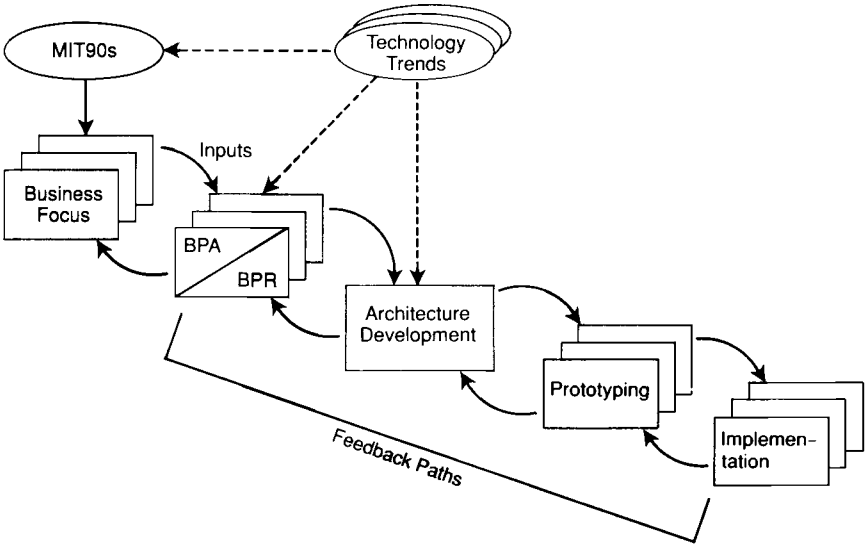


Fig. 3 OPEN*framework* Methodology
(BPA = Business Process Analysis, BPR: Business Process Redesign)

At DEVETIR, the business focus is simple: workplaces and the people at those workplaces. On the basis of specific needs of its clients DEVETIR wishes to focus its human resources on the establishment of safe, appropriately skilled and harmonious workplaces in Queensland.

The fundamental business focus changes only slowly with time but the accuracy with which the focus can be articulated, the level of commitment to it within the organisation and the implications of adopting that focus all change quickly as a project of this nature proceeds. Without strong management support an undertaking of this sort can easily result in proposals which fail to be implemented. Thus the constant involvement of senior management is absolutely vital.

8 Business Process Analysis/Redesign

Business process redesign, also known as business process re-engineering, is the methodology used at DEVETIR to reshape businesses processes in order to improve client service through cross-divisional coordination and to eliminate inefficient duplication of effort within the Department.

A business model which is totally independent of the organisational structure significantly enhances the potential to focus on the Departmental business processes rather than the specific activities undertaken by the individual divisions. Figure 4 illustrates the DEVETIR business model, known as the "neutral model", developed during Phase One of the project.

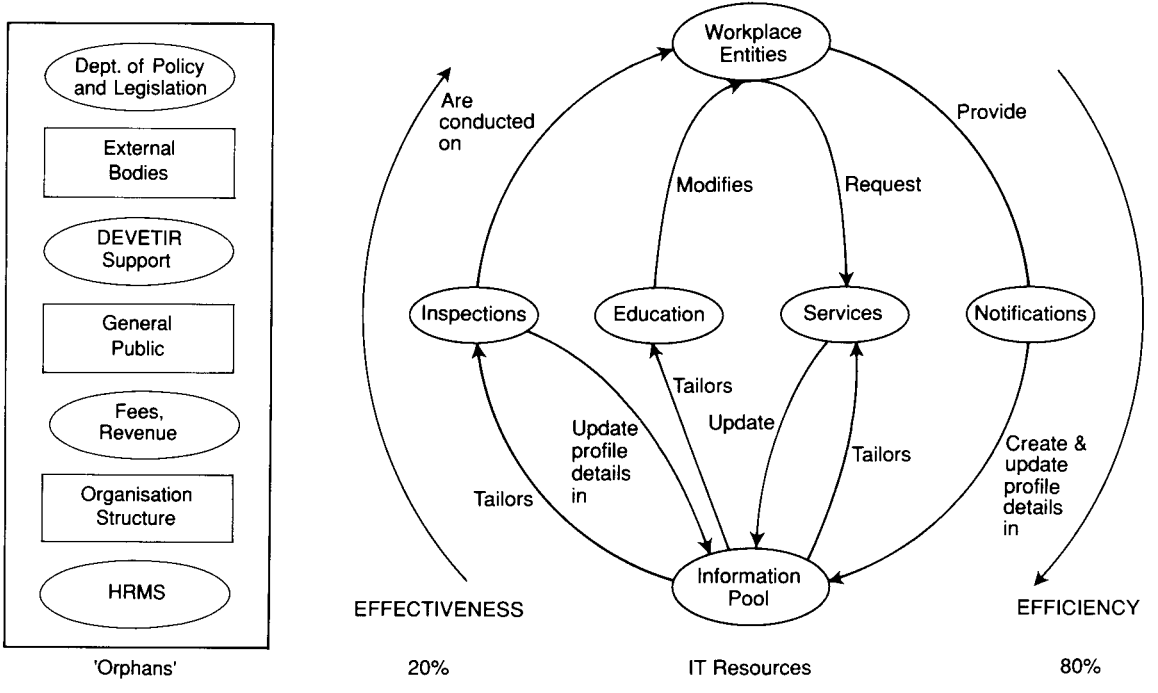


Fig. 4 The Neutral Model

This model was initially developed to illustrate the need to remove organisational units and structures from the original process models as this was making it impossible to identify core cross-divisional business processes. The model is "neutral to the organisation". The top level shows the business focus, the middle level shows the high value process classes and the bottom depicts a common information pool which supports the interworking of these processes.

"Orphans", the term for the items at the bottom left of Figure 4, are those things which are not covered by the focus of the model.

This model may appear simplistic; this is in fact its great strength. It illustrates in a simple manner the business focus, the high value processes, the role of IT support for those processes (i.e., efficiency on one side and effectiveness on the other) and, equally importantly, what is not being addressed.

Currently, 80% of the investment at DEVETIR in IT supports the reactive administrative processes (i.e., the right hand side) and only 20% of it supports the proactive activities aimed at reducing the impact of workplace-related injuries. This partly reflects the fact that computers have always been better at processing records than they have been at supporting a mobile, knowledgeable workforce. *This is changing.*

The analysis and redesign of business processes (shown in Figure 5) is a process in its own right. The business focus is used to identify the high value processes which are then captured in their current state as input to the redesign exercise. "High-value" does not simply mean that a process generates a large revenue stream. A high-value process is one which contributes significantly to the chosen business focus.

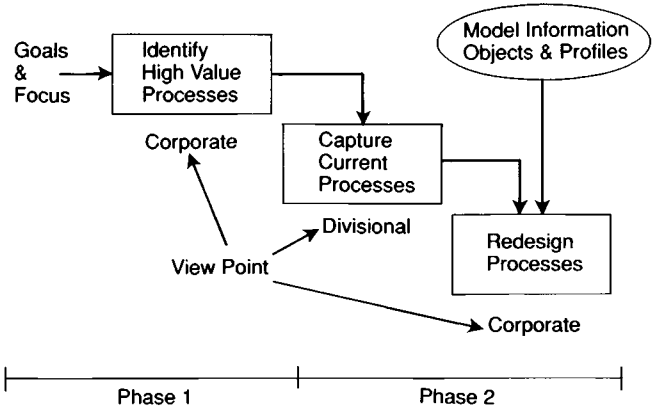


Fig. 5 Business Process Redesign

For instance a process collecting information which enables the organisation to target its resources on specific needs of clients is a high-value process even though it does not generate revenue directly.

In carrying out this process it is important to consider the viewpoint from which each step is undertaken. The architecture has a corporate viewpoint so it is important to identify all business processes associated with the target focus, regardless of where they are carried out in the organisation as a whole.

Current processes are captured in terms of their inputs, outcomes, process steps and resources in order that they may be compared with the redesigned processes in terms of business effectiveness and cost effectiveness. In order to *capture* current processes within an organisation structured on Divisional lines it is necessary to take a *Divisional perspective.*

It is absolutely essential that the *redesign* of processes is undertaken with a *Corporate perspective.* If this were not the case too many assumptions would be made on behalf of other parties in the process and opportunities would be lost as a result of these assumptions.

In terms of information modelling, in line with OPEN*framework* recommendations, the project has adopted an object-oriented approach to analysing information created and manipulated by the target business processes. The choice of an object-oriented analysis approach does not dictate an object-oriented implementation although it will obviously assist if, in the future, DEVETIR moves that way.

Object-oriented analysis, in the project's view, is an extremely valuable tool when the target architecture is highly modular and distributed. This is because it encourages an encapsulation of information and procedures, with defined interfaces for invoking those procedures. This makes it very much easier to achieve a high level of transparency, or independence, between distributed components.

Information objects at DEVETIR are defined in terms of whether they are corporate or local. Corporate information objects are maintained centrally and their integrity is guaranteed no matter where they may be located in the distributed computing environment. The management of local information and its integrity is not guaranteed to users outside of the local application.

9 Architectural Design

One of the significant conclusions of the MIT90s research programme is summarised in Figure 6.

The current IT position at DEVETIR is best described as "localised exploitation". Clearly if DEVETIR Management wish to focus on the incremental

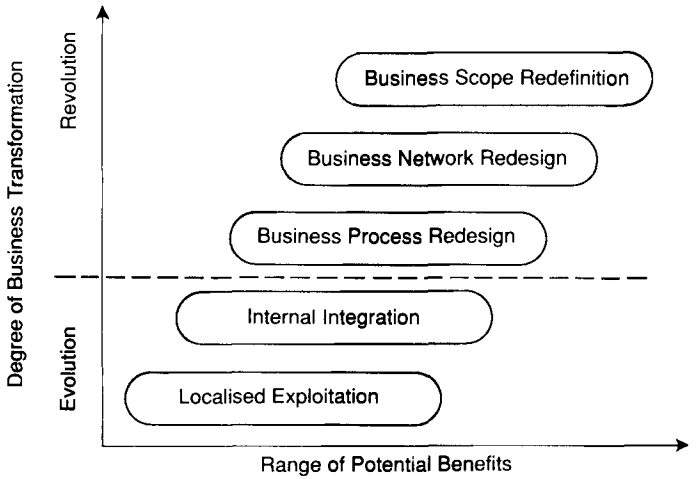


Fig. 6 The Role of IT (MIT90s)

redesign of processes, then the architectural design needs to be at least one step ahead. The target of the architectural design is "business network redesign".

Many businesses are looking at technologies such as EDI (Electronic Data Interchange) and EMAIL (Electronic Mail) as a means for developing loosely-coupled trading relationships with their suppliers and customers. At DEVETIR the architecture is similarly designed to support a loose internal federation of business units, within which business information objects common to Divisions are controlled centrally. A major objective of this approach is to push the boundary of this federation out into the business processes of DEVETIR's major clients.

The development of an architecture requires the support of IT staff as well as business management. In the view of the project this can best be gained by early publication of a set of *architecture design principles.* These principles must be directly linked to business policies. For instance two business policies of DEVETIR related to information ownership and divisional autonomy, lead to design principles concerning central control of corporate information objects in conjunction with the development of distributed applications.

The DEVETIR architecture is based on the following principles.

- use of technology appropriate to the support of end-to-end business processes in the form of long-running "cases".

- definition of interfaces between systems in terms of true business objects such as workplaces, employers, etc, and the actions which can be carried out on them.
- modular, open and distributed technical designs in which functional and information domains are clearly bounded and interwork with the minimum possible knowledge of the location, structure or construction of other domains.
- use of electronic directory systems to provide a single repository for maintenance of corporate information objects.
- an electronic messaging backbone designed to provide wide-scale sharing of complex information.
- management information derived from the support of core business processes and not by extracts from operational systems.
- adoption of a common office automation environment so as to achieve a single, consistent user interface to all DEVETIR services.

These principles are aimed at describing functional IT blocks (known at DEVETIR as Functional Modules), in terms of their defined behaviour through agreed high-level interfaces. By this means the architectural design is directly linked to the DEVETIR IT policy which states that IT systems will be designed to support a *high potential for change.*

The architecture was originally defined in terms of discrete functional modules interfacing to a "business process network". The latter contained a directory system (X500) to manage the central control of corporate information objects, a process support system (ProcessWise) providing overall coordination of core business processes and a message handling system (X400) for coordinating transfers of electronic message between functional modules.

The original architectural definition, published in June 1992, was a significant step in moving towards a highly distributed applications architecture. However, in more recent analysis and design work, adoption of more formal object-oriented approaches and feedback from the prototyping project has resulted in a move towards distributing the intelligence originally defined for the Business Process Network out into functional modules, some of which are specialisations aimed at the control of corporate information objects.

This has resulted in the distributed architectural components now being more in line with the emerging standard for Open Distributed Processing (ODP-X900) and is reflected in diagrams (Figures 7 and 8) below. It is also a result of taking a more formal stance on treating functional modules as objects in their own right.

A Functional Module supports a well-bounded area of either applications functionality (client) or information storage (server). At DEVETIR a variety of types of functional module are already defined; however, regardless of the functionality supported, all Functional Modules have some common com-

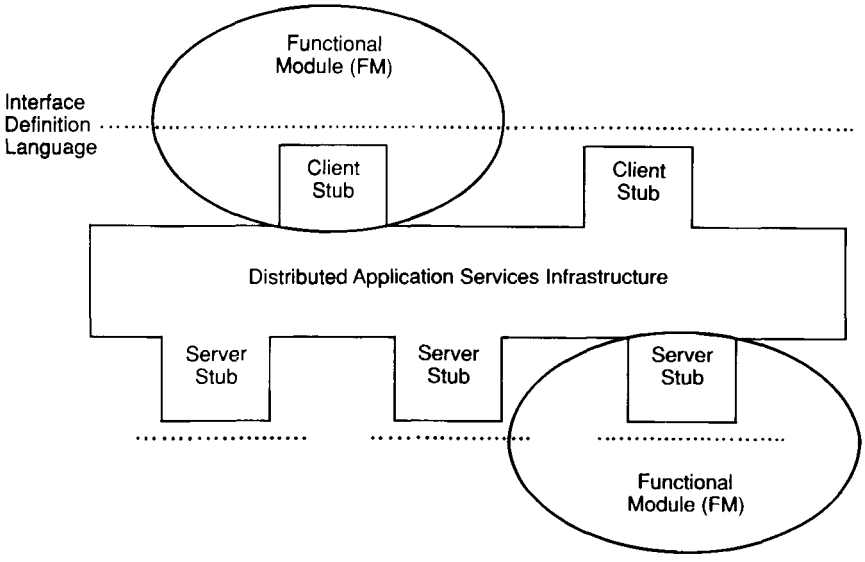


Fig. 7 Architectural Framework

ponents connecting them to the Distributed Application Services Infrastructure.

The key point about this approach, as illustrated in Figure 8, is that the business applications programmer is isolated from any knowledge of the underlying technologies or of the location and structure of other functional modules by the distributed application services layer. This layer manages all the interactions between modules in a manner transparent to the programmer.

The current description of the architecture as reflected by Figures 7 and 8 may appear to be a considerable softening of the original emphasis on electronic messaging as the communications backbone. Nonetheless messaging is the preferred technique and other techniques, such as TP, can only be employed if an extremely good business case can be made.

10 Prototyping

As might be expected it is difficult to convince business management that large amounts of money should be invested in development of architecture. Despite a great deal of support from the Corporate Management Group within DEVETIR, this has been a continuing issue; along with the need to educate IT staff in new technologies, it has led to the introduction of a Prototyping Project. This provides the "touch and feel" which is vital when translating abstract concepts into real deliverables.

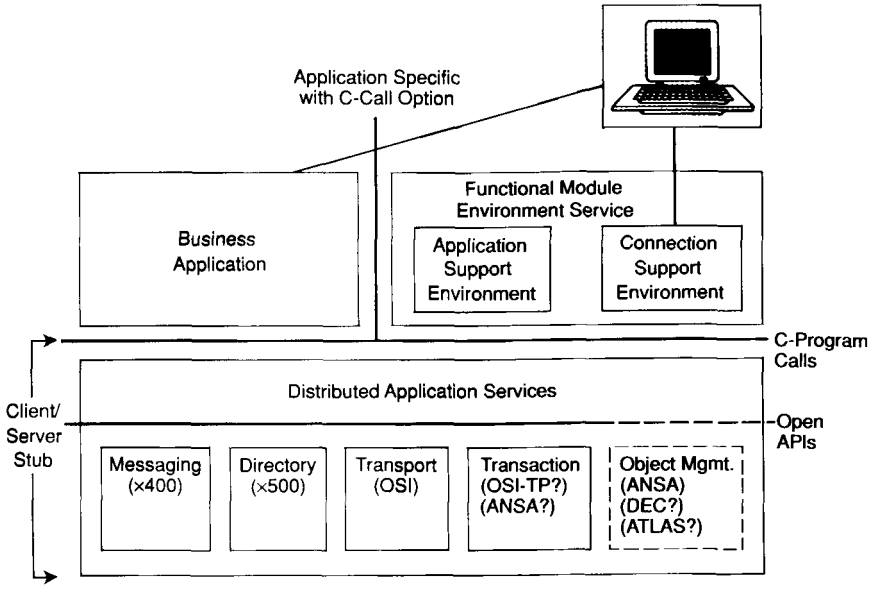


Fig. 8 A Functional Module

An OPEN Architecture is a constantly evolving entity. Feedback is constantly required to ensure it keeps up with real-world experience and with changes in the outside environment. Prototyping is a low-risk way to improve the architecture and systems developed within it prior to exposing them to the icy wind of the real world.

To date the prototyping project has implemented X400 messaging systems, X500 distributed directories, ProcessWise Integrator (an ICL process support system), ProcessWise Workbench (an object oriented process analysis tool), imaging and workflow management systems. Integration has allowed applications implemented in ProcessWise to exchange X400 messages with the office automation environment.

11 Implementation

The use of a common technical infrastructure means that business processes can be implemented incrementally on reusable technology which can be recovered should the project fail. It also allows the hardware and software to be installed and tested and the user made familiar with the basic facilities prior to implementing the target business processes. Small scale incremental implementation gives a rapid feedback on the business benefits and thereby allows the organisation to judge its future rate of investment based on tangible results.

For all of the above reasons the current phase of the Partnership Project, Phase Two, has been renegotiated and extends until the end of 1993 to incorporate a pilot implementation of a regional workplace management system based on the DEVETIR OPENframework reference architecture.

12 Lessons Learned to Date

The path by which the current position of the project has been reached has not been an easy one and there are some basic lessons to be derived:

- education is absolutely vital for both business and IT staff. It is also very expensive on key project resources and this aspect was significantly underestimated in the original project plans. Actions to address this problem are now in place; however, there is still a long way to go before wide-ranging support can be achieved throughout the organisation. Architecture projects are doomed to failure if they are seen simply as another output from the corporate ivory tower.
- it takes time for an organisation to adopt a common focus and culture, longer than it takes to define an architecture. Both need to be carried out in harmony.
- change brings risk. It is better to manage risk than it is to attempt to prevent change as a way of preventing risk. A positive approach to the management of change brings with it the need to manage risk.
- architecture is a business issue. The DEVETIR project has a great deal of technical skill, despite which the amount of technical debate has dropped significantly in favour of business understanding. Technology is unavoidable, but it should be used within a business context.
- finally, though business is paramount, IT is capable of facilitating fundamental change in a business and in its relationships with external parties. A great many negative views have recently been expressed in the media about the infinite capacity for IT to squander financial resources. On the other hand this must be tempered with the recognition that IT is extremely pervasive and has already altered many facets of modern life forever.

13 Summary

The objective of this article was to provide an insight into experiences gained at the coal face as a result of embarking on an architectural design project based on ICL's OPEN*framework* methodology. OPEN*framework* encourages adoption of most of the key concepts embodied in the DEVETIR architecture. It is hoped that the reader can understand why OPEN*framework* as an approach to adopting Open Systems achieves far more than defining a "buy list" of standards aimed at reducing costs. Business policy must be the driving force directing the continual evolution of an IT architecture which can adapt quickly to changes in the business and information technology environment.

Biography

Ian Craig is a senior business consultant with Fujitsu Australia Limited. The greater part of his 20 years in the computer industry has been spent with ICL in New Zealand, Australia and the UK. His work has covered a wide range from hardware and software development through customer implementation projects to business consultancy.

Strategic Information Systems Planning: A Process to Integrate IT and Business Strategies

R. Thurlby
Visiting Fellow, Brunel University

Abstract

Information Systems Planning techniques have historically been limited by their dependence on a given Business Strategy. Recent work from the Management in the Nineties Programme, described in the paper, has shown that IT is now a sufficiently powerful driver to have become interdependent with Business Strategy; consequently new IS Planning techniques have to be developed which support modelling of interdependent strategies.

This paper describes a new IS Planning methodology which enables interdependence to be analysed and dynamically modelled as a temporal process. The methodology employs the techniques of Strategic Alignment and Value Process Modelling, and these, together with their underlying theory, are described. In addition there is an examination of how analysis needs to be expanded beyond the organisation boundary using the concept of Process Invasion.

A case study is presented in the paper which describes the use of this IS Planning Methodology at Regional Electricity Companies in the UK. The case study focuses on how the methodology has been applied to develop and align Business and IS Strategies which respond to the opportunities presented by privatisation of the UK Electricity Industry.

The paper concludes by highlighting some issues raised by use of the techniques and how the issues will be addressed by research currently being undertaken by the author.

1 Introduction

The application and use of Information Technology, IT, by organisations has increased by orders of magnitude over the last 30 years. Introduced

originally to automate clerically intensive activities, it has now reached a position where most organisations could not operate without it. The more progressive organisations are indeed dependant on IT for continued success in their market place.

However despite this increasingly pervasive use of IT; planning for, and investment in, IT often remains an arbitrary process, isolated from business planning and independent of the business strategy. This situation remains in spite of the evolution of techniques during the last 10 years which enable an IT Strategy to be integrated with the Business Strategy. Initially these techniques derived IT Strategies which were reactive to a given Business Strategy; but recent research shows that IT can be a driver of Business Strategy, and methodologies are now being developed which recognise this fact.

This paper examines the development of business processes for Information Systems Planning, and discusses in detail the new techniques available to enable development of a Strategic Information Systems Planning Process which regards IT as a driver of Business Strategies. To demonstrate the use of this process, a case study of the application of Strategic Information Systems Planning in Regional Electric Companies is described.

2 The Development of Information Systems Planning

2.1 Early Approaches

As use of Information Technology moved from data processing towards information systems during the 1970's, methods were developed to enable the analysis of the information need of the end-user. Early analysis techniques tended to be applied to a single application supporting a limited set of business processes. The methodology known generically as Data Analysis guides the analyst systematically through a process which seeks to identify the functions needed to be supported by the system and the data required by those functions. (Rock-Evans, 1981). Recognition that data was often common to a number of applications led to enhancements which resulted in the Strategic Data Model (Gane and Sarson, 1979). This was the first attempt to understand the structure and interrelationships of the total data requirements of an organisation.

At this time the objectives of investment in IT began to change. Managers ceased to be solely concerned with using IT to improve operational efficiency. IT could now be used to provide information which would start to increase their business effectiveness. To determine the necessary information, however, required techniques which, as well as building data and functional models, also defined what the organisation was trying to achieve and linked these definitions with the data and functional models. Business Systems Planning (IBM 1984) was a widely used technique for this type of analysis.

Recognition and understanding of what was important for a business, if it was to achieve its objectives, had become a major issue for Information Systems designers. Without this knowledge they could not be sure that their systems were relevant to business needs.

Development of the concept of Critical Success Factors (Rockhart and Crescenzi, 1984) provided the technique by which Information Systems could be specified that gave direct support to Business Objectives, and thereby improved management effectiveness. Critical Success Factors (CSF) were those processes which had to be done absolutely correctly if an organisation's objectives were to be achieved. Furthermore each CSF could be quantified by a set of measures which determined whether it was being done correctly. Definition and analysis of the CSF measures provided the data from which the Information Systems specifications could be defined.

Together with the earlier planning and analysis techniques, an Information Systems portfolio could now be defined which was linked to the organisations business strategy. This portfolio was frequently analyzed into the standard 4 box matrix developed by the Boston Consulting Group and modified for Information System classification (McFarlan, 1984). The matrix is shown in Figure 1 below.

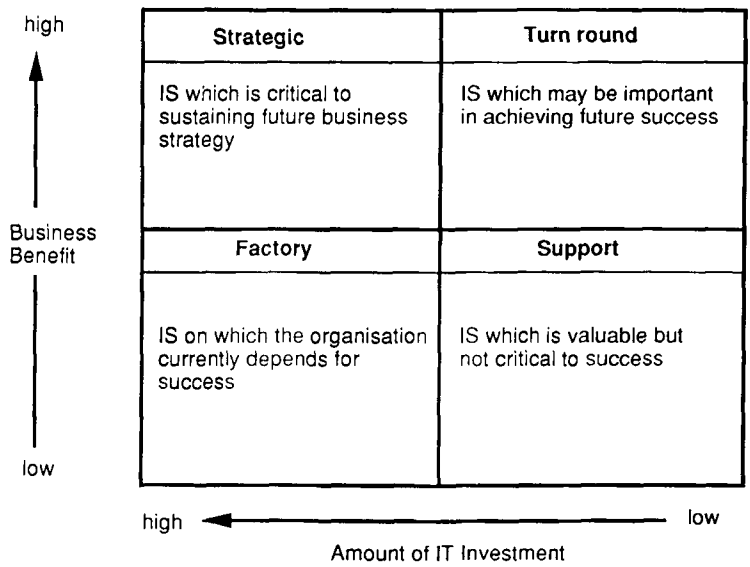


Fig. 1 Information Systems Portfolio

2.3 *Information Systems and Competitive Advantage*

The ability to create a link between business strategies and information systems led to another incremental step in the process of Information Systems Planning. This was to analyze where competitive advantage could be obtained from investment in IT. Although CSFs gave some information, it was not until the concept of Value Chains (Porter, 1985) started to be applied to IT investment that this step could be considered to have happened. Value Chains enabled an organisation to establish where most cost was incurred in building and delivering products and services. The high cost processes were then targeted for IT support to improve their efficiency and drive down costs.

2.4 *Limitations of IS Planning*

IS Planning techniques had now evolved into a sequential process, which, starting from a given set of Business Objectives, could develop an Information Systems Strategy. The process is shown in Figure 2 below.

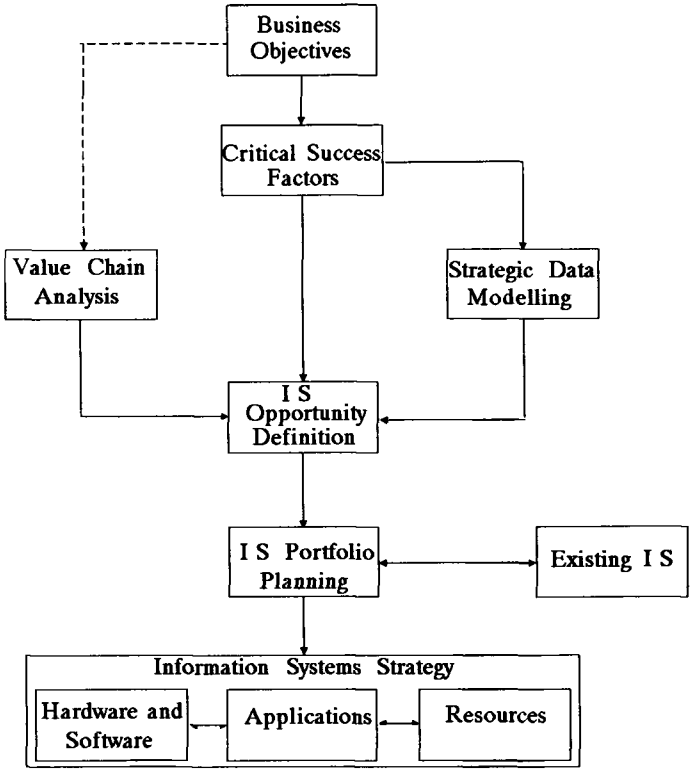


Fig. 2 IS Planning

There were 2 limitations to this approach. First, it took the set of Business Objectives as given and developed an Information Systems Strategy which was reactive to them. Second it was a once-off process and did not easily permit interaction and reiteration. Consequently many of the IS Strategies produced tended to be accurate at the time when they were done, but become increasingly inaccurate and out of step with the business as time progressed. It took the insights of the Management in the 1990s Programme (MIT90s) to address this issue.

3 Strategic Information Systems Planning

3.1 Management in the 1990's

The Management in the 1990s (MIT90s) programme was a 5 year research programme run by the Sloan School of Management at the Massachussetts Institute of Technology. It was sponsored by 12 leading companies and Government Departments, ICL and BP being the two UK sponsors. At the time of the programme's inception in 1984 there were a number of issues facing organisations. These were encapsulated into 4 major concerns.

- Would turbulence in business continue to increase?
- What caused the turbulence?
- What role could IT play to help organisations respond to turbulence?
- Was IT itself a cause of turbulence?

The programme set out to address these concerns, and rapidly came to focus on the link between Business Strategy and Information Technology. How to manage the link became a major thread of the programme, and it was soon understood that unless an organisation's processes, structures, and people skills, were clearly understood, then Business Strategy and IT could not be integrated. The programmes found that failure of a number of Information Systems Plans could be attributed to not changing an organisation's processes, or structure, to exploit the capabilities of IT, and so gain competitive advantage and achieve business objectives.

MIT90s had identified a fundamental weakness in existing Information Systems Planning methods. To address that weakness required two things. First a method to study the role IT could play in bringing benefit to an organisation. Secondly devising methods which linked organisations' processes and structure with Business Strategy and IT. In the event MIT90's produced a number of very significant findings (Scott-Morton 1990) which answered the questions posed by the programme. Of great significance were the findings which concerned the future role of IT in the organisation.

These were in summary:

1. IT does not provide sustainable competitive advantage by itself. It requires integration with the organisations processes and structure to achieve lasting advantage.
2. IT capability is now of sufficient influence to become a driver to change the organisation, its processes, products and even its market.
3. Although IT is now an agent of transformation there are still significant technical problems associated with unlocking data held in an organisations information systems, and using it to Informate (Zuboff 1988) the workforce.

The findings of MIT90s provided a paradigm shift for IS Planning. IT capability was being proposed as a driver which could change Business Strategy. Of equal importance, an organisation's processes and structure were identified as the mechanism through which Business Strategy and IS Strategy could be properly integrated. Both of these ideas needed to be incorporated into any IS Planning Process. Further consideration of the ideas then derived the additional idea that IS Planning was no longer a sequential process, but an iterative process with a number of possible start points.

Out of these findings, and the evidence from the MIT90s research which supported them, two new analysis techniques were developed. These were Strategic Alignment and Value Processes, and potentially they provided the mechanism by which IS Planning can utilise the new paradigm presented by MIT90s.

3.2 Strategic Alignment

3.2.1 Strategic Alignment Theory. The Strategic Alignment Model (Figure 3) can be regarded as the solution framework for MIT90s. In addition

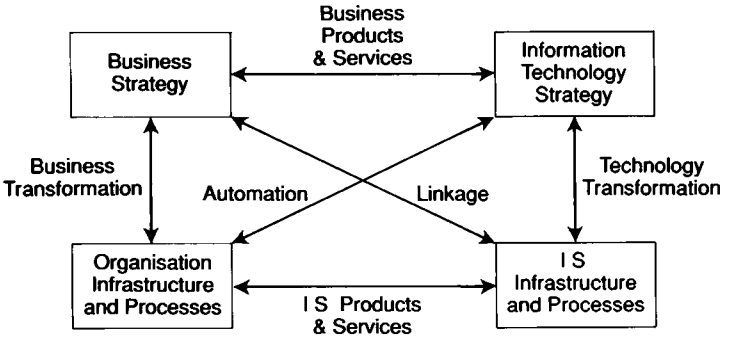


Fig. 3 Strategic Alignment Model

to its role in Strategic Information Systems Planning Strategic Alignment has a number of other purposes, for example in helping the diagnosis of organisational problems.

Using Strategic Alignment for Strategic Information Systems Planning is still a comparatively embryonic process, and has yet to be completely defined as a mechanistic technique. The case study of its application to Regional Electricity Companies (RECs) in Section 4 has elements of research as well as application. The results will not only provide a usable Strategic Information Systems Plan for RECs, but also provide research material to be used for developing the technique.

Currently Strategic Alignment is a three phase process.

Phase 1 Collect information to complete a definition of the content of the 4 elements of the model.
Phase 2 Analyze the contents of each element to ensure its consistency and compatibility with the rest of the element.
Phase 3 Analyze the content of each element to harmonise it with the content of the other 3 elements.

Phase 3 is normally tackled in 2 stages. The first stage is what is known as "a quick canter round the model" to pull out all the obvious linkages and relationships. The second stage is a detailed examination of all the linkages and relationships to create full alignment (Macdonald, 1991). Strategic Alignment is a complex process and can be very lengthy. It should not be undertaken lightly. Alignment is unlikely to be achieved after one detailed iteration. On subsequent iterations the analyst should be aware of the dangers of "paralysis-by-analysis". Nevertheless Strategic Alignment is an extremely powerful technique when used with thought and caution.

3.2.2 Strategic Alignment Practice. As stated in the previous section the initial step is to collect information which defines the contents of each element in the Strategic Alignment Model. Structured interviews and examination of published documents (eg annual reports) are a standard and effective method of proceeding. The information needed will vary from industry to industry and it is important to collect quantitative as well as qualitative data. Sets of objects relating to each element are listed in Figure 4.

In addition there are a number of objects which are common across a number of elements. These include: control mechanisms, triggers, paradigms and capabilities. It can be seen by examination of these objects that Strategic Alignment needs all the information required by earlier techniques of IS Planning. This is to be expected since the subject has not changed; it is the analysis of the subject which is fundamentally different.

Analyzing the information to produce alignment is the next step in the process. Although the detail of approaches and start points will vary, the following principles have been found to be of value:

Business Strategy	I T Strategy	Organisation Infrastructure and Process	I S Infrastructure and Processes
Objectives	Technology	Value Process	Entity (Object)
Competitors	Technology Objectives	Process Element	Information Systems
Markets	Technology Trend	Organisation Structure	Information Process
Legislations	Technology Competence	Organisation Element	Data Model
Regulations	Configuration	Resources (human)	Physical Database
Environmental Issues	Architecture	Skill	Service Deliverable
C S Fs	Standards	Role	I S Skill
Industry Process	Policy	Information Need	I S Organisational
Financial Data		Decision	Element
Competencies			

Fig. 4 Strategic Alignment Objects

1 The majority of drivers for change are found in the Business Strategy and IT Strategy elements so these are the logical places to start.
2 Initially it is useful to consider the strategy as a set of sequential objectives. In practice objectives tend to be pursued concurrently, but to place them in a logical sequence aids understanding and focus (Hay and Williamson 1991).
3 For an organisation to exist in a market-place, or enter that market-place, it needs to have in place a set of competencies before it can consider tackling its objectives. These need to be identified and described.
4 For that organisation to succeed in the market-place it has to achieve its objectives. To achieve each objective requires a set of capabilities to be delivered. Capabilities can be skills, value-processes and information systems.
5 Also impacting on objectives are external factors, such as market forces, competitive activity, regulation and technology. These have either a positive or negative impact on the objective, both in terms of the time needed to achieve that objective, and the benefit produced from it.

The approach described above can be described pictorially and is shown in Figure 5.

The description above gives an outline of an approach to the alignment process, the major complexities omitted being the concurrent nature of objective achievement, the second level interactions of the objects used to deliver capability, and the many-to-many relationship between objectives and capabilities.

The advantage of this method of tackling Strategic Alignment is that it provides a temporal axis which can form the basis of a dynamic simulation. Furthermore, by treating the process as sequential it can be broken down into a set of sub-alignments which makes the process manageable and understandable.

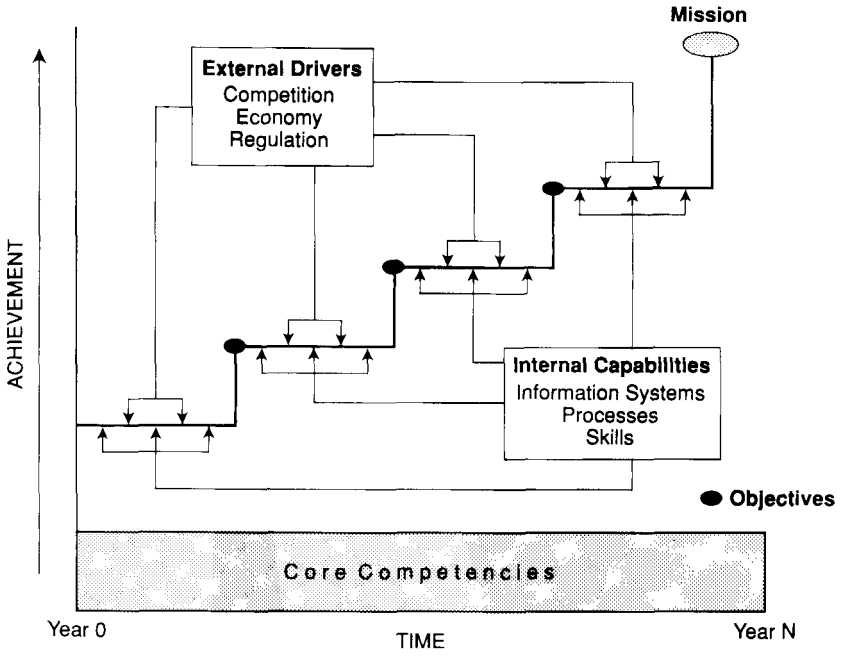


Fig. 5 Temporal Alignment

3.3 Value Process Model

A second technique to be developed in the MIT90s programme was that of Value Process Modelling. Although used on objects within the Organisation Infrastructure and Process element of the Strategic Alignment Model, Value Process Modelling is a technique in its own right. (Scott-Morton, 1990 {2}).

Value Process Modelling moves forward from the Value Chain concept (Porter, 1985) and changes the focus from one of cost to one of added value. Whereas Value Chains sought to identify where cost occurred and accumulate the cost into a set of discrete categories; Value Processes are concerned with logic, flow and interaction. They seek to identify the sets of processes, which when accumulated together, provide an added value to the organisation which is greater then the sum of the individual processes. An example of a Value Process is shown in Figure 9 below.

The second area where Value Process Models improves analysis capability, is that they are independent of an organisation's boundary, and permit definition of processes which integrate with the organisation's supplier and customer processes. In times where much competitive advantage is to be gained from inter-organisation collaboration, the ability to understand where other organisations' processes can be integrated with internal processes is

very important. The concept, known as Process Invasion, is shown in Figure 6 below.

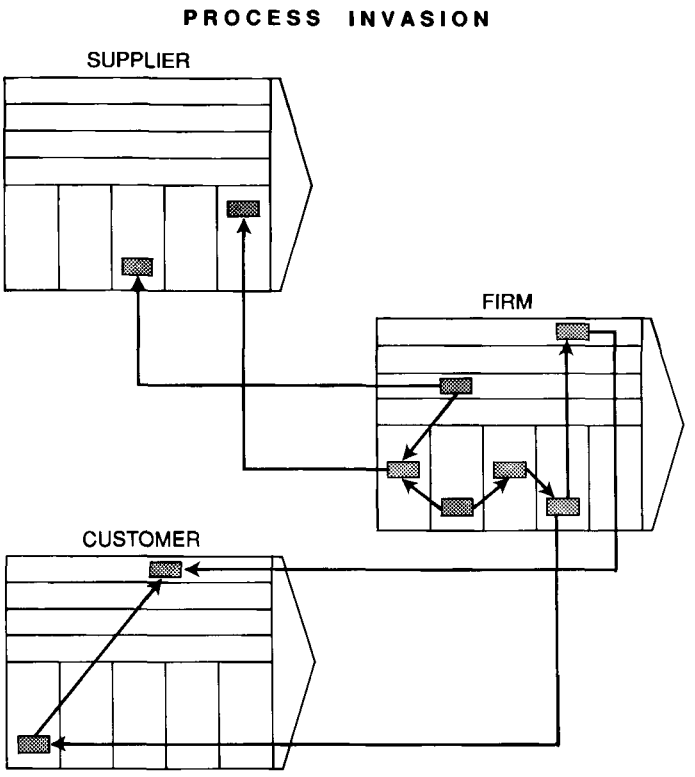


Fig. 6 Process Invasion

An interesting result to emerge from the study of Process Invasion is that the costs tend to move upstream to the supplier, and the benefits tend to move downstream. Just-in-Time inventory management is a good example of this, because the inventory costs are being unloaded on the supplier by the customer. Awareness of this has cost less thoughtful suppliers, who get into a Just-in-Time contract, a lot of money. Suppliers who think the problem through make sure that they can leverage sufficient other benefits, such as reduced production costs, through longer production runs, and larger sales volumes, through longer term contracts, to offset their increased inventory costs.

To develop a Value Process Model, use of a process identification and decomposition approach coupled with a Value Chain is appropriate. It is then subsequent analysis which develops the Value Processes. A top down approach is recommended and once the key processes have been identified, defined, and quantified, they can be examined to identify commonality

linkages and sequences. In many cases a flow chart format is a good way of analysing the process; but for discussion and presentation purposes its superimposition on the Value Chain template has been found to be valuable. Commonality can be found in a number of ways:

- support of one objective
- support of the same critical success factors
- response to a driver
- use of the same data

The fourth point is both valuable and dangerous. There are some sets of processes which do integrate logically through common use of data. Focus on data without consideration of the organisational and objective impacts will potentially lead to serious distortion of the model.

The final thing to consider is what exactly is required from the model. The danger is that the Value Processes are a reflection of current practice and status quo. The question has to be asked: "is this going to improve the effectiveness of the organisation by delivering capability to achieve objectives?"

For each Value Process, and each individual process in it, it is necessary to establish its necessity, its correct position in the value process and whether alternatives exist. This analysis also requires looking outside the organisation to establish invasion opportunities.

4 Strategic Alignment in Regional Electricity Companies

4.1 Introduction

The Electricity Industry has undergone many changes because of privatisation. One result has been to cause a fundamental re-think about the use of IT. Since many of the other changes involved their organisation structures, and the related processes, it would have been unwise just to consider IT in isolation. Through contacts with ICL a number of the Regional Electricity Companies (RECs) were aware of MIT90s. The studies were undertaken with the objective of producing a Strategic Alignment Model. An important part of this objective was to use Strategic Alignment to identify and evaluate potential changes to the strategies and processes which would improve the effectiveness of the business. Finally an outline Information Systems Development Plan was to be produced which was derived from the Strategic Alignment Model. Figure 7 below shows the route map for the study.

However before the results of the study are discussed, it is necessary to give a background to the Electricity Industry and especially the impact of privatisation.

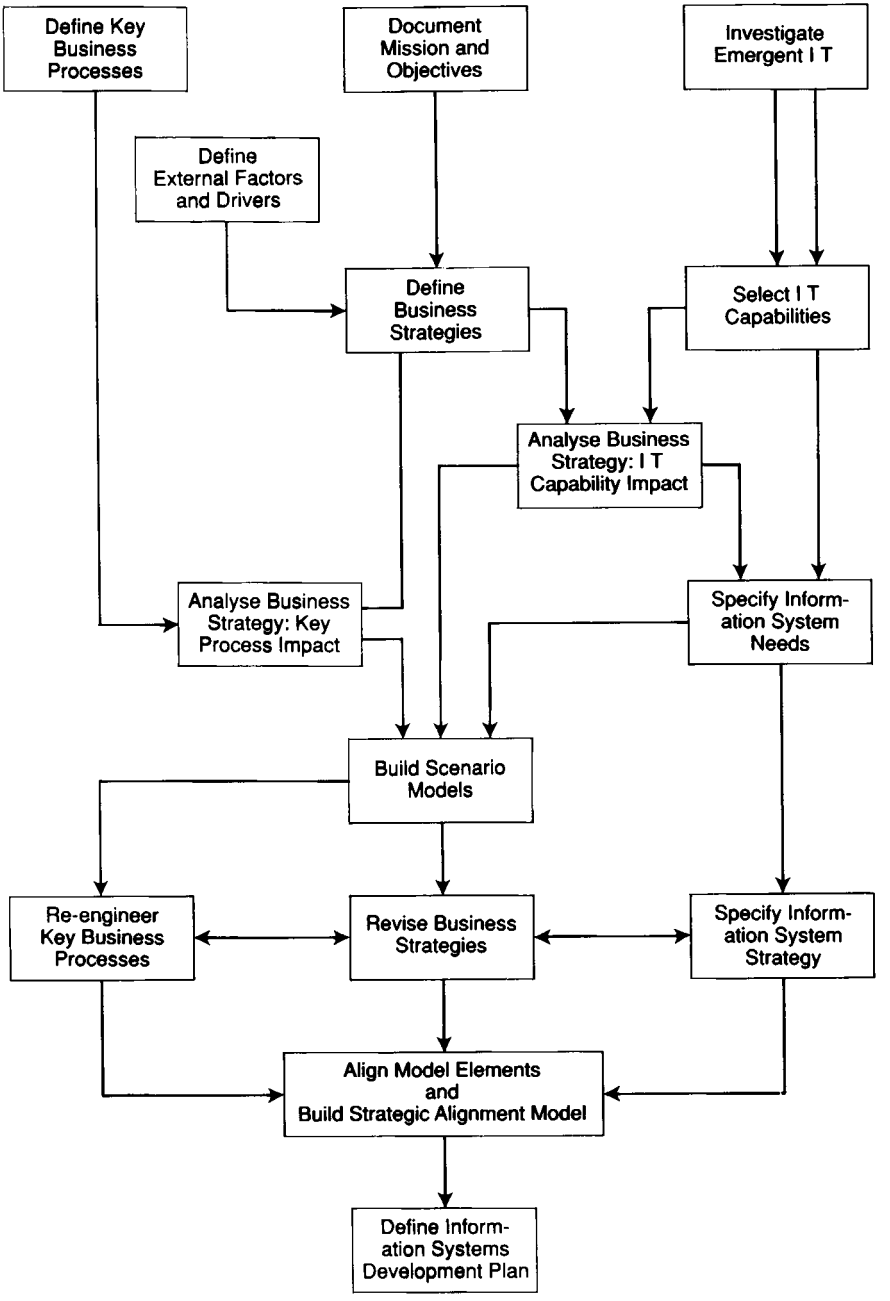


Fig. 7 Strategic Alignment Route Map

4.2 *The Impact of Privatisation*

The privatisation of the Electricity Supply Industry in the UK caused fundamental changes to the way organisations in the industry carried out their business operations. Whilst pre-privatisation Area Electricity Boards, responsible for distribution of electricity from the transmission network to the customer continued to have this responsibility, when they became Regional Electricity Companies (RECs). there were a number of fundamental changes underneath the veneer of a public monopoly becoming a private monopoly. The principal change on the Supply Side was the introduction of real competition to generate electricity. For RECs the result was that they now had to negotiate their own contracts with Generation Companies and also were able to generate their own electricity. A further requirement was that in a REC the Supply Business (buying and selling electricity) had to be kept financially separate from the Distribution Business, (managing the network).

Distribution was also subjected to change, primarily through being regulated on price and service, but also by having competition introduced to its non-regulated operations, which was work to maintain, refurbish and reinforce the network, including building extensions to the network which provided supplies to new customers. The Distribution Businesses have the bulk of RECs employees within their activities and were faced with threats on three sides. Their revenue was regulated through the RPI-X formula and Use of System Charges. They had to operate within defined standards of service, and the non-regulated areas of their business, where they had freedom, were being opened to competition.

4.3 *Regulation*

Regulation, as can be seen from the previous section, emerged as a major driver of the business strategy of RECs. It is therefore worth looking at the exact nature of this driver.

In recognising that RECs were natural monopolies, competition had to be introduced through generation and supply. However to prevent RECs from exploiting their position, the Distribution Business had to be regulated, which was one of the roles of the Office of Electricity Regulation (OFFER). OFFER introduced the RPI−X formula to regulate the amount by which RECs could increase their tariffs. RPI is the inflation rate, and X is a REC specific value, negotiated with the Regulator, based, among other things, on the cost of running the network and maintaining it. Since this cost falls within the orbit of Distribution activity companies are concerned to negotiate a realistic X-factor.

The second regulation area is standards of service. These are measures of how a customer is treated by its REC. It includes such measures as the number of minutes off-supply per customer each year, keeping scheduled

appointments, and the time taken to restore customers after a fault, plus technical measures about, for example, preventing voltage fluctuation outside pre-defined limits. Again many of these measures fall directly within the responsibility of the Distribution activities.

Finally there is the use-of-system charge. An electricity bill consists of two elements, a charge for the energy, and a charge for using the distribution network. This is a sum added to each unit of electricity sold and its value is controlled by the Regulator. Use-of-System is of prime importance since it is from this that 90% of their revenue accrues. To reflect further the importance of use-of-system, the RECs make very little profit from the buying and selling of electricity. Most of their profits come from use-of-system. Therefore to maximise profit means driving down the cost of operating the network. This impacts the Distribution Business directly.

4.4 Results of the Study

4.4.1 Introduction. The results produced by the Strategic Alignment Studies are described in the following sections. They cover the following areas:

- objectives of the Distribution Business.
- drivers impacting achievement of the objectives.
- competencies and capabilities needed to respond to the drivers
- information Technology and Information Systems needed
- value processes
- alignment

The volume of information collected in the studies is such that a complete description of all the results is precluded. Consequently a subset of the results is presented, but in such a way which demonstrates their interrelationships and how they align.

4.4.2 Pre-Privatisation Objectives. Prior to being privatised the objectives of the Engineering Department (the precursor of the Distribution Business) were as listed below.

- to operate the network in the most safe and secure way possible
- to maintain a continuous supply to all consumers
- to achieve the two previous objectives within an agreed budget.

The objectives were always quantified by a number of measures, which usually demanded incremental improvements on previous years' performance.

4.4.3 Post-Privatisation Objectives. There has been a fundamental shift in the objectives since privatisation. Although the objectives concerning safety and continuous supply remain (this is to be expected because they

cover areas which are subject to regulated standards) new, and fundamentally different ones, have been identified by the study:

- to improve overall levels of customer satisfaction and service at a rate better than set by the Regulator.
- to reduce the costs of the Distribution Business in real terms year on year.
- to maximise use-of-system charge revenues in order to assist the group in achieving its profit targets.
- to seek and create incremental revenue streams by fully exploiting skills and capabilities.
- to seek to influence the Regulator to ensure that any changes in regulation are favourable.

As can be seen the shift has caused a refocus away from engineering and cost management to customer service and profit.

4.4.4 Drivers. The research concentrated part of the information collection on what the senior managers saw as the major drivers that affected day-to-day and long term activities. There was a high degree of commonality in the results which are listed below:

- the need to respond to competition which is eroding their market.
- the need to drive down costs
- responding to, and managing, the Regulator's requirements.
- the need to change the culture from an engineering driven culture to a customer and profit culture.
- the need to exploit IT as an agent of change.

Each of the drivers was then further analyzed to pull out the detailed issues. These produce a set of competencies and capabilities as discussed in section 3.3.2. The competencies and capabilities associated with the driver "Need to Drive Down Costs" were found to be:

- improve productivity of industrial staff.
- deskill procedures and practices.
- multiskill industrial staff.
- pay staff at the market rate for the job.
- reduce system losses by improved network planning and control.
- develop new maintenance regimes based on usage rather than time.
- develop distant and remote control processes.
- plan work to eliminate peaks and troughs of activity
- introduce on-line network analysis capabilities to enable more accurate reinforcement and replacement planning.
- develop analysis capability to run the network to minimise ageing.
- investment risk appraisal capability.

The competencies and capabilities listed can be classified into a number of groups

Efficiency Improvers.
Process Re-engineering.
Organisational and Cultural.
IT Capability Dependant.
Information Availability Dependant.

Driver: Need to Drive Down Costs Classifications Competencies and Capabilities	Efficiency	Process Re-engineering	Organisational & Cultural	IT Capability Dependent	Information Dependent
Improve Staff Productivity	X		X		
De-skill Procedures and Practices	X		X		
Multi-skill staff	X		X		
Competitive Pay Rates			X		
Reduce System Losses	X	X		X	X
New Maintenance Regime		X			X
Distant/Remote Control		X		X	
Work Planning/Scheduling		X		X	X
Reinforcement Planning/Scheduling		X			X
Network Ageing Minimisation				X	X
Investment Risk Appraisal		X			X

Fig. 8 Driver Analysis Matrix

The analysis is shown in Figure 8.

Having identified the competences and capabilities required and classified them, analysis of their IT implications was the next step. These defined both the technology capabilities and information systems that would be required.

4.4.5 Information Technology. To develop each competence where it does not already exist, and more importantly, develop the capabilities implies an IT and Information dependence. Some are heavily dependent on IT and information, as can be seen from Figure 8 but others have less dependence.

Analysis of each associated competence and capability will determine the following;

- functional and process requirements.
- organisational impact and requirements.
- data needed to support the capability.
- specific IT required to deliver the capability.
- interrelationships with other capabilities and competences.

It was now possible to draw out the IT capabilities required and also build up the Information System's needs.

This activity was recursive in as much as the potential impact of emerging IT capability has to be considered, as well as exploitation of existing IT capability. In this study the IT capabilities needed to support the driver: "The need to drive down costs," were identified as being:

- expert system scheduling software, for work planning and network switching programmes.
- screen technology, capable of being run without mains power so Engineers could control parts of the network from vehicles at remote sites.
- wide band telecommunications, to support system control and management.
- intelligent meters and remote terminal units.
- Object Oriented data bases, to enable the dynamic nature of the network to be modelled so the topology and load data could be kept consistent.
- on-line network analysis software.
- demand forecasting software, based on neural network techniques.

Examination of the above list of capabilities shows a mixture of existing and emerging IT capabilities. It is also clear that the capabilities will have to be delivered as information systems if they are to have a positive impact on the objectives. For example use of neural network techniques in demand forecasting software should produce more accurate results than use of conventional algorithms. However, unless the database of network load histories is not available then the software cannot be used to support the processes.

4.4.6 Value Processes. From analysis of the capabilities and competences, and the information needed to support them, a number of value processes were identified. These were sets of individual processes which when integrated were found to improve effectiveness of the organisation. The research identified five value processes. Each improved effectiveness by increasing the level of control that the organisation could exert on its assets or costs. They were also identified by common use of a dataset by each process.

The five identified were:

Control of Energy.
Control of Resources.
Control of Money.
Control of Assets.
Control of Customers.

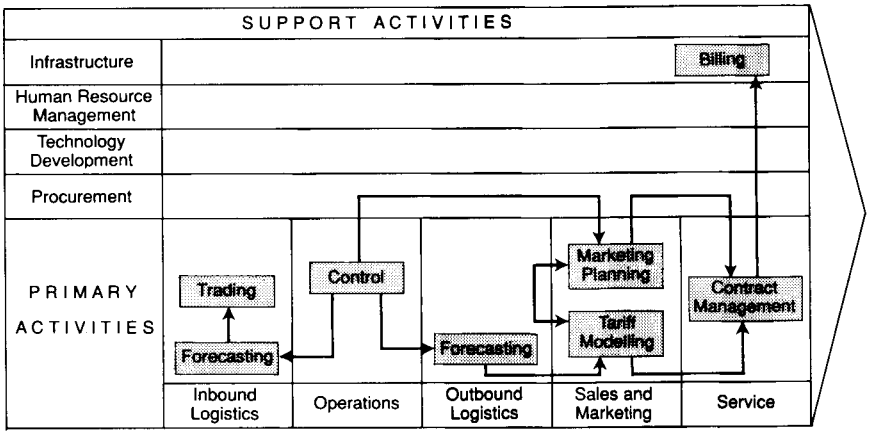


Fig. 9 Energy Control Value Process

Figure 9 shows the value process for Control of Energy as an example.

Each individual process uses data which defines the loads and flows of electricity in the network. They integrate to create the mechanism which controls electricity in a Distribution Business. Re-engineering these processes so they are co-operative and aligned, and provision of a common database with supporting information processing, will deliver capability which enables achievement of the objectives of the business (see Section 4.4.3). Further analysis revealed how the Control of Energy Process could possibly be extended into the value chains of a RECs suppliers and customers. This invasion is shown in Figure 10.

IT capability to implement control systems both upstream and downstream would provide significant benefit to a REC, through creation of an external energy management capability. The IT and communications technology exists to develop such a facility, but implementation, and resultant accrual of competitive advantage is dependant on negotiation of contracts which provide tangible benefits to all the players.

4.4.7 Completing Alignment. The Strategic Alignment process has now reached the point where objects in the 4 elements of the Strategic Alignment have been identified. By use of the concepts of competences and capabilities an initial degree of alignment has been achieved.

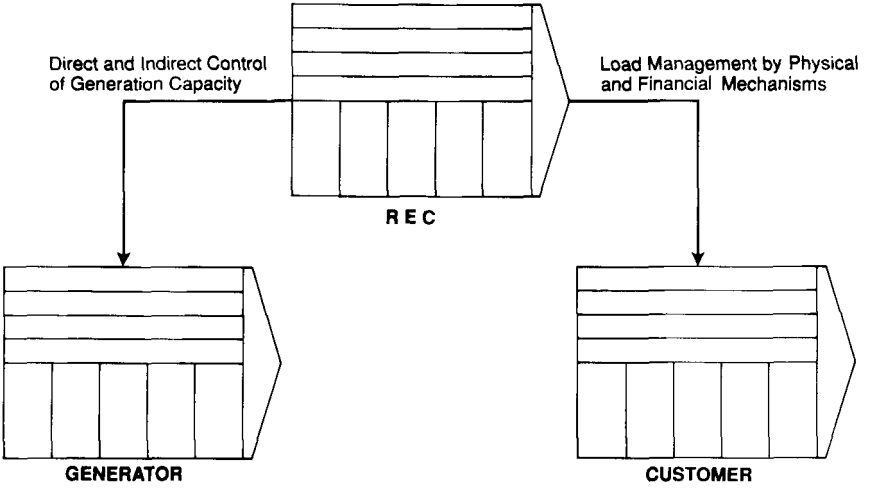


Fig. 10 Energy Process Invasion

The relationship between capabilities and IT and Information Systems has been defined, and the value processes identified. These objects were now mapped back to, and aligned with, the business objectives listed in Section 4.4.3. The purpose was two-fold. First to prioritise the objectives, and second to establish the impact of IT and value processes on these objectives. Of the five objectives, two emerged as the most important in terms of responding to the drivers. These were:

- to reduce the cost of the Distribution Business in real terms, and
- to improve overall levels of customer satisfaction and service.

For the first it was found that all five value processes identified had a major impact on the achievement of the objective whereas for the second only Control of Resources, Control of Energy, and Control of Customer had significant impact.

Moving round the model to look at the Information Systems, the need to have a set of systems relating to each value process was identified. These were then aligned back to the objectives to check applicability. From the Control of Energy process, the applications which impacted the objective to reduce costs of the Distribution Business in real terms were all concerned with effective distribution of electricity. The five most important applications were:

- load analysis (loading of feeders).
- switching schedule control.
- load forecasting.
- loss minimisation.
- fault and incident management.

As was suggested by the Value Process Analysis, data is an integrating element and data modelling of the five applications showed that they had a common data requirement. The common data was a record of the load of measurements at each node in the network. So to develop the above five applications requires a supporting database of load data to be developed. Relational Database technology would satisfy this need. But to provide the applications would benefit from emergent IT and three areas were identified:

- neutral networks, for load analysis and load forecasting.
- expert systems, for switching schedule control.
- low energy graphics screens, for fault and incident management.

This last technology would enable control of the network to be passed to remote sites during the restoration process.

So having identified the IT needed, the penultimate step was to relate it back to the Objectives to establish what its impact would be. In this example IT impacted the Objective in two ways. First its availability would reduce the time taken to achieve the Objective and second, analysis showed it could improve the level of achievement of the objective. That is, the benefits would be greater. A final alignment step reconsidered the Value Process and three further implications were discovered.

- skills in the control room staff would change in order to exploit the applications.
  - deskilling as regards Switching Schedule Control, due to automation.
  - reskilling as regards Load Analysis and Load Forecasting in order to improve effectiveness of decision making.
- control-room working procedures would require re-engineering to enable control to be passed to field-based staff.
- industrial relations issues would need to be resolved concerning the empowerment of field-based staff.

Thus the alignment process identified the process re-engineering and skills issues which would need to be addressed if full benefit is to be gained from implementation of the Information Systems. Use of the Strategic IS Planning Methodology in general and Strategic Alignment in particular, cannot be considered complete unless this work is done.

5 Conclusion

The research project completed an alignment process for each of the Objectives and then attempted to integrate the results. This was a complex process, not least due to the overlap and commonality of instances of the Objectives being analysed. The case study demonstrated clearly the wealth of results and richness of the Strategic Alignment Model through the examples discus-

sed. Strategic Alignment is emerging as a powerful but complex technique which, as can be seen from the case study, requires persistence and thoroughness on the part of the Strategic IS Planner.

One obvious problem is that it is not easy to check the validity of the results because they will not be known until the plan is implemented. By then of course the drivers may have changed.

To address this issue is the next stage of research. The case study has shown a Strategic Alignment Process will produce a valid model. This model needs to be automated by use of process simulation software, which will simulate the model and assess the result. A further benefit of such an approach is that an object instance, or metric, can be changed and the model re-run. This capability would then enable IS Planning to move from being a static process to a dynamic process. As was identified by MIT90's, business turbulence will continue to increase, therefore Strategic IS Planning must evolve into a dynamic process for it to be a valuable analysis toolset.

Acknowledgements

The author would like to thank Professor G Musgrave of Brunel University, and K Hugh Macdonald of ICL, for their support and guidance during his secondment and in the preparation of this paper.

References

ROCK-EVANS, R. *Data Analysis*, IPC Business Press 1981.

GANE, C. & SARSON, T. *Structured Systems Analysis: Tools and Techniques*, Prentice Hall inc 1979.

IBM CORPORATION. *Business Systems Planning: Information Systems Planning Guide, Application Manual*, GE20-0527-4, IBM Corporation, 1984.

ROCKHART, J. and CRESCENZI, A. Engaging Top Management in Information Technology, *Sloan Management Review*, Summer 1984.

MCFARLAN, F. Information Technology Changes the Way You Compete, *Harvard Business Review*, 98–103, May 1984.

PORTER, M. *Competitive Strategy*, The Free Press, 1985.

SCOTT-MORTON, M. (Ed) *The Corporation of the 1990's*, Oxford University Press. 1990.

SCOTT-MORTON, M. (2) *The Corporation of the 1990's*, Oxford University Press, 1990, 299–309.

ZUBOFF, S. *In the Age of the Smart Machine*, Heinemann 1988, 9–11, 159–171.

MACDONALD, K.H. Future Alignment Realities, *Proc. of Unicom Conf. on Creating a Strategic Business Based on IT Policy*, 1991.

HAY, M. & WILLIAMSON, P. Strategic Staircases, *Long Range Planning*, V24 no. 4 pp. 36–43, August 1991.

Biography

Bob Thurlby graduated in Chemistry from Durham University and worked in the management services departments of a number of companies before joining ICL's

Product Support organisation in 1976. In this role he was responsible for providing technical advice and guidance to ICL's early customers of IDMS and DDS.

In the early 1980's his work concentrated on the use and application of data analysis and database design methodologies and he used his experience to contribute to the development of ICL's Fourth Generation System Building software.

Since 1986 he has been a Principal Consultant working with ICL customers in the Electricity Supply Industry. In this role he has led a number of major development projects for applications to support the engineering, commercial and generation functions of the Electricity Industry.

In recent years he has specialised in the development and application of Information System Planning methodologies and is currently on a secondment to Brunel University as a Visiting Fellow, where he is undertaking research into this subject.

Describing Systems in the OPEN*framework* Integration Knowledge Base

Stuart O'Connor
ICL OPEN*framework* Division, Manchester, UK

Abstract

The development of more complex information systems, enabled by recent advances in distributed computing and integrated systems, has resulted in a more architectural approach to describing systems. This paper discusses how information systems may be described formally. The proposed fundamental component is a Solution; which has a finite set of properties; an information system is made up of a combination of solutions. Direct applications of this theory include: structural design of a database to hold integration information; replacing structured methodologies by computer based analysis tools; and checklists for information capture.

1 Introduction

ICL's OPEN*framework* provides a systematic approach for enterprises who wish to maximise the effectiveness of information systems in their particular business. It is valuable to enterprises who see the need for responsiveness and change in a turbulent business environment and is particularly suited to enterprises who wish to realise the benefits of open systems in terms of cost effectiveness, potential for change and interaction with business partners. [Brunt, 1993].

The OPEN*framework* Systems Architecture provides the vocabulary for an enterprise to create a blueprint or *architecture* for its own integrated business and information systems. The architecture is described in overview in [Brunt and Hutt, 1992], and its purpose is explained in other papers [Brunt, 1993; Kay, 1993] in this issue. Using OPEN*framework*, an enterprise can first consider its requirements in the form of a structured model which enables managers to explore requirements for change in a relatively cheap and risk free manner, and secondly use OPEN*framework* information to populate their modelled systems from many available open systems products. In

offering this vision, it is recognised that different enterprises need to start at different places in considering change for their business or information systems and have different degrees of investment in existing equipment and systems which must be accommodated. There is potential to improve this process in terms of speed and accuracy by developing computer based tools. It is an easy step to put integration information on a database; automating the structured methodology process would necessitate tooling – the formal basis of which should be synergistic with the information database.

Services based on OPEN*framework* techniques and associated material such as workbooks, software tools and information are available for use by the enterprise in realising some or all of the benefits of OPEN*framework* in an incremental manner.

It is convenient to consider these services in the following categories:

Modelling Services:	To help the enterprise create an OPEN *framework*-style computer model of all or part of its business and information system activities as a basis for exploring changes; to provide tools and collateral items to help with the exploration of the models.
Product Choice:	Matching the requirements produced by modelling to the most suitable products to extend and build operational systems.
Operational Support:	Helping to ensure that the enterprise gets the best value from its operational systems.

The OPEN*framework* Integration Knowledge Base (IKB) is a repository or library of information which supports systems integration in general and the types of services described above. In the particular support of these services, the IKB holds the following types of information:

OPEN*framework* architectural definitions:	OPEN*framework* Architecture defines terminology, classifications, interfaces, units of measurement, etc. which are used in the building of modelling objects.
Re-usable modelling objects:	Objects which support the modelling services. These objects are normally tailored to the enterprise specific requirements through one of the modelling services and then stored in the enterprise's own repository in the form of the enterprise's own, customised architecture.
Product choice advice:	Information which helps the enterprise choose the best available product to perform a task. This information ranges from general terms such as recording trends in the information industry to quite specific help in matching a set of modelled requirements to the best choice of available product.

Operational Support: The IKB itself is too general to support a particular enterprise's operational system. However, the modelling objects supplied from the IKB are tailorable, using OPEN*framework* services, into a form usable to support the operational system and to form the basis for systems management activities. This is a natural extension of the enterprise-specific architecture held in the enterprise repository.

2 Solution Objects Within the IKB

A central concept for systems description in the IKB is that of a *solution.* A solution object in the IKB represents part of an information system. It may be a large or a small part, so solutions may be composed of sets of other solutions. It may be described in abstract form when its function is known but its realisation is not, or in concrete form when the function and its realisation (that is, the choice of specific hardware or software products) have both been decided.

The term *solution* was chosen to emphasise that each solution object performs a function which is the solution to some set of requirements. A *solution object* in the IKB can represent any part of an information system. The term is not used in the marketing sense meaning "an application". In [Brunt and Hutt, 1992] the terms *component* and *assembly* are used for a similar concept. One disadvantage of these terms is that they reflect the bias of the observer – one man's component is another's assembly.

The diagram below shows the area of applicability of the Solution object in describing the ranges of systems; large to small, abstract to concrete.

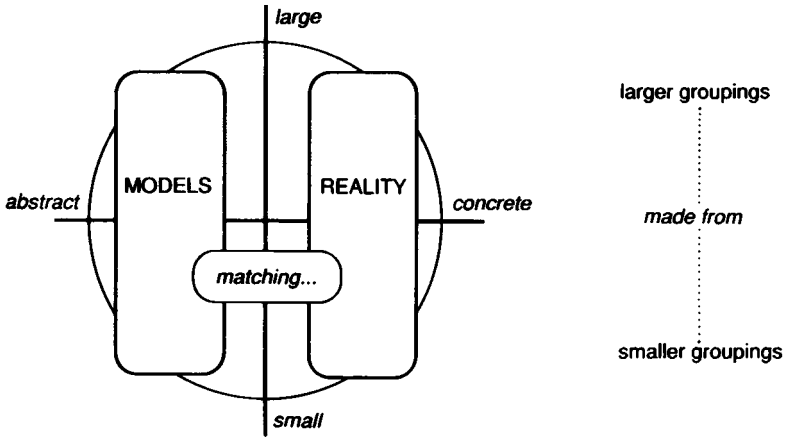


Fig. 1 Applicability of solution objects

Using the same basic object to describe all these areas is valuable because:

- larger solution objects can be described recursively in terms of smaller solution objects and,
- complex attributes of modelling objects, (*abstract* solutions) can be matched with real world products or product groupings, (*concrete* solutions).

The solution object is presented as a named entity whose attributes are formulated in terms of OPEN*framework* architectural definitions; its definition is built up as described in the following sections.

2.1 Solution Composition

A solution is an encapsulation of the whole or a part of an information system either in concrete or in abstract form or in a mixture of the two forms. It is generally composed of smaller solutions except for the most elementary solutions which are indivisible. It therefore follows that, on the concrete side, the indivisible solutions are products; and, on the abstract side, there are some indivisible abstract solutions whose only necessary property is that they can be mapped to concrete products.

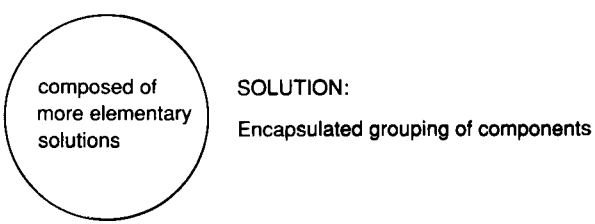


Fig. 2 Composition of solution object

2.2 Solution External Definition

At the boundary of the encapsulation, the solution is defined in terms of:

- the *services* which it offers (as in client server terminology), and
- the *services* which it requires.

Both the services offered and required are expressed in the same format which will be described later. This makes it easier to match the requirements of one solution to the services offered by another.

2.3 Additional Information

The services offered and required by a Solution object can be described in terms of interface specifications. However, there are other important characteristics of a solution which have an important bearing on the suitability of

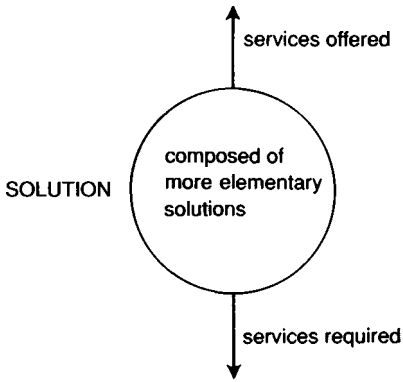


Fig. 3 External interfaces of solution object

components for particular applications. It is important that these qualifications are captured in the IKB.

The IKB distinguishes two kinds of qualification information: *verified information* that emanates from either the product authority or has been assured by specific testing, and *unverified* reports submitted by users from their own experience. The former category is known as *advice and guidance* (A&G), the latter category as *commentary.*

A shell of such information is added to the solution object during its life in the IKB as shown in Figure 4.

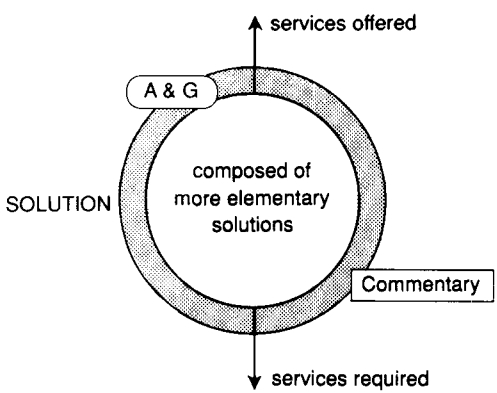


Fig. 4 Adding users' experience to solution object

2.4 Adding Solution Relationships

Solutions have been shown to be related by the component relationship, one solution referencing a set of other solutions for this purpose. However

it is expected that solutions may be linked together for other, as yet unforseen reasons. In order to accommodate this possibility, the IKB provides a facility for solutions to be related with other solutions and with other forms of IKB objects by means of the *link*. Links are another form of IKB object which enable a grouping of solutions to be created dyamically; the reason for the grouping is expressed in the link object itself.

One use of this is when there is some information which applies when two or more independent solutions interact (e.g. when some information is known about a particular interaction of two or more solutions which have not been grouped as the components of another solution). In this case, the link object groups the solutions together and acts as the storage place for the information which is relevant to that grouping. Another use is when it is known that one or more Solutions are a good match for the requirements of another.

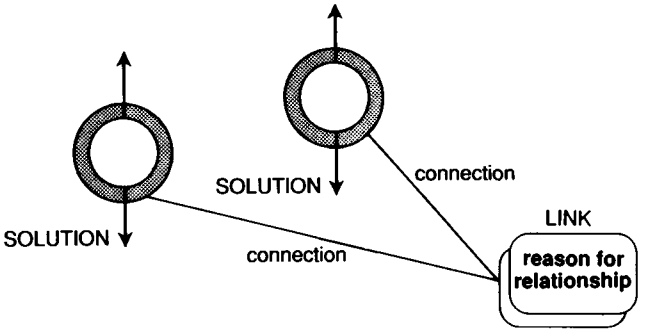


Fig. 5 Linking solution objects

2.5 *Versions and Variants*

It is expected that Solution objects will be updated from time to time in order to add information or amend existing information. Updating is supported by the concept of *versions*, a new version of the solution object is created for each amendment without automatically destroying the previous version. In general, when a solution is referenced from another object, there is an implication that it is the latest version which is being referenced. However, for particular purposes, a reference may be to a particular version of a solution which has the effect of ensuring that any information referenced will not be changed with time. Clearly, care must be taken when deleting old versions that these are not being referenced by current versions of solutions.

When creating a new solution it will frequently be the case that the new solution is a variation of an existing solution. A *derived solution* may be used to simplify the creation of such Solutions. If a new solution, solution Y, has the attribute '*derived from = solution X*' then this means that all the

attribute values of solution X apply to solution Y except where they are explicitly overwritten (non NULL values exist) in solution Y. Solution Y may refer either to a specific version of solution X or to the latest version. Referring to a specific version simply avoids storing replicated information. Referring to the latest version ensures that subsequent changes to solution X implicitly affect solution Y as well.

2.6 Administrative Information

Finally, there is a set of information associated with the solution concerned with its existence and status within the IKB rather than with its system descriptive properties. This information includes a *solution identifier* which is unique within the IKB, but the solution may also have pseudonyms, other names by which it is known in various environments. The object is the smallest unit of information within the IKB to which access can be controlled and so it is the atomic unit of integrity; this includes *integrity of ownership*, which is the right to permit others to access the information and the *integrity of origin*, which is the responsibility for the degree of accuracy of the information.

3 Presentation of Solution Objects

3.1 Attribute Naming

The information held in a solution object is presented to the user in the form of a set of packages each of which is named and typed. Using object oriented terminology, these packages map onto the concept of attributes of the Solution object. These packages can be likened to a set of paragraphs or sections in an SGML conformant structured document [van Herwijnen 1990] and they are named and presented in an hierarchical form.

Each attribute, as well as being named, has an associated *datatype* which explains the way the information held under that name is encoded. Typing is discussed in a later section but, for the present, it is sufficient to say that there are two broad categories of datatype – those suitable for computer operations (e.g. integer, enumerated types ...) and those which are really only suited for human browsing (e.g. diagrams, text). The latter do of course have to be machine intelligible for the purpose of display and text can be searched using a comparatively simple character string matching algorithm.

In general, it is desirable to hold information which can be usefully fed to machines in as precisely typed a form as possible. This usually implies many attributes being held in fine grain form but unfortunately we start from a position of relative inexperience in knowing what these useful, fine grain attributes are in detail. Thus, it is recognised that for different kinds of solution, and indeed for different versions of the same solution at different points in time, it will be desirable to express named attributes in increasingly fine grained form. For example, it may be that in time it is possible to

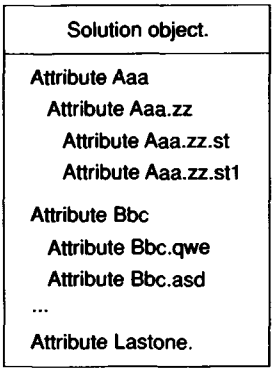


Fig. 6 Attitudes of solution objects

improve the precision of some information within the same solution type; thus a named attribute holding that information as a text item may be developed to have an additional named attribute of a different datatype, say *integer* or *table*. At the same time, it is desirable that the information held in all kinds and versions of solutions is coherent and compatible and so it is desirable to impose some regular pattern on the organisation of attribute naming across all solutions.

This is achieved by using the technique of *type inheritance*, used in object oriented programming. Every solution *instance* is derived from a template or *type* and the attributes of every solution template are organised according to a common root template which specifies a mandatory set of top-level attribute names, similar to the chapter names in a book. From this root template, further solution templates may be derived by adding attribute names, but these must be added according to the special rules for solution objects. Each new attribute is added within the scope of an existing attribute and in fact takes that attribute's name as a prefix. This causes the attributes to be organised in an hierarchy similar to that of a document where there are a fixed number of Chapters but paragraphs and sub-paragraphs may be added in a structured manner.

There are many reasons for this form of organisation; firstly, each attribute value in all solutions falls under one of the 'well known' top level attribute names (i.e. subject headings). Secondly, it is possible to form a query of the type "retrieve all solutions where there is something known about attribute Name_". Such a query can be processed by checking all the solutions formed from the same template and also all the solutions formed using templates which are descended from that parent. Thirdly, structured name formation helps when creating a new unique attribute name, and also helps in enabling blocks of similar named sub-attributes to be used under more than one owning attribute name – this is shown in the re-use of the service specification

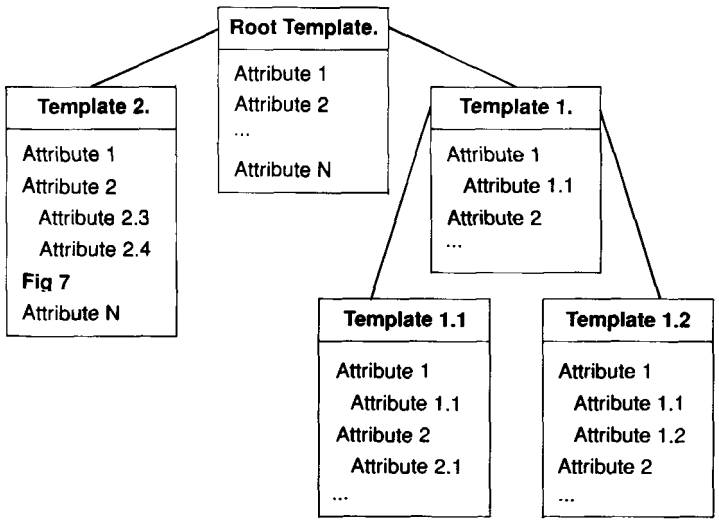


Fig. 7 Control of attribute naming

under the services offered and Services required sections in the example 'root template' below.

3.2 The Solution 'Root Template'

This section summarises the top level set of attributes which characterise the Solution Root Template, incorporating the facets of Solutions which have been previously discussed.

1. IKB administrative information
   e.g. solution name, synonyms, owner, derived from, date created, access permissions, etc.
2. services offered
   in the form of a service spec. – see section ...
3. services required.
   in the form of a service spec. – see section ...
4. advice & guidance
   help under sub-headings ...
   4.1 installation
   4.2 customisation
   4.3 etc.
5. internal composition
   component parts & how this solution is constructed
   5.1 solution components
   set of references to other Solution objects
   5.2 constructors (see later).
   explanation of how the components are connected together

6. links to requirements objects
   special use of IKB link objects for this purpose
7. other links.
   other references to links to form groups for purposes defined under sub-headings ...
8. commentary.
   users' experience of this object ... in their own words

3.3 Attribute Datatypes

Each named attribute is assigned a *datatype* which is a specification of its encoding. Much has been written on this subject, the OPEN*framework* information management reference architecture provides a guide. Datatypes are traditionally recognised in programming languages as forms which can be manipulated in a particular way. Such simple common types are *integer, float, character*, more complex are *table, diagram.* More recently, in the PC world, typing occurs on a larger and more complex scale with files taking various well known suffix names (e.g. .TXT .DOC .XLS) to identify their internal encoding and to define what operations may be performed on them (usually performed by particular applications such as Excel, Word for Windows, etc.).

The IKB has to resolve some conflict between the number of datatypes in the outside world and the number with which it can afford to deal. The following model illustrated in Figure 8 shows the approach taken.

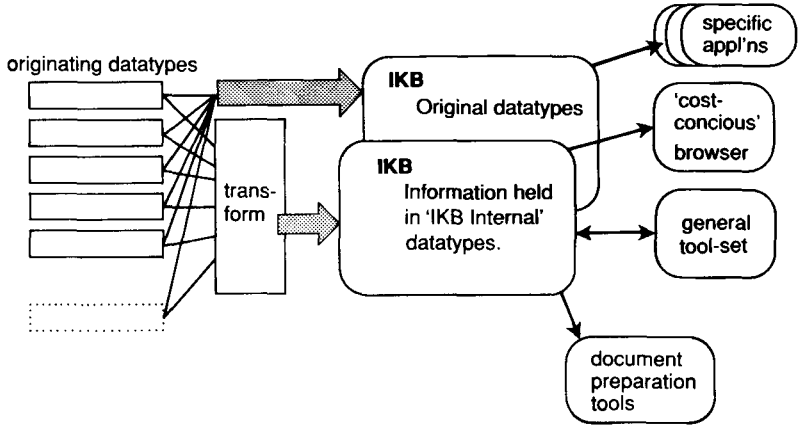


Fig. 8 Use of original and internal datatypes

The IKB acts as a repository for information for a number of purposes. These may be classified into simple browsing or document preparation and tool-based operations, many of which are concerned with modelling. The definition of a core set of IKB internal datatypes sets the agenda for the

browser and tools which must be capable of dealing with these expected datatypes. The precise definition of this core is determined on one hand by the need to support an adequate range to meet the needs of the users (information consumers) and on the other hand by the expense of providing a variety of software methods within the tools to cope with the variety of datatypes.

Additionally, on the input or data capture side, it is recognised that the world produces information encoded in many datatype forms and it is important that the IKB can make the transformations between these and the internal datatypes with minimum loss of information. In passing it is interesting to note that such transformations do not *necessarily* lose information and that information can be gained by, for example, recognising that what is apparently a text input item is in fact a reference to an IKB object. However, in order to guarantee a service whereby no information is lost, the original datatype encoding is preserved within the IKB, labelled in such a way that it can be given back to the particular application type able to understand it.

3.4 Hypertext Presentation

The organisation of information into solution objects and the further definition of contained information under structured attribute names makes a hypertext presentation very attractive as one option for browsing. Attribute values within the context of a solution can be viewed at various lexical levels, following the naming structure and highlighting the presence of lower lexical levels, which can be expanded individually on demand. Also, references to other objects can be highlighted on screen as starting points for hypertext links to other objects.

4 Solutions, Service Specifications & OPEN*framework* Architecture

It has been stated that a solution object within the IKB represents an encapsulation of some part of an information system and that at the boundary of the encapsulation, the solution definition is in terms of the Services which it offers and requires. Both the required and offered Services are expressed in the same form – the *service specification* which is itself expressed in terms of the OPEN*framework architecture*. By expressing the requirements and the offerings in the same form, we provide the basis for *comparability* and *matching*. Also, we provide the basis for the process of *construction* which is used to explain the interior of the Solution object – but more of that later. The OPEN*framework* architecture gives us the basis for consistent classification and terminology which ensures that the information held in the service specification retains a high value by being highly reusable.

The service specification, the totality of the Service offered/required, is *decomposed* into more manageable units called *functions*. Each function is an

expression of *what is done or required to be done.* Typically, a solution will offer or require a small number of functions but technically there is no limit.

Associated with each function is a set of *interfaces* which defines the means whereby the function is accessed. This is an expression of *how something is caused to be done.*

Also associated with each function is a set of *quality statements* which express *how well* the function is (to be) done. The Quality Statement is made in terms of a set of *workload/value* pairs; the *workload* expresses the stress on the system and states what is to be measured under these conditions, the *value* associated gives the values attached to these measures either as observed or as required. Note that the Workload may be in some cases quite general (e.g. "under all conditions" the processor maintains an "average of x MIPS") or it may be quite specific (e.g. the "XYZ Transaction Benchmark" is measured at "y Transactions/sec".). Note that the service specification, in terms of function, interface and quality statements, work equally well when expressing the capabilities or requirement of a solution. This leads to the important conclusion that, when used in its abstract form, the prime role of the solution object may be a requirements statement for a concrete system.

The Service Specification may be summarised as follows, using pseudo-BNF notation.

Service Spec	*Function {Function ...}*	//	
Function	Function Name,	//	each function has name
	Function Description,	//	freeform description
	Interface {Interface ...},	//	Named Interface[s]
	Quality Statement.		
Quality Statement	[Workload, Values] {[Workload, Values],...}.	//	set of Workload/values pairs.

Fig. 9 Formal specification for service definition

In order to formulate these service specifications in a controlled and therefore *comparable* and *matchable* format, it is desirable to control the values which can be entered for particular fields in the Specification. This is where the vocabulary provided by OPEN*framework Systems Architecture* comes in. Function names, interface names, workload names, and the units in which values are expressed are all examples of fields in the service specification which may contain only entries whose values and meanings are specified by OPEN*framework* architecture. This is not merely a bureaucratic rule but one which enables an engineering approach to be taken to the definition of systems by a technique analogous to the use of *enumerated data types* in standard programming languages.

Interfaces are closely associated with function and are controlled similarly. Note that in the solution context, the term interfaces includes hardware safety standards, programming interfaces, communications protocols and the specifications of objects which are transferred between entities which may be independent of the means of transfer (e.g. files or EDI documents).

For certain industry standards, particularly those known as open interfaces, there are conformance standards and certification processes which can provide users with a degree of confidence about products' interfaces. *Conformance statements* may be available for a product explaining certification of interfaces offered by that product. Within a solution, an interface may be corroborated by such a conformance statement, derived from a product within the Solution. The Interfaces offered by a product or system tend to be more valuable when corroborated by conformance statements.

For the quality statement, OPEN*framework* qualities reference architectures are the basis for the definition of the workloads or circumstances in which a specified set of measurements are to be made. This concept is well developed in the case of systems performance benchmarks defining workloads and measurements to be made but there is more work to be done in providing such definitions in other areas such as availability, usability and so on.

5 Solutions & Constructors

As already explained, the solution object can be composed of other solution objects and is characterised at its boundary by its service specifications. The composition of a solution can be described as simply a list of component Solution objects but this alone will not tell how the components have been arranged or constructed to form the whole. To provide the full description of the internal construction of a solution, another set of attributes, collectively called the *constructor*, is needed to describe the means by which the component solutions are connected together.

Within the industry, various forms of constructor are referred to in various degrees of precision; client/server, network service, operating system programming interfaces are all forms of constructor – that is, means of describing a relationship whereby discrete entities (e.g. Solutions with external Interfaces) are arranged to interwork.

Within the IKB and its associated modelling services, two parallel approaches are taken towards constructors. On the one hand, simple formal constructors (e.g. programming interface bindings) and complex informal descriptions (e.g. client/server diagrams and textual descriptions) are in common use and must be useable wherever appropriate. Informal descriptions are particularly useful in the context of describing the *content* of a solution, as this is usually a description of something concrete which has been constructed and is known to work (this information being held in the A&G and commentary parts of the solution). On the other hand, a highly

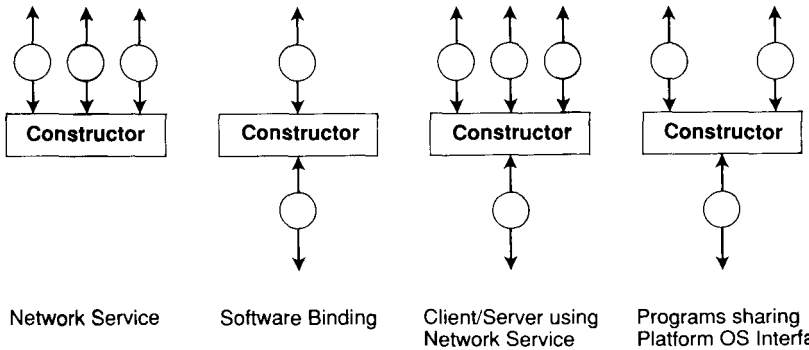


Fig. 10 Examples of constructors

formal approach to complex systems constructors is desirable when the external service specifications of a solution are to be used as the basis for exploring the *possibilities* of creating larger, novel configurations of solutions which have probably not been previously constructed. This area is currently being researched with a view to providing low risk constructability services and involves the use of further information in the form of *rules*.

6 Conclusions

This paper has introduced the Integration Knowledge Base (IKB) and the solution object as a tool for recording information about information systems. It has shown how the solution object uses OPEN*framework* architecture to organise and label information within it and how standard object oriented techniques of encapsulation, specialisation and inheritance are used to help in the management of complexity.

The Solution object is shown to be capable of specifying information system requirements in an abstract manner as well as recording information about instances of systems built from real world products. OPEN*framework* services are being developed continually to assist the enterprise in the formulation of requirements and to provide information about real world systems which assists the enterprise to satisfy requirements most effectively.

7 References

Solution objects have been developed as a practical tool using relatively standard, hopefully low risk, techniques which have been popular over the past ten years, particularly under the data management and object oriented labels. In this work originality was not of concern and nearly everything has been borrowed from somewhere, and not necessarily acknowledged, (as is usually the case in the software industry). However, the following reading list has been valuable and in some cases also entertaining to the author.

References

BRUNT, R.F. and HUTT, A.T.F. (eds). *OPEN*framework. *The Systems Architecture: an introduction.* Prentice Hall, 1992.

BRUNT, R.F. *An introduction to OPEN*framework, *ICL Tech J.* Vol. 8 no. 3 pp. 351–364, 1993.

COAD, P. and YOURDON, E. *Object Oriented Analysis.* Yourdon Press, 1991.

GRAY, P. *Open Systems: a business strategy for the 1990's.* McGraw Hill, 1991.

van HERWIJNEN, E. *Practical SGML.* Kluwer Academic Publishers, 1990.

KAY, M. H. *The evolution of OPEN*framework. *ICL Tech. J.*, Vol. 8 no. 3 pp. 365–382, 1993.

MEYER, B. Object Oriented Software Construction. Prentice Hall, 1988.

OBJECT MANAGEMENT GROUP *The Common Object Request Broker.* OMG Inc. 1992.

RINGLAND, G. A. and DUCE, D. A. *Approaches to Knowledge Representation.* Research Studies Press Ltd, 1988.

Biography

Stuart O'Connor

Stuart O'Connor has worked for ICL since 1966 in many capacities as software and systems designer. In the late seventies he was Systems Design Manager for the 2966 system and in the early eighties became one of the patent holders for the System 39 Macrolan fibre optic system for multiprocessors. As a VME architect he has had responsibility for Ingres, X.400, and VME/X before joining OPEN*framework* Services Group as an information architect.

Multimedia and Standards for Open Information

Ian R. Campbell-Grant
ICL Fellow, ICL, Bracknell, UK
C. R. Smethurst
Company Architect, OPEN*framework* Division, ICL, Kidsgrove, UK

Abstract

This paper is divided into two main portions; the first introduces the key multimedia technologies and outlines some of the impacts on society consequent on achievement of a global networked multimedia capability.

A key technology is the establishment of generally accepted standards for representation of multimedia information. The second part of the paper looks at this in more depth, relating it to OPEN*framework* and indicating the key areas in which standards are needed and are evolving.

1 Introduction

This is a wide topic and this paper is divided into two main portions. This first part deals with what is meant by multimedia, introducing the key technologies and outlining a few of the impacts that achievement of multimedia technology will have on society.

There is widespread recognition of a user need to handle multimedia information in an integrated manner from the desktop, and for a worldwide networked facility for interchange of such multimedia information. Such a worldwide multimedia information capability would depend on five major factors:

- desktop systems (hardware and software), providing user and application capabilities to access and manipulate digital multimedia information;
- networking connections, providing the physical basis of a communications system to interconnect one desktop with another at the appropriate bandwidths and cost needed for generally available and responsive multimedia communications;

- infrastructure facilities, providing the carriage of multimedia messages, with reliable routing and transmission;
- information types, providing generally accepted standards for the various types of multimedia information;
- provision of libraries, providing access to information in digital, multimedia form.

Whereas the first of these (desktop systems) need not adhere to standards, provision of the remaining factors depends on the adoption of generally accepted standards. Global cooperation is necessary to establish such standards.

In view of the significance of this area and the recognition that it impacts many aspects of system design, an OPEN*framework* multimedia *specialisation* is specified, this is dealt with in more detail in Sections 3, 4, 6 and 7 of this paper.

When a networked multimedia capability is generally available it can be expected to have significant implications for society, for example giving a general availability of "telepresence" capabilities at a much higher level of facility than the current phone or video connections, bringing the "global village" closer to reality. It should also have significant positive environmental implications (eg a reduction of travel by making work come to the worker) and also make collaboration much more flexible – as once remote working practices are widely established distance will become much less of a barrier to joint working.

To look at one area more closely, education can be substantially assisted by making "interactive text books" practicable, for example using moving images (eg to illustrate growth of plants), providing interactive atlases to highlight geographic information in a manner responsive to students' needs, or by incorporating references to "virtual museums" accessible via multimedia libraries.

With such information networking capabilities, electronic universities will become practicable – lecturers and teachers time will be used more efficiently and may be used for remote advice on demand. It will be necessary to provide full accredition for such courses.

2 Multimedia Technologies

This section outlines the above technologies in more detail.

2.1 Desktop Systems

The desktop workstation will incorporate a modular core of digital multimedia services facilitating applications in accessing and processing multimedia information – this will handle many of the complexities of multimedia

information for the application software. These services will also provide an end-user with easy-to-use facilities for capture and presentation of multi-media information.

The services will provide synchronisation and timing facilities and will make such features as compression/decompression transparent to the applications and the end-user. The desktop workstation will also act as the "telecomputer" – handling all the telecomms aspects within its core of services.

2.2 *Networking Connections*

This area is developing rapidly but much higher bandwidths and reliability are required than are currently available. The first protocols for real-time digital multimedia connections are still under development today.

Services are needed providing high bandwidths, to carry uncompressed digital TV and to support a scalable user base, growing to many millions of users.

To respond to this demand, networking probably needs to evolve from todays "root and branch" networks to the general provision of switched star networks. Fibre optic cables and protocols appropriate to this carrier are capable of providing such networks with the much higher bandwidths that are required.

To be generally accepted, communications will also need to be far cheaper than is the case today. As an illustration of the order of magnitude, services will probably need to provide a thousand-fold increase in bandwidth with only a ten-fold increase in cost.

2.3 *Infrastructure Facilities*

To use the networking connections there is a need for services to manage the network and to provide a seamless networking environment. These services need to guarantee bandwidth, error quality and a limit to connection delays. Reliable connection is key to widespread adoption and in case of problems the services must provide graceful degradation. Addressing, routing and delivery must be both simple and secure, as private information and valuable corporate information will be transmitted. Comprehensive directory services are necessary.

The provision of such network services to support global interoperation will require a centralised development between the world's PTT suppliers and depends on their cooperation on an unparalleled scale. Other suppliers such as Cable TV network suppliers will also be involved in this cooperation.

2.4 *Information Standards*

Multimedia information is of various types, such as audio, video, animation, film clips, graphics, images as well as text. This information needs to be transmitted between systems and to be stored for substantial periods of time. There is a need for generally accepted standards for all types of information.

Hardware support will be required for information compression/decompression; limited forms of compression already exist in connection with the facsimile services, but in general there exist today no compression algorithms for high-quality multimedia information.

Examples are the digital standards under development by ISO for photographic and motion pictures, and also High Definition TV standards. A number of levels are under development within these standards to provide for different requirements.

These information standards will also need to allow for a standard "look and feel" for the multimedia information.

2.5 *Provision of Libraries*

To exploit the multimedia networking capabilities, a comprehensive set of multimedia libraries will need to be established. Such libraries should, for example, provide access on demand to multimedia information eg video sequences, (for example, retrieval by a student of an interactive reference work).

Electronic publishing and authoring tools are a step in this direction, but for the success of such services widely accepted standards are necessary. For such libraries to be economic an "information economy" needs to evolve whereby information can be bought over the network.

3 Standards for Multimedia Information

The paper now deals in rather more detail with information standards for multimedia.

It is not reasonable to expect that the above technologies will be introduced in a coordinated manner just to create a global network capable of handling multimedia information. There must be more immediate drivers, consequently it is important that these information standards are needed also by the IT business today.

Businesses need such standards because there is a major trend towards storage and interchange of electronic information within and between companies. Such information needs to be transmissable between multi-vendor systems, to be processable between such systems interactively and to be

retainable over substantial periods of time. Although studies still indicate that only a relatively low proportion of information is held in electronic form, this proportion is increasing rapidly in European businesses; during this decade companies will become heavily dependent on electronic information handling to remain competitive. (Scott-Morton, 1991.)

To respond to these trends, an OPEN*framework* Multimedia Specialisation has been developed; it is concerned with how such information is handled by IT systems and ought to be a major concern for system integration specialists.

Any multimedia architecture needs to present a route forward which has a close relationship with key elements of many of the strategies that apply to particular types of system component. These include strategies for image systems, filing systems, databases, security, paper handling interfaces (printers/scanners), user interfaces, system interconnection, distributed computing, messaging, and the various interworking services.

Multimedia information is much richer, that is to say more complex, than previous systems such as Ascii/Telex and Facsimile. Conseqently, the standards for multimedia information are extremely detailed and the interchange technologies are far more demanding. The precise multimedia information standards that are adopted will have substantial significance for the internal architectures of many components of information systems.

4 OPEN*framework* Views of Multimedia

From the OPEN*framework* perspective, multimedia is concerned with the set of capabilities that provide access and manipulation of various types of multimedia information intended for human perception.

The Multimedia Specialisation for OPEN*framework* does not specify services that are independent of information type such as database systems and networking systems. However, the specialisation covers some aspects of all other OPEN*framework* areas. Multimedia is specifically addressed by OPEN*framework* in order to provide consistency and synergy across systems adhering to OPEN*framework* in handling multimedia information.

The Multimedia Specialisation Architecture recommends the standard information types to be supported by OPEN*framework*; it also recommends the capabilities that will be provided within OPEN*framework* for such information to be modelled, stored, retrieved, transferred across a network, and manipulated within an information system.

While use of other information types is not precluded within OPEN*framework*, particular support is not recommended by OPEN*framework* for other information types.

Adherence to OPEN*framework* implies the use either of standard information structures and standard content types, or of application specific structures and standard content types.

The Multimedia Specialisation also covers Unimedium Information, which is not handled separately because of the need for it to fit with multimedia information. Thus, unimedium information is regarded as a special case of multimedia.

5 Open Information Standards

A current issue of particular importance in multimedia is that various groups of standards have been emerging with overlapping scopes. It is recognised internationally that work is needed to develop a harmonised set of Open Information Interchange standards.

Major committees on international standards with overlapping scopes include those responsible for:

- engineering information (CAD/CAM and product definition data);
- graphics standards, including hierarchical structures and 3-D processable graphics;
- electronic documents;
- inter-application communication (EDI).

That these activities are in progress in different groups with little common membership leads to a danger of an eventual duplication of standards. The current situation is rather akin to the development of a range of communication standards without any single "OSI model" to interrelate them. Although various models are under development, all are currently far from complete.

This is an expression of the general points on the explosion of standards in the paper by Brunt in this issue (Brunt, 1993).

An integrated approach to standards for information interchange is currently not adequately addressed, whether by formal standards or by proprietary standards. Nevertheless, this is an area where, as we have seen above, users requirements can be stated simply and are urgent.

We should recognise that a proliferation of Open Information standards with overlapping scopes will not benefit vendors or users. Consequently OPEN*framework* is cautious in this area and is using ICL's established position in Open Systems to influence work towards convergence or alignment of the models applicable to all of these areas.

The various formal and semi-formal activities underway in this area include:

- The *Open Document Architecture* (ODA) (ISO 8613, 1992) is an ISO standard that initially standardised an electronic analogue to paper documents, together with a processable format. ODA provides a model which could be developed to represent all humanly perceptible information; many detailed proposals have been made to extend it to this full scope. Some of these are now at an advanced stage of development, for example a "HyperODA" extension and an audio content architecture. In addition extensions are proposed to extend ODA to allow handling of documents by automatic applications.

- For *graphics* ISO has a family of standards whose scope includes standards for all forms of information that can be presented visually to a human being. These include the Computer Graphics Metafile (CGM) standard (ISO 8632, 1987) and the "PHIGS" standard.

- ISO standards for *document interchange and manipulation*, also include the Standard Generalized Markup Language (SGML) (ISO 8879, 1986) aimed at publishing systems, and development of other standards in this area of application is underway by ISO/IEC JTC1/SC18/WG8 including:
  DSSSL (Document Style Semantics and Specification Language). The emerging DSSSL standard is designed to support all human perceptible information as well as to provide for the handling of documents by automatic applications;
  SPDL (Standard Page Description Language). This is designed as a form suitable for describing any fully formatted documents that can be represented by laser printer technology.
  Technical directions for alignment and harmonisation of these standards between themselves and with ODA have been agreed by ISO but are being implemented only to a limited extent.

- ISO and CCITT are developing enhanced standards for:
  - photographic information within the Joint Photographic Expert Group (JPEG);
  - motion picture information within the Motion Picture Expert Group (MPEG);
  - High Definition TV standards (HDTV).

- *Electronic Data Interchange* (EDI) standards have been aimed initially at interworking between applications. The scope is now being broadened by the development by ISO of an "Open EDI" model which provides a comprehensive framework for various information standards to be used in conjunction with EDI standards, and applying to all forms of information that are human or machine perceptible or processable. This "Open EDI Model" is under development by ISO, but is not yet fully fleshed out.

- For *engineering and product definition data*, a comprehensive standard known as the STandard for the Exchange of Product model data (STEP) is under development by ISO/TC 184/SC4/WG1 and an intercept of this

standard exists known by the acronym IGES (NBS, 1988). The STEP/IGES standards are aimed at standardising all forms of engineering information, and are to provide for inclusion of documentation, graphics etc., as well as engineering data, thus encompassing the scope of the ODA, CGM, SGML, DSSSL, SPDL Standards and maybe also EDI.

- The US Defense Department CALS (Computer-aided Acquisition and Logistics Support) programme (US DoD, 1990) is selecting standards to define how the various forms of human- or machine-processable electronic information shall be interrelated. This framework is currently under study for application to NATO.

- In addition, various proprietary standards exist or are under development, including:
  - Digital CDA (Compound Document Architecture)
  - IBM MODCA (Mixed Object Document Content Architecture), and IIA (Information Interchange Architecture).

  These are being developed to become full multimedia architectures. Both companies see these standards as having substantial influence on the internal architectures of many components of their information systems.

A particular attempt at convergence by standards for multimedia and hyper-media was that, a special group was established in ISO at a very high level, (reporting to the ISO Technical Committee responsible for all generic standards for information technology) to determine the direction for multimedia and hyper-media standards. ICL played a significant role in this group and avoided developing new architectures for multimedia/hyper-media. The leaders of the ISO technical work on ODA and the ISO work on SGML jointly developed an agreed proposal that the ISO Subcommittee responsible for ODA and SGML should have overall responsibility for a single model applying to all standardisation in the field of hypertext and multimedia. This work would be handled in ISO by means of extensions to existing work, in particular ODA with the "HyperODA" extension.

6 OPEN*framework* Multimedia Reference Model

The standards outlined in Clause 5 of this paper have many architectural approaches in common. In order to understand how these may interrelate it is useful to have a common model covering the various standards. This section describes an OPEN*framework* reference model containing the major system components. Each of the major system components is then described in more detail.

The demand for multimedia systems and applications is driven by the business information requirements of an enterprise. Consequently multimedia systems are constructed and applications are designed to facilitate and enhance the business processes, with the objective of providing the

enterprise with competitive advantage. This is an evolutionary process, of which the regulating factors are cost effectiveness, system usability and the rate of advance in multimedia technology.

In order to build effective and open multimedia systems there is a need to define the standard information types that multimedia systems will manipulate; there is also a need for a development environment and tools to facilitate systems construction. Figure 1, below, illustrates the main components of a multimedia system using a model based on the OPEN*framework* structure model. The major components are highlighted.

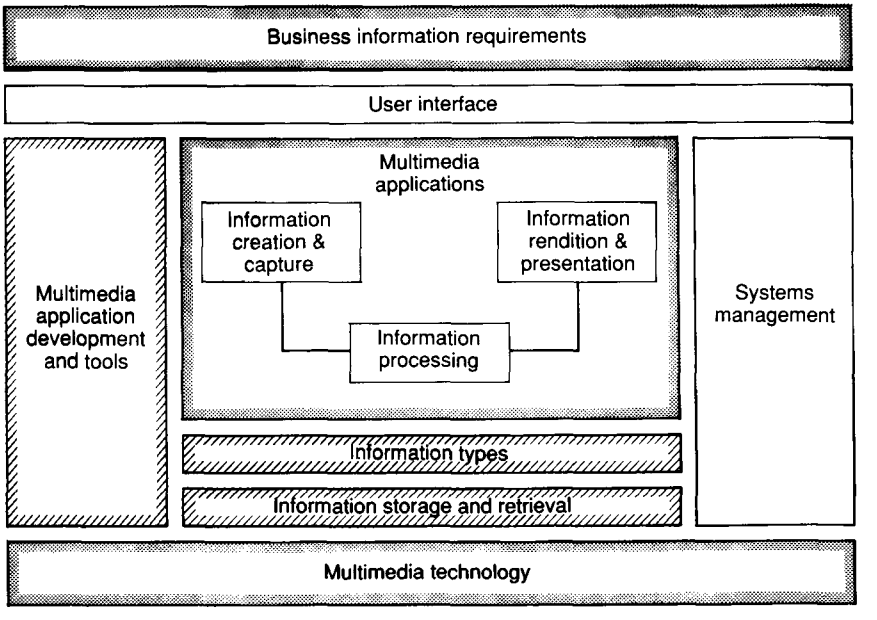


Fig. 1 Components of Multimedia System

- Business information requirements.
  These can be varied and wide-ranging. The underlying requirement is to assist the automation of business processes, primarily by communicating business information between people efficiently and effectively. In addition, some automatic processing of this information by applications can be introduced to facilitate business processes. Thus, multimedia provides for matching and merging the capabilities of computers and people, allowing information to be handled more easily, clearly and quickly.
  Examples of technologies involved are video conferencing, video mail, distribution and storage of high quality documents, imaging and filing correspondence, structured information systems, state-of-the-art business presentation systems, highly interactive education and training packages, and information "kiosks".

- Information types.
  OPEN*framework* standards for information types combine a variety of types and content formats covering both information structures and representation together with data structures and representation (encoding). In order of preference, standards may be International Standards (to facilitate interoperability), de facto industry standards or standards particular to OPEN*framework*. The trend will be increasingly to adopt international standards in order to facilitate use of such information with any applications adhering to these standards. Information types are formed from one or more content types, together with structural information such as logical, layout, or application specific structures, for example, filing structure.
  OPEN*framework* Standards include:
  - Logical structure information which represents the human-perceptible and/or application-specific meaning given to the content.
  - Layout structure information which represents the way that content is to be presented to the user.

  Content types are either general or application-specific. The general content types include: audio, video, raster (image) graphic, geometric (vector) graphic and computational data, as well as character text.
  The subject of information types is discussed further in clause 7.

- Information storage and retrieval.
  Concerns principally secure storage of large amounts of information, sometimes for a long time (often years, sometimes decades). Traditional storage such as magnetic discs and tapes are being augmented by newer technologies, for example, optical discs, CD-ROMs and digital audio tapes.

- Multimedia applications.
  Multimedia applications facilitate business processes, covering information creation and capture, processing, storage, retrieval and presentation.
  - Information creation and capture.
    Information is created or captured for subsequent use, storage, or onward transmission. Forms of information input until recently only possible with high priced specialist components are now readily affordable. Thus, traditional keyboards and mice for input are being augmented with components for capturing other content types such as scanners, cameras, microphones and video recorders.
  - Information processing.
    This concerns modelling, creating, updating and deleting information as well as accessing information for application processing. In addition applications can automate business processes, possibly involving a number of processing steps associated with different individuals or processing elements. Many generic information processing components may be used directly or be customised to a business need, including editors, browsers, converters, and transmission agents such as print spoolers.

- Information presentation.
  This concerns extracting information from the information system, ultimately for human perception. Advances in technology, with the need for improved quality in the perceived information, have lead to high quality, affordable information output components such as printers, display screens and loudspeakers but with high quality visual or audible presentation. Information output components may generate intermediate forms intended for eventual human perception, such as creating a CD-ROM.

– Multimedia application development and tools.
  Tools which handle multimedia information types include: general authoring, audio editors, video editors, graphics and clip art library, animator packages, hypermedia authoring, information type conversion and business presentation.

– Multimedia technology.
  The evolution of technology simplifies the adoption of multimedia systems by making new ways of working more practical. The current key technologies are:
  - high-powered silicon engines,
  - digital information standards replacing analogue standards,
  - standards for compressing information for interchange,
  - support systems such as storage systems, scanners, cameras, speakers (to some extent still to reach affordable prices and performance),
  - application development tools to treat image, audio and video as any other type of information,
  - and finally interoperability standards and conformance and certification capabilities.

7 OPEN*framework* Information Types

A multimedia information type is defined to be "any collection of multimedia information that can be handled as a unit for the purposes of creation, capture, storage, retrieval, editing, interchange, presentation, or processing". The term *infoplex* is used by OPEN*framework* to apply to a standard OPEN*framework* information type. The term *document* is used in its commonly accepted form (paper and electronic); a document is a restricted form of infoplex, and is described later in this clause.

Figure 2, below, provides a simplified information model of objects and processing involved.

The information base contains a set of infoplexes which are subject to infoplex processing. The infoplexes themselves may be either self-contained sets of information or interrelated by hypermedia linkages. In either case, the infoplex itself is essentially a handle onto a subset of the information in

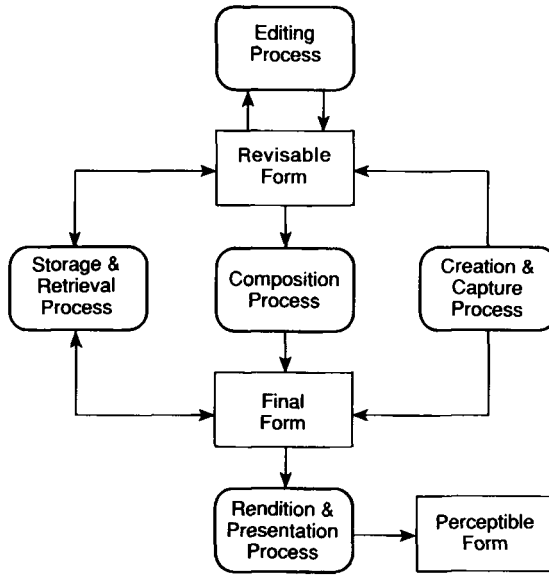


Fig. 2 Information Processing Model

the information base. This handle provides for the subset of information to be handled as a unit for one or more purposes.

This section describes the formal models of infoplex and document objects, and describes models of the processing transformations that can take place on infoplexes.

Information bases evolve with time as do the applications which use them. However, for business purposes all information should be accessible by any authorised users and should not be rendered inaccessible by barriers resulting from technological limitations or technical evolution. It is important to note the danger that information bases may become information islands with no or limited means of information transfer and to take steps to avoid this. Use of appropriate standards is the most significant contribution to a solution.

7.1 Infoplex Model

This reference model introduces the concept of multimedia information modelled by a multimedia "infoplex". A multimedia infoplex consists of content information together with structural information. As illustrated in Figure 3, below, the infoplex is modelled by a process of successive decomposition. This section expands on the reference model to give a more detailed definition of an infoplex in terms of its constituent parts.

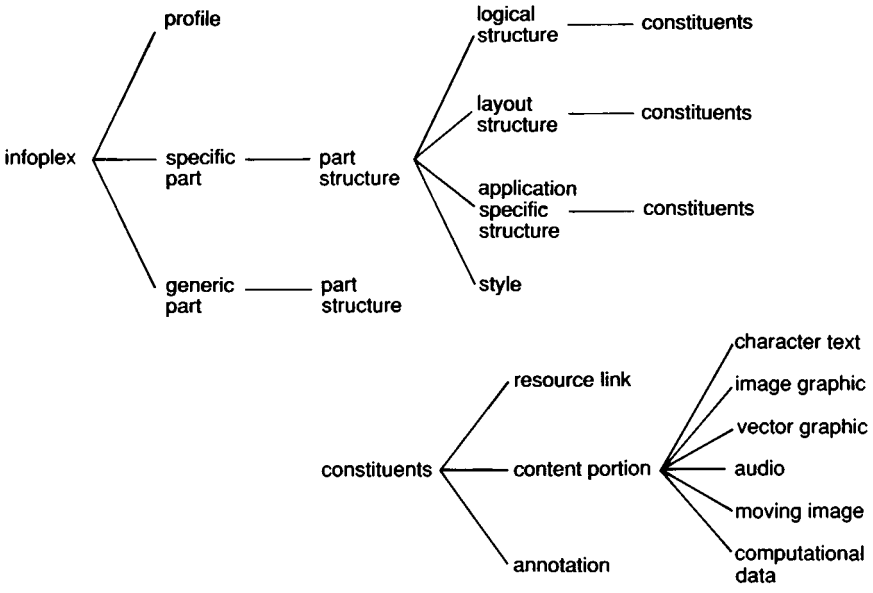


Fig. 3 'Infoplex' Model

In this model, an infoplex consists of:

- An optional *profile* which describes the infoplex as a whole
- Zero, one, or more *generic parts* which specify rules for the structure and content of a class of infoplexes
- Zero, one, or more *specific parts* which give the actual structure and content of a particular infoplex.

The structures of the generic and specific parts are provided by the corresponding part structures. The generic structures specify the rules describing a class of documents, while the specific structures describe a particular document.

Examples of multimedia information that may be modelled by an infoplex range from a simple string of text with no structure, through word processor documents, to "desktop publishing" specifications and hypermedia structures.

This infoplex model is consistent with the models adopted by the International Standards community, principally in their ODA and SGML families of standards, to support general information architectures/hypermedia facilities.

The constituents of an infoplex are listed below.

7.1.1 Profile The *profile* contains information that describes the characteristics of the infoplex as a whole. The profile is concerned mainly with the attributes such as title, author, owner, keywords, date of creation, retention date, archiving characteristics, access controls, and revision history. This information is intended for use in managing the infoplex during storage and retrieval.

The profile also contains technical information such as the structures that the infoplex contains (for example, logical, layout, application specific, or a combination of these), a list of external references made from the infoplex, the coding standards used for the different types of content, the fonts and character sets used, the rules to be applied to the infoplex (for example spell checking, or formulae for spreadsheets). Thus, the profile provides sufficient information to enable:

- an application to decide whether it can successfully present or process the infoplex,
- a recipient to understand what is required to handle the infoplex.

7.1.2 Generic Part The *generic part* is a set of rules which describe a class of infoplexes and constrain the specific infoplexes of that class. The generic part may be constructed as a skeleton (or template) applicable to every specific infoplex of the class, and in this role has its own part structure including content, which allows for factorisation of information.

The generic part may also specify a set of rules describing permissible transitions from one infoplex of the class to another, thereby guiding or constraining editing applications or other application processing. Further, the generic part may also refer to a metaclass definition which can constrain manipulation of the generic part itself. Any number of levels of such a class hierarchy is possible, although in most cases either one level (class) or two levels (class and metaclass) will meet requirements.

The generic part may be included directly in the infoplex when it is stored or interchanged or may be included by reference. The generic part is optional (if omitted there are no template and no rules for the specific part), and also may include many class structures, for example when used to define a class of hypermedia infoplex.

The set of all infoplexes that adhere to a particular generic part is termed an infoplex class.

7.1.3 Specific Part The *specific part* defines a particular infoplex. The specific part defines the overall structure of the infoplex and the relationships between the various structures of which it is composed (logical, layout, and application specific) and between these structures and the content. The specific part is required to adhere to any template or rules specified by the

generic part, if present (or referenced). The specific part is optional; if absent the infoplex is in generic form only. The specific part may include many separate part structures, for example when used to define a hypermedia infoplex.

Part structure

The *part structure* defines the structure of the generic and specific parts in terms of:

- logical structure
- layout structure
- application specific structure
- styles.

7.1.4 Logical Structure The *logical structure* is formed by successive decomposition and thereby defines the set of relationships between logical objects or object classes (components). (Here and elsewhere the term "component" is used to represent objects or classes of object, when there is no need to distinguish between these.) These relationships are used to express the meaning in terms to which humans and/or applications relate naturally. For example, at the overall level this may identify the infoplex as consisting of an introduction, a number of chapters and an index. In turn the chapter may be decomposed into a heading and a number of sections. These objects may in turn be decomposed. The structure provides for multiple logical objects to be combined (for example a "graphic" and a "caption" together might form a "figure"). This kind of grouping can be applied at any level.

When defining a hyperstructure, links may be inserted between the nodes at any level of the logical structure. The decomposition may be recursive allowing the logical objects to be nested when defining a generic structure. However at the lowest level each logical object (for example paragraph, figure, footnote) must contain a link to a content portion.

7.1.5 Layout Structure The *layout structure* is only of interest for formatting or presenting an infoplex. The layout structure defines for each content portion the spatial and temporal parameters, for example, the page and the position and dimension of the imaging area, whether the content portion is to be made visible immediately or later by some trigger, whether the presentation of a content portion is to be synchronised with the presentation of some other content portion.

7.1.6 Application-specific Structure The *application-specific structure* defines application-specific structuring to be applied to the infoplex. For example this may represent the structure of a worksheet or the structural information of a spreadsheet application. Other examples are a structure representing the successive levels of decomposition of mathematical formulae or musical notations.

7.1.7 Style The *style* information describes particular techniques to be applied to processing or presentation. For example, precisions to be applied in manipulation. Other specific examples of styles are the presentation and layout styles used within the infoplex. Presentation styles are used to govern aspects of formatting an image such as indentation, justification, font, emphasis, line spacing, colour and many others. Layout styles define properties associated with the layout of content portions, such as 'starts on a new page', 'keep together' and margin offsets.

Changing the definition of a style can change the appearance of the content portions when they are presented or laid out according to that style.

7.1.7 Resource Link The *resource link* defines a link to information external to the infoplex, for example this may refer to shared information such as a company logo or standard text to be included by reference.

7.1.8 Annotation The *annotation part* mirrors the practice of writing remarks in the margins of paper documents. Annotations can be of any type of content, such as written, spoken or graphic. Annotations may be associated with any of the structures or content portions of an infoplex without modification to other structure or content descriptions. Accordingly, annotations have their own structure and content. In an infoplex there may be none, one, or many annotation parts, these may be used, for example, to identify annotations by different authors.

7.1.9 Content Portion The *content portion* defines an individual piece of content of an infoplex. The content portion may be associated with any of the infoplex structures or with annotations. A number of different types of content information and their method of encoding are defined below. Each content portion has just one type of content information. The different content information types and encodings used are specified in the profile to avoid the need to read the whole infoplex before processing it.

Content types are either general or application specific. The general content types include:

- character text,
- raster graphic (image),
- geometric graphic (or vector graphics),
- sound,
- moving image,
- computational data.

These general content types provide support for application specific content types which include:

- business graphics,
- spreadsheets,
- tables,
- mathematical formulae,
- chemical formulae,
- music notations.

7.1.10 Character Text *Character text* is held as a formatted or unformatted string, depending upon whether or not the infoplex has been formatted. Any internationally registered character set is permitted which includes ISO 646, ISO 8859 and ISO 6937.

7.1.11 Raster Graphic (Image) *Image (or raster) graphic* defines a picture in terms of a rectangular matrix of points, with independent definitions of the colour and darkness of each point, given as digital values.

The primary encodings defined in this area of application are Group 3 facsimile (CCITT T.4) and Group 4 facsimile (CCITT T.6). A higher level of facility will be provided by the ISO standard for photographic information ("JPEG") which is still under development. The JPEG standard will provide graphing of sufficiently high resolution and quality for accurate representation of images such as human chest X-rays. A number of de facto standards are in use with varying degrees of support, which include TIFF (Tagged Image File Format), and PCX.

7.1.12 Geometric Graphic (or Vector Graphic) *Vector (or geometric) graphic* defines a picture in terms of straight line segments, arcs, filled polygons, filled arc segments, "points" of various shapes, and individual characters of text. Each such element can have attributes of foreground and background colour, for example dash pattern, fill pattern and style.

The primary encoding in this area of application is that of the Computer Graphics Metafile (CGM, ISO 8632).

7.1.13 Sound *Audio* defines recorded voice or music that can be played back during infoplex presentation.

A number of standards exist or are emerging; one of these is ADPCM (Adaptive Differential Pulse Code Modulation).

7.1.14 Moving Image *Moving image* defines digitised video sequences.

Standards for inclusion of digital moving image are still a few years away. ISO is developing such a standard in its Motion Picture Expert Group ("MPEG").

7.1.15 Computational Data *Data* defines application-specific data included in an infoplex for use during processing by particular, identified applications.

7.1.16 Business Graphic *Business graphic* defines the automatic generation of pie charts, histograms, and other graphs from numerical information held in other content portions in the infoplex (for example within a spreadsheet). International standards for business graphics are not yet established.

7.1.17 Spreadsheet A *spreadsheet* defines a rectangular grid of cells, each of which may hold a numerical or textual value, or a formula. The use of formulae allows the displayed value in a cell to be calculated automatically from the values of other cells and from external information (for example today's date, or current page number). International standards for spreadsheets are not yet established.

7.1.18 Other Application Specific Content Types The items described above are those in which work towards open information standards are at an advanced stage. At a less advanced stage, but also important are standards for *tables*, *mathematical formulae* and *musical notation*.

7.2 Document Model

The infoplex model can represent information in a richer form than conventional paper or electronic word processed documents. For this reason the multimedia architecture defines a document model as a specialisation of the infoplex model. The document model, illustrated in Figure 4, below, is intended to introduce the document in its commonly accepted form and relate it to the infoplex. This document definition is adequate to represent electronic documents adhering to either the ODA or SGML family of standards.

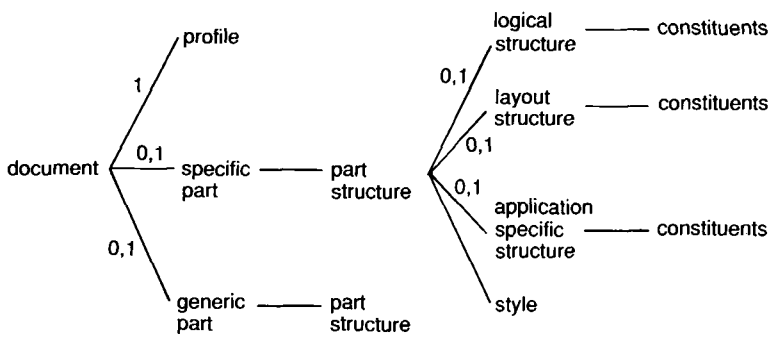


Fig. 4 Document Model

A document may have zero or one generic and specific parts and must have at least one of the structures (logical, layout or application specific); a combination of structures is permitted, but not more than one of the same type of structure. This document model is not suitable for representation of temporal content types (audio or motion image) nor for representation of hyperstructures.

References

A very large number of references could be provided, but material not publically available or not yet published is excluded. The following are considered to be the most important published documents.

Information technology – Open Document Architecture (ODA) and interchange format. ISO 8613, 1992.

Information processing systems – Computer graphics – metafile for the storage and transfer of picture description information. ISO 8632, 1987.

Information processing – Text and office systems – Standard Generalized Markup Language (SGML). ISO 8879, 1986.

Initial Graphics Exchange Specification (IGES) Version 4.0, NBSIR 88–3813, US Department of Commerce, National Bureau of Standards, Gaithersberg, Maryland. NBS, 1988.

The Corporation of the 1990's, Information Technology and Organisational Transformation, Oxford University Press (NY) ISBN 0–19–506358–9. SCOTT-MORTON, M. (Editor) 1991.

Computer-aided Acquisition and Logistic Support (CALS) – Program Implementation Guide. MIL-HDBK-59. US DoD, 1990.

Also referred to was: BRUNT, R.F., An Introduction to OPEN*framework ICL Tech. J.*, Vol. 8 no. 3 pp. 351–364, 1993.

Biographies

Ian R. Campbell-Grant

Ian Campbell-Grant is an ICL Fellow, with responsibility for representing ICL at the highest technical level and for operating as an external technical authority. He has been with ICL for 24 years, and since 1981 has been instrumental in the development of application level standards for interconnection of ICL office products.

His current role is focused on OPEN*framework*, where he is Company Architect for Multimedia, responsible for ensuring that the ICL portfolio relates appropriately to multimedia technology, so as to meet business needs most efficiently. He also played a major part in developing equivalent international open system standards. He led the ECMA group that produced the first version of the "Open Document Architecture" (ODA) standard, and now chairs the ECMA Technical Committee on document architecture and interchange.

He is General Editor for the ISO/CCITT ODA standard and chairs ISO/CCITT groups responsible for ODA development and design directions. He is also heavily involved with the emerging profiles for ODA use in Europe and world-wide. He is chairman of the Architecture Council of the European PODA Project, and also chairs the X/OPEN Working Group dealing with Document Interchange. He holds a bachelors degree in mathematics and two masters degrees, one in computer science

from the Massachusetts Institute of Technology (MIT) and the other in mathematical statistics. In addition he holds a degree of electrical engineer from MIT.

C. R. Smethurst

Roy Smethurst graduated in mathematics from Nottingham University and has had 28 years experience in the computer industry. This included writing compilers for COBOL, Fortran and PL/1 for System 4 and 2900 systems, acting as designer/strategist for the VME SCL (system control language), and its job management, spooling and file transfer functions.

He has been the ICL representative on FTAM on BSI, ISO, SPAG and EWOS expert groups and system strategist for Mainframe Systems for corporate office systems, image systems, and systems management (change control using process support systems). He is now ICL company architect for Availability in ICL's OPEN*framework* team, and is involved in OPEN*framework* specialisations for multimedia and CALS.

VME-X: Making VME open

Paul Coates
Computer Systems Division, ICL, Manchester, UK

Abstract

VME-X is a product running on ICL Series 39 servers, which supports multiple, multi-user UNIX-like environments, alongside the normal workload of a Series 39 machine. It conforms to X/Open standard XPG4 base version 1. The paper includes a brief functional description of VME-X, followed by a description of its structure and implementation, which, because it runs as an application on top of a general-purpose operating system, are rather different from a normal UNIX port.

1 Introduction

1.1 The Move to an Open VME

VME is the operating system designed and developed by ICL for its mainframe computer systems. It is now the operating system for the series 39 range of computers. VME was designed as a general purpose system, with special emphasis on support of large-scale On-line Transaction Processing (OLTP) applications. A brief outline of its main architectural features appears in Section 3.1.

Over the last decade ICL has been remoulding VME from a proprietary system – one whose programming interfaces and networking protocols were defined by the vendor and could be changed at the vendor's sole discretion – into an open system supporting standard, non-proprietary interfaces and protocols. This program, known as *Open VME*, is one of a number of strands of development within ICL, which have come together within the OPEN*framework*. The main benefits of open systems, for applications that work to the standards, are:

- Application portability. The task of porting an application from one open system to another should simply consist of recompiling the same source code.

- Interoperability. Components of a distributed application running on open systems should still work together if some are moved to different open systems.

The transition to open networking protocols is largely complete, and this paper is concerned only with application portability.

The body to which ICL looks for a definition of open interfaces is the *X/Open Co. Ltd.*, an association of computer vendors, software vendors and users, devoted to the promotion of open systems. At approximately three-yearly intervals, X/Open produces *X/Open Portability Guides* (XPGs), which recommend standards that computer vendors should support. The most recent are XPG3 (1988) and XPG4 (1992). Starting from its objectives of application portability and interworking, X/Open aims to identify all areas requiring standardisation, and to recommend a coherent set of standards, covering as many of these areas as possible. It adopts officially ratified standards where they exist, and *de facto* standards in other areas. X/Open also provides an independent certification service for computer systems. X/Open *branding*, as it is known, is a guarantee of openness widely recognised in the industry. The X/Open interfaces resemble those of UNIX systems much more closely than those of any other established systems, but a system fundamentally unlike UNIX can still achieve X/Open branding, as this paper will show. For more information on X/Open see (Taylor, 1991).

1.2 VME-X

VME-X, first released in 1991, is a software product that runs on VME and supports standard programming interfaces as defined by X/Open. Its main aim was to obtain X/Open base branding, for which it had to support:

- the C language
- a standard set of system interfaces and headers for C programs
- a shell equivalent to the Bourne shell found on UNIX systems
- a standard set of shell commands and utilities.

Other important objectives of VME-X were:

- Its development was to take at most 2 years.
- It should as far as possible be a practical porting platform, not only for the relatively small number of applications written strictly to X/Open standards, but also for other software designed to run on UNIX systems. To this end, VME-X also supports about 60 system interfaces and about 80 shell commands which, while not mandated by X/Open, are commonly found on other open systems.
- Applications ported from UNIX should look and feel the same on VME-X as on their original platform.

- The skill and effort needed for day-to-day administration of VME-X should be low, so as not to deter existing VME customers with no open systems expertise.

There was also a strong requirement for a degree of harmonisation between VME-X and the rest of VME. VME dates from the early 1970's, before the appearance of any official standards for operating system interfaces. It has its own 'native' interfaces, proprietary to ICL and completely different from those of any other system. The majority of applications available on VME predate VME-X and depend wholly on the VME native interfaces. The main aims in this area were:

- It should be easy for a customer to progress gradually from native applications – those using the VME native interfaces – towards open applications.
- Native and open applications should be able to run on the same system at the same time, and there should be scope for interworking between them.
- Performance of an open application should be comparable with that of a native application doing the same job.
- VME's particular strengths, such as disaster tolerance and data security, should be offered for the system as a whole, no matter whether the applications being run were open or native.

Portability of Transaction Processing applications is one area which is *not* within VME-X's scope. A separate project within the Open VME program, running in approximately the same timescales, is dedicated to this topic.

In the rest of this paper, Section 2 describes VME-X from the user's standpoint. Section 3 describes the implementation; and a rudimentary knowledge of the structure of UNIX systems is assumed here.

2 Using VME-X

2.1 Using VME-X from a terminal

The user starts by logging into VME. He is asked to give a personal identification and password and to state the *service* required. In general, the system will offer a number of services, each providing a different working environment. Some may provide a traditional shell or command language environment suitable for the computer-literate, while others may be tailored to particular applications or particular categories of user. VME keeps an access control list for each service, determining who may log into it.

Of the services offered, some will be native services, and others will be VME-X services. If the user chooses a VME-X service, the next thing seen is the first message from the user's initial shell. If this is the X/Open standard shell, the first message would normally be

```
$
```

inviting the user to type a shell command.

Thereafter, the session looks and feels just like a traditional UNIX session. All the normal shell facilities, such as pipes and redirections, are supported. As on most UNIX systems, an initial program can be defined for each user. Some users could be assigned an alternative shell, such as the C shell or a specialised application like mailx.

Native VME commands can be invoked, and they can be used in pipelines just like any other command. The user can also start nested sessions by invoking another service, which could be a native or a VME-X service, but in this case the output cannot be redirected or piped.

Where the system offers several VME-X services, each one resembles a multi-user UNIX system in its own right, and behaves as a separate open system. Generally, each has its own independent X/Open-compliant filestore, distinct from the native VME filestore. If an organisation has several departments each wanting to install and administer its own open system, this can be achieved by allowing each department its own VME-X service; the initial cost and continuing maintenance costs will be much less than if each had bought its own UNIX computer, as the hardware and much of the software administration can be handled automatically by the underlying VME system.

Different services can be protected from each other using VME's extensive security facilities (Parker, 1989). However, where a number of services with compatible security classifications need to share data, a file system can be made available on all of them simultaneously, provided at most one service has write access to it.

Native VME files can be accessed, using the syntax

/dev/vme/*vme-filename*

The file is assumed to be a VME serial character file in the usual VME format and character encoding. A binary view of the same file could have been taken by using /dev/rvme instead of /dev/vme. The user's access rights to a native VME file are evaluated just as if he had accessed the file from a native VME service.

Each VME-X service recognises a subset of the people known to the VME system, and only recognised users are allowed to log into the service. Where a user is recognised by several services, he can be logged into one service and access files on another. The syntax is:

/dev/vme/*other-service*!/usr/lib/...
or /dev/vme/*other-service*!my_directory/my_file

In the latter example, the user is accessing my_directory/my_file in his home directory on the other service. This method also allows access to FIFOs (named pipes) on the other service.

VME-X supports de facto terminal standards such as vt100 and vt220. However, the most prevalent terminal standard on VME systems is the ICL 7561 standard, which is obligatory for many native applications. An ICL 7561 terminal (or emulator of that terminal standard) stores keystrokes locally to form a block of text which is only sent to the mainframe when the user presses a special data forwarding key. The average overhead per keystroke in the mainframe is very low, and this is the secret of VME's ability to support tens of thousands of terminals at a time. However, applications cannot monitor every keystroke as they can on standard terminals; because the system does not react to a key depression (other than by echoing it on the screen) until perhaps a hundred keystrokes later, the application writer cannot offer a really high-quality user interface.

VME-X is supported on ICL 7561 terminals, albeit without the true UNIX look and feel, and the vast majority of the commands and utilities work without serious degradation of image. Only one standard command – the venerable UNIX text editor vi – was completely unusable, and a separate screen editor is supplied for ICL 7561 terminal users.

2.2 Application programming and porting

Experience at present is limited to applications written in C. The C compilation system resembles the one issued by Unix System Laboratories (USL). Pre-processing, compilation, assembly and linking can be carried out independently. The usual symbolic debugger and object code file analysis tools are provided. In addition to the usual cc command line options, the user has the following choices:

- Dialect of C. This may be Kernighan/Ritchie C, or ISO C (Kernighan et al, 1978 and 1988). A transitional option is also available, whereby either type of source is accepted, and statements legal in both dialects are interpreted according to the ISO standard.
- Arithmetic type mapping. This may be *architectural*, where longs are 64 bits and shorts are 32 bits long, or *alternative*, where longs are 32 bits and shorts are 16 bits long. Ints are always 32 bits long.
- Arithmetic Overflow and Underflow checks can be enabled or suppressed.

Because the series 39 hardware supports 64-bit arithmetic but not 16-bit arithmetic, the architectural mapping results in more efficient code than the alternative mapping. Some applications can exploit the architectural mapping to work with higher limits than are possible on most UNIX systems. There are a few other advantages, such as the clock time resolution which is in microseconds, as opposed to tenths of a second.

To be portable, a C application should not assume any particular size for arithmetic types, but experience suggests that the majority of applications do make such assumptions and will only work with the alternative mapping. Even applications which do not rely on the exact number of bits in a short or a long are often marred by assumptions about relative sizes (for instance, that two shorts can be packed into an int or that an extern declaration for a function returning a long can be omitted).

The X/Open System Interfaces and Headers are now supported to the XPG4 standard. Compared with the System V Interface Definition (SVID) issue 3, on which System V.4 is based, VME-X includes all but 25 of the Basic Operating System interfaces and all but 14 of the Basic Library interfaces. Most of these missing interfaces are concerned with symbolic links, and will be included when these are supported in the near future.

To enable applications to be independent of particular terminal types, a *curses* library and *terminfo* database are supported, just as on UNIX System V systems. A separate curses library is provided for ICL 7561 terminals (requiring a separate executable), or they can be driven in a more limited fashion using the ordinary curses library. Some applications cannot ergonomically be driven from ICL 7561 terminals.

Code running in a VME-X environment cannot call native VME system interfaces directly. The recommended technique for making a set of related VME functions accessible is to produce a *device driver*, so that the VME-X application can use a special file and make stylised calls on standard interfaces such as write() and ioctl().

Applications can access native files using the /dev/vme, /dev/vme8 or /dev/rvme syntax outlined in the Section 2.1. Character files in VME-X filestore are conventionally held in ASCII, as on most UNIX systems. In native VME filestore, serial text files are held in a variant of EBCDIC and with special record headers instead of the newline character customary in UNIX. When /dev/vme or /dev/vme8 is used, files are converted automatically between the two formats, and the VME-X application sees the file as ASCII characters (ISO-8859/1 characters if /dev/vme8 was used), just as if it were in VME-X filestore.

Native VME commands can be issued, and their input and output directed to VME-X files.

2.3 Interworking with other systems

Like most UNIX systems, VME-X supports the '*uucp*' suite (*uucp* is the UNIX command transferring files between terminals), FTP and TELNET, allowing users to transfer files between open systems or login from one system to another. They can be used from UNIX systems to access a VME-

X service, or vice versa. They can also be used between two VME-X services on the same system, although other methods are usually more efficient.

A VME-X service can act as a file server on a network of UNIX and DOS systems. This means that the client system (running UNIX or DOS) can 'mount' part of the VME-X service's filestore, so that it looks to the user of the client system like an integral part of the local filestore. A series 39, has powerful I/O hardware, perhaps more than 500 Gb of disc filestore, and automatic file security services; it can thus provide a secure and easily accessible home for far more data than most client systems could maintain themselves.

This facility is based on NFS (Network File System), and the client system needs the NFS client product (PC-NFS for DOS systems) to take advantage of it.

VME-X's future will increasingly be as a platform for the back-ends of distributed applications. In most such applications today, the programming interface used by one component to communicate with another is a *transport* interface – an interface where each side sees the route to the other as a two-way-simultaneous binary-transparent data channel. VME-X supports the *Internet* protocols, TCP/IP and UDP/IP, driven via the XPG3 *X/Open Transport Interface* (XTI) or the *Sockets* interface; the OSI connection-oriented transport protocol can also be driven via the XPG3 XTI. The XTI can also be used to communicate with native VME services or other VME-X services on the same system. This is often more efficient than accessing a FIFO as described in Section 2.1.

With Internet protocols, a series 39 with VME-X will be a multihomed host, with each VME-X service having its own Internet address. This ensures that applications with built-in port numbers can operate independently on each service.

Standards are now emerging for communications interfaces at a higher level than the transport level, and it is expected that distributed applications will increasingly use these instead of transport interfaces. They will be an important focus for future VME-X development. Among the infrastructures being planned is an X/Windows client facility, which will allow users to drive VME-X using a modern graphical operating environment.

2.4 Administration

Each VME-X service can have a separate administrator, so different departments within an organisation can each have their own service. The administrator is provided with a menu-driven package for routine tasks, including the recording of users, the allocation of users to file systems, daemon control, setting of service parameters, and control of the uucp suite and NFS. The administrator's job is simpler than the equivalent job on most UNIX systems,

as it is only concerned with a software service, not a physical machine. For this reason the sysadm package found on many UNIX systems is unsuitable for VME-X.

Some administrative tasks have to be carried out by the VME system manager. This includes the allocation of disc space for file systems, the management of Internet addresses and OSI network and transport addresses, installing VME-X releases, and rationing the amount of processing resource each service may use. None of the tasks expected of the VME system manager requires any expertise in open systems.

The automatic file security systems on the VME system can be used to maintain backups. With these systems, the backup tapes are controlled solely by the VME system manager, and users never need to know about them, even when a file is being retrieved. The VME-X service administrator periodically creates disc archives containing files needing to be secured, and the VME system automatically secures them. If preferred, an entire file system can be backed up, and this is particularly useful for the root file system. If a VME-X root file system is destroyed, the administrator need only log into a native VME service, then a single VME command will restore it from the latest copy, which can then be used immediately.

If the system is part of a network containing UNIX machines, VME-X can arrange to manage backups for them too. Again, neither the VME-X nor the UNIX system administrators need be concerned with tapes.

2.5 Access from native VME services

If a user or application running under a native VME service is in a context where a native VME command could be issued, they are also permitted to issue VME-X commands on VME-X services where the user is recognised. VME-X commands issued in this way are input to whichever program is registered as the user's initial shell. Input and output can easily be directed into native VME files.

Similarly, a native user or application can specify a VME-X file in any context where a native VME file could have been specified. The syntax (as the reader probably guessed from Section 2.1) is

vmex-service!/usr/lib/...
or *vmex-service*!my_directory/my_file

In the latter example, the user is accessing my_directory/my_file in his home directory. As before, this method also allows access to FIFOs (named pipes).

Normally the file will be assumed to contain ASCII character data and it will be automatically presented to the native application in ICL EBCDIC record format. However, if specially requested, a binary view of the file will

be given. Files containing character data conforming to the ISO8859/1 or ISO6937 standards can also be catered for.

3 VME-X implementation

To develop an open system, one must generally implement new language compilers, a new kernel, and new system administration software. Once a C compiler and kernel are available, the Unix System V versions of the C system interfaces, headers, the shell, and the commands and utilities can be ported from source code obtainable under licence from Unix System Laboratories Inc. (USL). An open system does not have to rely on USL code, but almost all do to some extent. The advantages of using USL source are lower development costs and conformance to any unwritten traditions that applications ported from UNIX may rely on. USL also provides source code for compilers and the kernel, but this can only be used if supplemented by specially written architecture-dependent modules.

VME-X uses mainly the USL versions of the C system interfaces, headers, shell, commands, and utilities, and the C compiler is partly based on USL code. Much additional non-kernel code was written specially for VME-X, for example:

- the service administration package
- code to handle the logging in of users, which is different from the UNIX login mechanism
- code to handle ICL 7561 terminals
- some replacements for USL library code to improve performance (for example by using series 39 string-handling instructions).

The VME-X kernel contains no USL code, though certain parts resemble it in principle.

Where the USL code was used, few significant changes to it were needed. The architectural arithmetic mappings with 64-bit longs and 32-bit shorts were used, and caused hardly any problems.

Using USL code inevitably makes the system resemble System V in many respects not mandated by X/Open. Files used by the system or by standard utilities are the same as on System V. For example, users are recorded in /etc/passwd, and uucp control files are in /etc/uucp. Object code is held in COFF format, the same format as on most System V systems; this allows VME-X to provide the usual UNIX debugging tools such as nm and sdb, and they work in exactly the way that UNIX programmers expect. Moreover, the file hierarchy in the issued system resembles that of System V.4, though there are differences; for instance there is no /stand, because the equivalent code is held as a VME executable file, not as part of the VME-X filestore.

The most interesting challenge was the design of the VME-X kernel. Before describing this, however, a brief digression is needed to describe the underlying platform – VME and series 39.

3.1 Series 39 and VME

3.1.1 Architecture. A *Virtual Machine* or VM consists of:

- a number of processes, each with an associated stack.
- up to about 8000 private address spaces each of potentially variable size, called *local segments*

A stack is a particular case of a local segment. The system as a whole has up to about 8000 further segments each of which may be *global* (shared between selected VMs) or *public* (shared by all VMs).

A store location is identified by its *virtual address*, consisting of a segment number and a byte displacement relative to the start of the segment. A global segment may have a different segment number in each virtual machine. A public segment has the same segment number in all VMs.

Each segment is divided into 1024-byte *pages*, and each page is held either in main store, or on a disc backing store. The units in which the operating system discards and fetches may range from a whole VM down to a minimum of two consecutive pages.

A process's *access level* is measure of its trustworthiness, and takes the form of a number between 1 and 15. 1 is the most trusted level. Each segment has a *read access key*, the maximum access level at which a process is allowed to read it. *Write access keys* are defined similarly. Access levels 1–5 are reserved for the operating system, access levels 6–9 are for privileged system software, and access levels 10–15 are for application software. These controls do not apply to the current process's stack segment.

A process changes its access level by executing a *system call* instruction or as the result of an *interrupt event*. The system call mechanism may be used for calls between more privileged and less privileged application code, or from application code to the operating system. Architecturally there is no distinction. The system maintains tables of all possible system call instructions; some are public, others local to particular VMs. For each possible system call instruction, there is defined:

- the virtual address of a code procedure,
- a target access level **t**,
- a system call access level **s**.

It is legal for a process running at access level **p** to execute a system call if and only if $\mathbf{p} \leq \mathbf{s}$. It transfers control to the code procedure and changes the

access level to **t**. On exit from the code procedure, the access level reverts to **p**. Some system calls are designated as *stack-switching*; with these, the process switches to a separate stack segment on entry, and reverts to the previous stack on exit. All system calls with $\mathbf{t} > \mathbf{s}$ are stack-switching, most others are not.

Interrupt events resemble system calls, except that they never cause stack-switching and the code procedure is not in general entered immediately, unless $\mathbf{p} > \mathbf{t}$. Normally it is entered the next time an instruction is obeyed which lifts the process's access level above **t**. Interrupt events are the usual way of communicating from more privileged to less privileged software. For instance, they are used for all notifications from the operating system to applications, including program faults, timers, terminations of asynchronous actions, such as I/O transfers and inter-VM message processing.

Flag events are similar to interrupt events, but there is no code procedure; the receiver has to call the operating system to sense any occurrences.

The CPUs are scheduled to VMs pre-emptively. Each VM has an absolute priority number, and no VM can use a processor if a higher priority VM could use it. More than one VM may be executing (perhaps waiting) in the operating system at a time. Semaphores prevent concurrent execution, where this is necessary. (For details of series 39 multiprocessing see (Warboys, 1985).)

Processes within a VM are not scheduled by the operating system. Control is passed to another process when the application executes a *process-switching call* instruction. At most one process per VM can be executing at a time, even on multiprocessor systems.

I/O transfers to central devices (mainly discs, magnetic tapes and high-volume printers) are scheduled according to the VM's priority. There is no caching in VME. Data is transferred directly between the hardware and user-space buffers, which have to be locked in main store unless the entire VM is on backing store. On an unshared disc file, the I/O initiation thread through VME runs entirely in the user's process, on the same stack, and generally without any semaphore usage. I/O other than to central devices (for instance, across/comms links) goes via operating system buffers.

3.1.2 File and resource control. VME maintains a record of each *person* who uses the system. Each person may be a member of zero or more *usernames*. Thus persons and usernames in VME are analogous to users and groups respectively in UNIX.

Files are organised into hierarchies, one for each username. The non-terminal nodes are called *groups*, the terminal nodes are *files* and *libraries*. Each node has privacy and security attributes, including access control lists, defining which other usernames may access it (only usernames – the lists do not

discriminate between the individual persons belonging to the username). The terminal nodes of the tree have file descriptive data distinct from the file content, and this includes details of where the file is held. By contrast with UNIX, a file's physical location cannot be deduced from its hierarchic name. A single node may appear in several places in a hierarchy (or several hierarchies), like UNIX files with multiple links.

A *library* is a collection of files with a single set of attributes. These files do not count as nodes in the hierarchy in their own right. An executable code file always belongs to a library and may (a) be derived from many source modules, (b) have many entry points, and (c) reference entry points of other executables by name, which may reference further executables, and so on. The libraries in which the referenced objects are to be sought can be influenced at run-time. A set of base procedures is resident in public segments, but if a VM calls a procedure outside that base, the executable is loaded from filestore into one or more segments, normally local ones; any other executables referenced directly or indirectly from it are normally cascaded in at the same time. The code remains available for execution in the VM as often as desired.

Garbage collection is handled by *Block Structure*. A process, acting on behalf of the whole VM, can *begin* a block, and any code loaded is 'tied' to the most recently begun block. The code is discarded from its segments when the process *ends* the block, in a fashion reminiscent of the scoping of variables in a block-structured programming language. Blocks can be nested, but can only be ended in a first-in-last-out fashion. Block structure applies not just to loaded code but also to the release of open files and indeed any kind of operating system resource.

3.2 *Structure of VME-X VMs*

A VME-X service has a pool of VMs. One is allocated to each terminal user, one to each batch job, certain daemons (background programs performing some function on behalf of the service as a whole) have VMs of their own, as do many functions driven from other machines using application-to-application connections. There is also one extra VM called the *system server*, which differs considerably from the others and is described in Section 3.2.1. From VME's standpoint, all the other VMs are dedicated to running an application program called the S3 kernel, which (as far as VME can see) has a single entry point. It runs mainly at access level 10, partly at 9. In each of these VMs there is a separate instance of the S3 kernel's data, which is all local. The S3 kernel has two basic functions:

- it knows how to load an executable program in COFF format from VME-X filestore and enter it.
- it provides a set of entry points to which the program's kernel calls can be fixed up.

(Strictly speaking, the system interfaces which the program calls are all implemented as library routines, but these routines may issue kernel calls.) The S3 kernel's interfaces to the programs it loads are a subset of the SVID Basic Operating system interface.

Once initialised, the S3 kernel forks a new process. Except in the system server VM, this process executes either the user's initial program as specified in the /etc/passwd file (except for certain daemons, where /bin/sh, the standard X/Open shell, is always executed). This fork, like all forks, is implemented by creating a new process in the same VM. If a user's initial program was /bin/sh and he now issues a pipeline consisting of two commands, there will be at least five processes in the VM: two processes used by the S3 kernel to administer the VM, a process for the shell and a process for each command in the pipeline, plus any further processes that those commands may have generated using fork() system calls.

All application programs running on VME-X execute at access level 11. When they make calls on the kernel, these are translated into non-stack-switching system calls to the S3 kernel. The process's access level drops to 10 while executing in the S3 kernel, and perhaps lower still if the S3 kernel makes inward calls into the VME operating system. On exit from the S3 kernel, the access level reverts to 11. The read and write access keys of the S3 kernel's data are 10, so it cannot be corrupted or read by applications. The S3 kernel makes extensive use of interrupt events to detect I/O conditions, timer requests made by user programs, hardware-detected faults in user programs, and messages from the system server. The target access levels of all these events is 10, so the S3 kernel's interrupt procedures can get entered, no matter what the user-level code is doing. On the other hand, these events do not interrupt the S3 kernel itself.

From the description so far, it will seem that each VM is trying to act like a UNIX system in its own right. Of course many kernel functions operate on resources shared by several VMs. In many cases, the S3 kernel can achieve this by suitable exploitation of standard VME facilities. This is true, for instance, of all non-disc I/O, pipe handling, and most signal processing. Some functions which a UNIX kernel normally has to handle – for instance store paging and swapping – can be left entirely to VME.

In other cases, VME alone does not provide adequate coordination. In these cases, the S3 kernel passes the request to the S3 kernel instance in the system server VM using VME message-passing facilities. Since there is only one system server VM per service, inter-VM interference cannot occur. Many of the less critical, more complex kernel calls are processed in the system server, as are certain disc I/Os.

Excessive use of the system server can lead to bottlenecks and a drain on the CPU owing to the cost of message traffic. Part of the work of the VME-X project was to add to the VME operating system interface, so as

to avoid certain uses of the system server. For instance VME facilities were added to support inter-process communication(IPC) functions – shared memory, semaphores and message queues. The new VME interfaces conform to established VME standards and look nothing like UNIX interfaces. However, they do provide the essence of what the S3 kernel needs to support IPC without explicit interaction with other VMs.

Process scheduling takes place at two levels. Each VM is scheduled by VME according to its priority. Within each VME-X VM, whether user VM or system server, the S3 kernel schedules the processes. When a process issues a request to the system server, the S3 kernel will not allow it to run again until the system server replies. In the meantime it will schedule other processes in the VM if possible. In theory, a terminal user can have two VMs on two different processors, working simultaneously on his behalf. The S3 kernel also reschedules processes if the current process exceeds its timeslice or if it sleeps.

3.2.1 Differences between the system server VM and user VMs. The system server has an S3 kernel, similar to that of other VMs, but instead of a shell, it loads a special COFF program called the C kernel. Each message from another VM in the service is passed to the C kernel for action, and the S3 kernel issues a reply when the action is complete. The C kernel is modelled closely on parts of the System V kernel, although none of it is actually ported from USL.

The C kernel's interface to the S3 kernel is quite different from the SVID-like interface of other VMs. Processes are also handled differently; there is a pool of processes allocated on demand for each incoming message from a user VM. For instance, if a user VM issues a **read()** request, this may be passed to the system server, which allocates a process. Several physical transfers may be needed, and the process remains allocated until the last of these is complete. Each process in the system server has its own stack but, in contrast with user VMs, there is only one copy of the static data, just as in a System V kernel there is only one instance of most of its data items.

3.2.2 Filestore. The VME-X filestore is modelled closely on System V filestore. It contains up to 100 file systems per service, each implemented as a file in VME terms. Internally each file system looks just like a System V file system, except that the blocks are 2048 bytes long, but there is no code in the VME operating system that understands this structure. Thus, VME-X files are not recognised as files by the VME operating system; actions such as create, delete, open and close are not implemented by invoking the corresponding native VME functions, but are handled purely by the C and S3 kernels.

The VME system does recognise the special syntaxes for VME-X file names as described in Section 2.5, and this is what enables VME-X files to be

accessed from native environments. On detecting a VME-X file name in a context where a VME file name would be expected, the VME system routes the request by sending a message to the system server of the appropriate service. All subsequent actions on the file are routed in the same way.

Except during initialisation, the S3 kernel avoids exploiting VME block structure, both in the system server and in other VME-X VMs. VME files are closed explicitly when no longer needed. VME code modules are never unloaded, but this is harmless because VME-X application code has no facility for loading VME code in the VM. Where a VME-X user invokes native VME commands, these are always executed in a separate VM, which is not within the VME-X service.

3.2.3 Device Drivers. A System V kernel has standard internal interfaces known as the DDI/DKI whereby bodies outside USL can insert *Device Drivers*, subsystems designed to drive particular types of hardware devices. The generic kernel expects all drivers to present it with a common interface; this is the *Device Driver Interface* (DDI). The drivers themselves may use the facilities of the rest of the kernel by calling functions in the *Driver-Kernel Interface* (DKI).

The S3 kernel in VME-X also supports device drivers. Its device driver interface resembles the System V DDI, but is slightly different in form. Device drivers are compiled and constructed as ordinary VME programs, not as VME-X COFF files, and they have the entire VME application programming interface available to them. Each is a distinct executable file separate from the S3 kernel, and linked from it using VME loading mechanisms when the user logs in.

Adding a new device driver or a new version of a device driver is a painless process, not requiring a rebuild of any existing software or a closedown of the service. (Existing logged-in users continue with the old version, but new users can be made to pick up the new version as soon as it is installed.) Driver failures will at worst kill the VM, hence in general only one user will be affected.

3.2.4 Process creation. When implementing an open system on a platform not designed for that purpose, one inevitably runs the risk that some essential function cannot be made to work at all. With VME-X, no such impasses were hit, but one area – process creation – caused some interesting problems. The fork() function creates a new process, which is an exact copy of the calling process. Thus, if one process calls fork(), two processes exit from it. VME's local segments are address spaces private to a VM, but not private to a process. When the data areas are created for the new process, they therefore have different local segment numbers from those of the old process; hence any virtual addresses within those data areas would appear to need adjusting. This is an impossible demand on the S3 kernel, which

cannot know which items in the application's data are virtual addresses and which are not.

The only solution was not to have any virtual addresses in the data areas, and to represent addresses by displacements relative to the process's address space base. All language compilers have to observe this rule when generating code, and the S3 kernel has to ensure that a certain machine register points at a fixed position relative to the address space base of the current process. The upshot is that separate compilers are needed for native VME and VME-X, and executables for VME and VME-X are different. The incompatibility lies not just in the red tape (COFF as opposed to native VME's OMF) but in the nature of the executable code itself. Two executables, one native and the other on VME-X, cannot call each other using the series 39 architectural call instruction. This dichotomy is one we would have preferred to avoid.

A second problem presented by fork() was that it always creates the new process in the same VM as the old one. This is usually appropriate. However, where the new process is a daemon, it will probably disconnect from its parent's session (tty-group). It is then strange for it to execute in the same VM as that of the user who happened to start it. The attributes of the VM – such as its VME username, scheduling priority, and privilege level – may be inappropriate. For this reason, VME-X contains an extra system interface that combines the most of the functions of fork() and exec() and creates the new process in a new VM. There are some restrictions; for instance files open in the old process are not open in the new process.

3.3 An approach that was not adopted

VME-X relies crucially for its performance on several characteristics of VME. The versatile VME loader and the in-process system call mechanism allow code from different executables to interact by a simple hardware call instruction without the disruption of process-switching, and this is critical to VME-X.

However, it was at first thought that a much fuller use of the facilities in VME could have been made. For instance, it was suggested that files and directories be implemented as VME files and groups respectively, that exec() be implemented using the VME loader, that VME native object code formats be used instead of COFF and so on. The advantages of this approach were thought to be a lower implementation cost and a much more seamless integration between VME-X and VME. Furthermore it was thought by some that a design of this kind could lead to a Grand Unified System Interface, encompassing both the X/Open and the existing VME system interfaces.

This approach ran into difficulties. When selecting a native VME function on which to base the implementation of a UNIX kernel function, one often had the choice between (a) a VME function offering most but not all of the

capabilities required, but not easily generalised, and (b) a VME function far more complex and general than required. If the product was to be X/Open-compliant, the latter had to be selected, but the unwanted generality meant that for any given processor rating the VME function was slower than the original function on a typical UNIX system. Often the VME function would have parameters giving the system information allowing it to select between a simple efficient mode of working (exploited by 99% of native applications) and a slower and more general mode; and the latter had to be selected for VME-X because the X/Open interface had no parameter allowing the system to deduce whether the simpler efficient mode would have been acceptable. As most applications on VME-X will have been originally designed for UNIX, it was decided that this kind of kernel design would not give adequate performance.

Even if the performance problem had been overcome, a Grand Unified System Interface would have led to further difficulties. Such an interface would have been exceedingly hard to understand. Features like VME block structure, defined to act on resources of all kinds, would not have coexisted comfortably with X/Open functions. Asynchronous activities, like X/Open signals and VME interrupt events, would have interfered with synchronous activities and with each other, probably in a very confusing manner.

Finally, it was believed by many that customers wanting to develop portable applications would have more confidence in VME-X if it did *not* have proprietary extensions intimately bound in with the open environment. It was felt that, while we certainly wanted powerful interworking facilities, there should nevertheless be a clear boundary between the native and open environments. It was important to aim at the *right* degree of integration between the two, close enough to permit effective interworking but distant enough to avoid unwanted interference.

4 Summary

4.1 Open-ness

VME-X has undoubtedly made VME into a truly open system, complying with both *de facto* and formal standards. Expert UNIX users can rarely tell that VME-X is not UNIX. Porting of C applications from UNIX to VME-X is now considered no more difficult than porting from one UNIX system to another.

VME-X was awarded XPG3 base branding by X/Open in May 1991. It was the first time that this branding had been achieved by a system originating from the mainframe world. After further development, XPG4 base version 1 branding followed in October 1992. No other system, not even a UNIX system, achieved this goal before VME-X.

VME-X reached the market in late 1991 as an optional extra to VME. From 1993, it will become a standard component of all Open VME systems.

4.2 Performance

VME-X performance is broadly similar to that of a UNIX system on a processor of similar power. Kernel functions implemented without recourse to the system server VM – including pipes, non-disc I/O, and some disc I/O – are less mill-hungry than one would expect on UNIX. Against this, there are one or two areas where over-reliance on the system server VM has led to excessive mill usage. Work in progress to correct these will come to fruition in the near future.

4.3 Integration with native VME

The degree of integration between VME-X and native VME usually exceeds its users' expectations, especially at the command line level. For application-to-application interworking, connections can be made using the X/Open Transport Interface, or, if a suitable device driver is developed, a VME-X application can drive a VME application in the same VM. The facilities will be improved as higher level interfaces are supported in the future (see section 2.3).

5 History

Implementing UNIX-like systems on top of other operating systems is not new. An early project is described in (Lycklama, 1978), and there have been others, notably on the VM operating system on IBM-compatible mainframes.

VME-X was not even the first port of a UNIX-like system to VME. In 1984–1986, a port of UNIX System V.2 was carried out. The resulting product, known as VNS, was never marketed, but it was on the whole a viable platform for porting applications from the UNIX world. VNS was all the more remarkable because it was developed at arm's length from VME. With one exception, no changes in VME were allowed. This was the underlying cause of VNS's drawbacks. The VME-X team was able to examine VNS's strengths and weaknesses in detail.

The existence of VNS also obviated countless bootstrapping problems – such as how to create the first ever COFF-producing C compiler or read in the first ever UNIX tape.

6 Acknowledgements

The VME-X project is a collaboration between ICL and Industry Standard Software Ltd., with approximately equal contributions from each company. Several members of staff at Industry Standard Software have expertise in

both UNIX and VME, having worked on VNS in the 1980's, and are therefore uniquely well qualified to contribute to the project.

Thanks are due to Mick Meaden, head of the Industry Standard Software team, and Nic Holt and Mike Kay of ICL for helpful comments on an early draft of this paper.

Parts of VNS were written by members of staff at Praxis Systems Plc, working under contract to ICL. Modified versions of some of their code survive in VME-X.

References

LYCKLAMA, H., The MERT operating system, *Bell System Tech. J.* Vol. 57(2) 1978.

WARBOYS, B.C., VME Nodal Architecture: a model for the realisation of a distributed systems concept, *ICL Tech. J.* Vol. 4(3) 1985.

PARKER, T., The VME High Security Option, *ICL Tech. J.* Vol. 6(4) 1989.

TAYLOR, C.B., X/Open – from Strength to Strength, *ICL Tech. J.* Vol. 7(3) 1991.

KERNIGHAN, B.W. and RITCHIE, D.M., *The C Programming Language*, Prentice Hall 1978 and 1988 (1978 edition describes Kernighan/Ritchie C, 1988 edition describes ISO C).

Biography

Paul Coates has worked for ICL since 1969, mainly in software development. He has worked in the VME-X team since its inception in 1989.

A New Approach to Cryptographic Facility Design

Jim Press
ICL Mid-Range Systems Division Reading, Berks, UK

Abstract

This paper introduces the principal concepts behind the design of the Cryptographic and Key Management Service (CKMS), which forms part of the Cryptographic Support Facility (CSF) available for use within other ICL products.

The CKMS is a new approach to the provision of cryptographic services based on Object-Oriented techniques. It allows client applications to be independent of the details of the underlying cryptographic algorithms, thus aiding their portability. It also ensures that cryptographic algorithms are used in conformance with the local security policy and has been designed to support easy algorithm replacement.

1 Introduction

Cryptography is an essential building block for providing security in an open distributed system. Cryptographic techniques can be used for the protection of the confidentiality and integrity of data in untrusted environments, and are used not only by applications for the protection of user data but also to prevent subversion of system security control information.

Cryptographic services are often provided to applications by allowing direct access to cryptographic algorithms and expecting the application writers to have some expertise in how to use them sensibly (i.e. in a secure manner). This also makes the applications heavily dependent upon specific algorithms and therefore not very portable.

A better approach is to provide a cryptographic facility to hide some of the complexity and to provide applications with a simpler interface. Unfortunately such facilities are usually designed around one or two specific algorithms, and this is reflected in their interfaces. For example, IBM's cryptographic facility (IBM, 1990) is designed around the US Data

Encryption Standard (DES), and its clients are required to have some knowledge about DES modes and parameters. It is not easy to change the algorithms used by such a cryptographic facility without changing the interface and thus affecting the client applications.

Within the Security Quality of ICL's OPEN*framework* (Fairthorne, 1991), the '*Cryptographic Support Facility*' (CSF) provides cryptographic and supporting services to its clients, which include other security facilities, applications and infrastructure components. The core of the CSF is the '*Cryptographic and Key Management Service*' (CKMS) which provides cryptographic operations and functions for the generation and management of cryptographic contexts (described later). The '*Key Distribution Service*' (KDS) augments the CKMS providing functions for the distribution of keys associated with cryptographic contexts.

The subject of this paper is the main design principles behind the implementation of the CKMS. These allow client applications, such as ICL Access Manager, to be independent of the details of the underlying cryptographic algorithms. Such algorithm-independence is very important for applications which wish to be portable. Portable applications which require cryptographic services cannot make assumptions about which algorithms are supported or even permitted (e.g. due to export controls or national security restrictions). For the most part, such applications just want a service to provide the operations to protect their data; they are not usually concerned with how these operations are achieved but only the quality of service provided.

The main aspects of the design addressed by this paper are the provision of services to client applications in an algorithm-independent manner, and keeping the CKMS itself independent of specific algorithms.

2 Object-Oriented Design

Object-Oriented Design is an approach to software design which models systems as collections of cooperating objects, treating individual objects as instances of a class within a hierarchy of classes.

An object has the following properties:

a) It has *state* encompassing properties and their current values.
b) It has *behaviour* defined by the 'services' it provides to its clients. Clients do not directly access the internal state of an object but send 'requests' to the object for services to be carried out to access or manipulate the object's internal state.
c) It has *identity* denoted by a name. To make a request, a client identifies the object which is to perform the service and names the request.
d) An object is an instance of some *class*. A class contains a common structure and a common behaviour applicable to all instances of the class. Classes can be derived from other classes (inheritance).

An easy to read introduction on the subject can be found in (Coad & Yourdon, 1991) and further reading can be found in (Booch, 1991).

3 Cryptographic Context Types and Instances

Within an Information Technology system, any activity takes place within a *Context* encompassing the process environment provided by the operating system and the supporting infrastructure. The '*Security Context*' is part of the Context containing security-related information. In turn, a Security Context may itself contain one or more '*Cryptographic Contexts*', which contain information relevant to the provision of cryptographic operations. This includes identifying an algorithm and the parameters needed by the algorithm (e.g. the key, the Initialisation Vector, the mode of operation, etc.).

Where two or more entities wish to communicate securely, the association between them will have one or more related Cryptographic Contexts. For example, there may be a Cryptographic Context related to the provision of confidentiality of data, and another related to the provision of data integrity.

Within the CKMS, a Cryptographic Context as used by a client is realised as a '*Cryptographic Context Instance*', or '*Context Instance*' for short. A Context Instance is private to a client, however entities which are communicating within a common Cryptographic Context will have compatible, but not necessarily identical, Context Instances. For example, Figure 1 illustrates two entities A and B using asymmetric techniques (Press, 1989) to protect data integrity, A's Context Instance contains A's private key whereas B's compatible Context Instance contains A's public key.

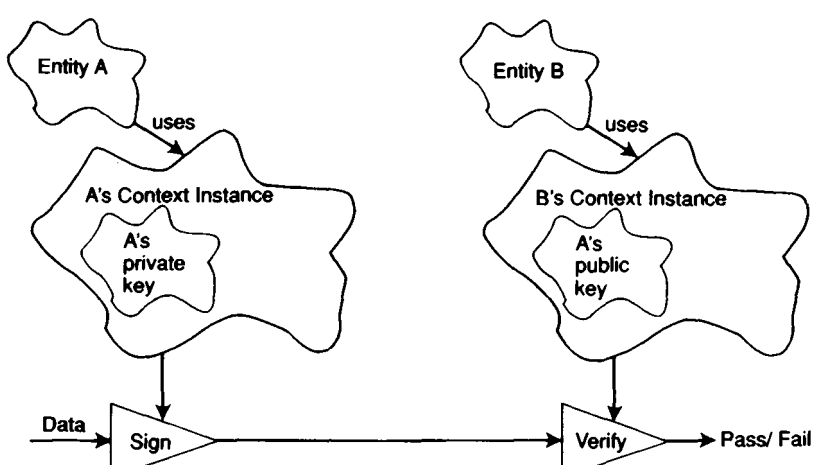


Fig. 1 Use of compatible but not identical Context Instances

A Context Instance can be viewed as an associative object between a client application and the CKMS, encapsulating the details of how the CKMS is going to provide the quality of service that the client wants.

To enable clients to request the creation of Context Instances in an algorithm-transparent manner, Context Instances are created from a '*Cryptographic Context Type*', or '*Context Type*' for short. A Context Type is a blueprint for Context Instances specifying how they should be constructed in order to provide a given quality of service to a client.

A Context Type can be viewed as an associative object between the administrator and the CKMS encapsulating the details of how the administrator wishes a particular algorithm to be used in order to satisfy a particular quality of service requested by a client. The Context Types effectively form a model of the local security policy regarding the use of cryptography in the customer's enterprise.

A client can request that a Context Instance be created, without needing knowledge about the underlying algorithms, by specifying the quality of service that it requires in terms of the functionality (e.g. data confidentiality or data integrity) and the level of protection needed (e.g. low, medium, high). The CKMS selects an appropriate Context Type which is used to construct a Context Instance, generating keys and other parameters as is necessary, and returns the reference to the client.

Having created a Context Instance, a client only needs to supply its reference and the data to be operated upon to a cryptographic operation supported by the CKMS. This means that the interface of CKMS cryptographic operations can be kept simple and free of any algorithm-specific information. The general form of a CKMS cryptographic operation is:

operation_name (context_ref, input_data, output_data, result)

This contrasts with other cryptographic facilities, such as IBM's (IBM, 1990), where algorithm-specific parameters must be supplied to every cryptographic operation.

Figure 2 illustrates the client application view of the CKMS. This shows a client application requesting a quality of service (1), which is serviced by the CKMS selecting an appropriate Context Type (2) to create a Context Instance (3), whose reference is returned to the client (4). The client can then make requests for cryptographic operations providing a reference to the Context Instance to be used and the data to be operated upon (a). The CKMS locates the referenced Context Instance (b) which guides it to the correct Cryptographic Algorithm. The Cryptographic Algorithm is then invoked (c) and uses the Context Instance (d) to perform the required operation.

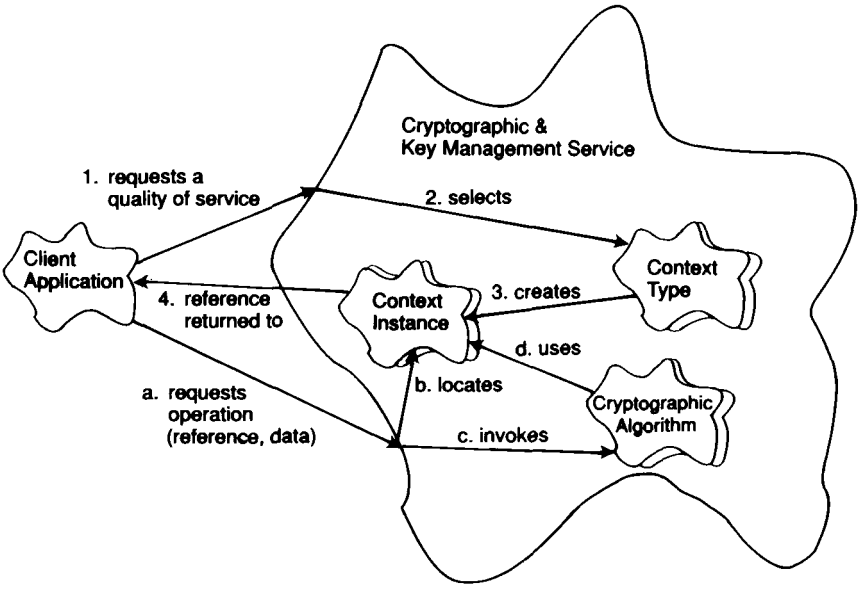


Fig. 2 Client view of the CKMS

Since Context Instances can only be created from Context Types, Context Types provide a strong control over how the cryptographic algorithms can be used. This is particularly important in situations where permission is granted to use strong algorithms subject to certain restrictions, such as limiting their use to integrity only or restricting the size of keys.

Context Instances can be exported for distribution purposes, copied (with restrictions on use) and created as persistent objects (stored securely in some form of long-term storage) or transient objects (short-term).

The CKMS also supports client applications which may be more fussy about the cryptographic mechanisms which are used, by allowing them to specify the desired algorithm, mode or key size, in addition to the required quality of service. Just because a client specifies a particular mechanism does not mean that it will be allowed to use it. The CKMS uses the mechanism specification as additional search criteria when selecting a suitable Context Type. This ensures that a client cannot circumvent the security policy, for example by specifying a strong algorithm to be used for confidentiality when the security policy says that it should only be used for data integrity.

4 Context Type Creation

Context Types and Instances provide a mechanism by which the CKMS can hide the details of the underlying algorithms from its clients. However, the security administrator, who sets up the Context Types to model the

security policy of the domain, should not be required to have cryptographic expertise. In this section we look at how the CKMS can allow the administrator to be algorithm-unaware also.

In accordance with the Object-Oriented philosophy, a Cryptographic Algorithm will know how it can be configured in order to achieve a given quality of service. For example, an algorithm may be able to provide several levels of strength by providing different modes of operation, or by allowing differing key sizes. A Cryptographic Algorithm will therefore be able to create a Context Type to encapsulate the information required to meet a requested quality of service. It will also be able to respond to queries regarding which qualities of service it is capable of supporting.

This approach means that the responsibility for the security of the resultant services are placed upon the algorithm designers/implementors.

An administrative application, run by the security administrator, is able to request the creation of a Context Type by simply specifying the quality of service that Context Instances created from it will satisfy. The CKMS locates an algorithm which can be used to meet the desired quality of service, and it is asked to create an appropriate Context Type.

Provision is also made for a more algorithm-aware administrator to specify an algorithm and a required mode or key size. The CKMS locates the specified algorithm, if it exists, and asks it whether it can meet the desired quality of service using the specified mode or key size. If the quality of service can be met, the algorithm is asked to create an appropriate Context Type.

Figure 3 illustrates an administrative application requesting that a Context Type be created for a quality of service and optionally for a specific algorithm & mode or key size (1). The CKMS selects a Cryptographic Algorithm (2) which can satisfy the requirement (3). The Cryptographic Algorithm then creates an appropriate Context Type (4), whose reference is returned to the administrative application (5).

Several Context Types can be created to satisfy a particular quality of service, this may be particularly the case where Context Types have to be created for inter-domain operations. Usually the administrator would select one as the default which is to be used when an algorithm-unaware client requests a Context Instance to be created for that quality of service.

5 The Family of Algorithms

This section looks at the concepts that allow the CKMS itself to be designed such that it minimises the dependencies upon specific algorithms. This exploits the Object-Oriented concepts of inheritance and polymorphism, explained in detail in (Booch, 1991).

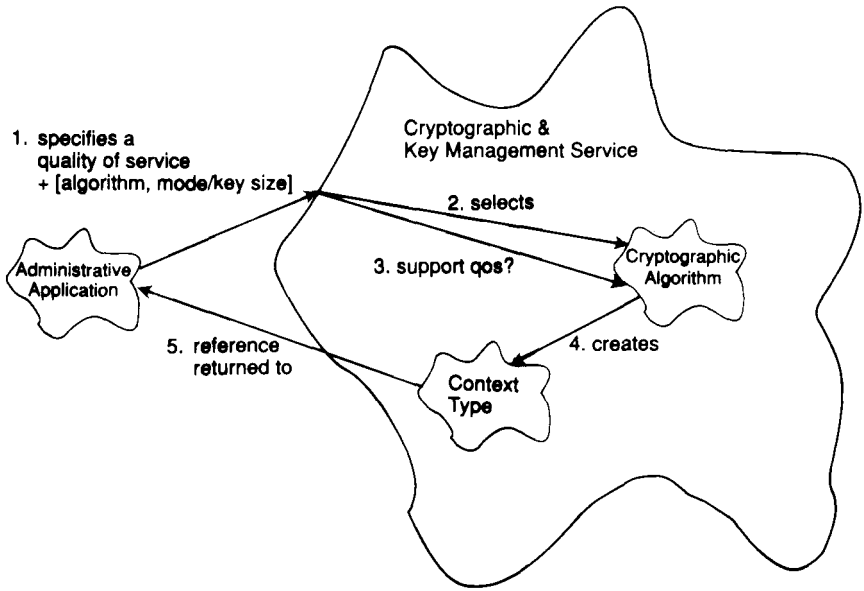


Fig. 3 Context Type creation

Inheritance is the concept that a class can be derived from a more general class, inheriting its attributes and services, whilst extending or modifying its implementation. For example, a general super-class 'Random Number Generator' could be specialised into a 'Finite Field Linear Shift Register' sub-class (a special type of random number generator based around a linear shift register describing a polynomial in a finite field).

Polymorphism is an ability which allows an instance of a sub-class to have different implementations of services of its super-class and yet still be treated as though it were a type of its super-class (i.e. specialisations can be treated using the generalisations from which they were derived). For example, a 'Finite Field Linear Shift Register' object can be treated as though it were a 'Random Number Generator'. This means that the CKMS does not need separate code to use each type of Random Number Generator, but can simply have a single piece of code which can use Random Number Generators in general.

Algorithms used within the CKMS are objects instantiated from a class derived from a more general super-class known to the CKMS. A class hierarchy is defined to encompass cryptographic algorithms, key generators and random number generators. When an Algorithm object is created, it will register its availability with the Supported Algorithm List, unless it is intended for another object's private use. The Supported Algorithm List maintains a list of Algorithm objects which are available for use within the CKMS.

Advantages over more traditional techniques include:

- Implementors of new algorithms can reuse code, through inheritance, of existing classes from the hierarchy. Only the new parts of an algorithm need to be evaluated for assurance if the super-classes have been previously evaluated;
- Objects within the CKMS only need knowledge of a few general super-classes in order to use an Algorithm object. An algorithm can be added to the CKMS by a single line of code to create an Algorithm object, without affecting the rest of the CKMS;
- The CKMS need have no knowledge of whether an Algorithm object implements a pure software algorithm or whether it is software interfacing to a hardware crypto device;
- Clients of an algorithm need no knowledge of whether a cryptographic algorithm is symmetric or asymmetric. This is hidden by encapsulation within an Algorithm object and within the Keys associated with Context Instances;

There are many different ways in which a taxonomy of algorithm classes could be designed. For example, cryptographic algorithms can be symmetric or asymmetric, stream or block ciphers, some may support integrity operations only, others may support one-way encipherment only. A comprehensive class hierarchy, where for example cryptographic algorithms are classified into those which support confidentiality, integrity and one-way encipherment respectively, was originally tried but was found to have many levels, was difficult for non-specialists to understand and led to a large use of multiple inheritance (classes inheriting from more than one super-class) and an increase of classes that were known about by the CKMS.

In designing class hierarchies, (Coad & Yourdon, 1991) advise that if most types of something have a particular attribute or service in common, then this characteristic is put into a generalisation class and specialisations which do not share this characteristic are treated as exceptions. For example, most cryptographic algorithms can be used for confidentiality (data encipherment and decipherment) and data integrity, so these services are placed into the general Cryptographic Algorithm class. Those cryptographic algorithms which do not share one or more of these services have to provide appropriate exception handling for the unsupported services. Classification along these lines led to the hierarchy shown in Figure 4. This was found to be easy to understand, easier to use and allows more generalised code to be written.

In Figure 4, the classes above the horizontal line are the general classes known to the CKMS, the classes below this line are exemplary.

The class 'Algorithm' contains attributes and services common to all algorithms. This is itself a sub-class of 'Managed Object' which is discussed later. The next level of classes derived from 'Algorithm' denote the three main types of algorithms which are dealt with within the CKMS:

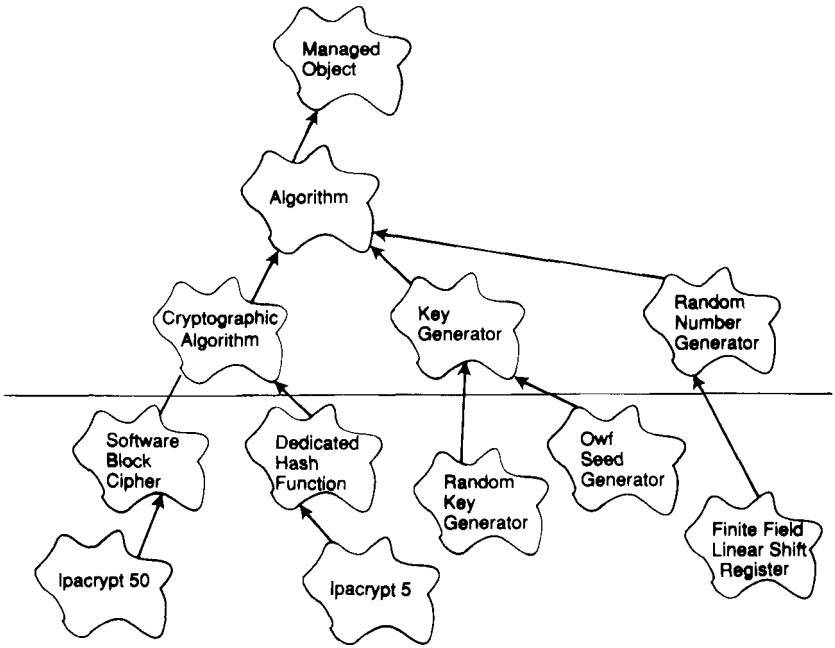


Fig. 4 Algorithm Class Hierarchy

- Cryptographic Algorithm
  This encapsulates the attributes and services common to cryptographic algorithms in general. This class defines the interfaces of several services whose implementation is deferred to a derived class: encipher data, decipher data, seal data and check seals;
- Key Generator
  This encapsulates the attributes and services common to algorithms which generate a key from a seed for use within a Context Instance. This class defines the interface of a deferred service to generate a key;
- Random Number Generator
  This encapsulates the attributes and services common to random number generators and defines interfaces of deferred services to generate a random number and to reset the generator;

Code within the CKMS is not concerned with specialisations beyond this level. Algorithm objects are instantiated from classes which are derived from the classes mentioned above. Implementors are free to add intermediary classes which may be useful, and may use multiple inheritance.

For example, in Figure 4, the class 'Software Block Cipher' provides an implementation for the Encipher, Decipher, Seal and CheckSeal services (deferred from the class Cryptographic Algorithm). The class defines the interface of two new services to encipher and decipher a fixed length block

of data, where the implementation is deferred to a further derived class. The class 'Software Block Cipher' effectively provides a general implementation of ISO standards to encipher and decipher data (ISO, 1991), and to generate and verify integrity seals (ISO, 1989). In order to use these implementations, a further sub-class (e.g. 'Ipacrypt50') must be defined to provide the implementations for the deferred services to encipher and decipher a block. This illustrates the property of code reuse, as any new cipher can be defined as a sub-class of 'Software Block Cipher' and this will inherit the ISO conformant services.

Another example in Figure 4 is the class 'Dedicated Hash Function', a sub-class of 'Cryptographic Algorithm', providing implementations to one-way encipher data and generate and verify integrity seals by invoking the encipher service of a One-Way Hash Function, such as 'Ipacrypt5', which is derived from this class. An implementation would be provided for the decipher service which if invoked would raise an error since one-way enciphered data cannot by definition be deciphered!.

6 Bit Strings

Algorithms view the data they process as a string of bits. In software environments, data is encoded into a character representation optimised for the underlying hardware of the host machine (e.g. 8, 9 or 16 bits). This would normally mean that an implementation of an algorithm has to be aware of how bits are encoded in the character representation, and thus different implementations would be required for different machine environments.

Data which is to be transferred to a different machine environment will normally conform to some agreed transfer format (e.g. 8 bit ASCII, ASN.1 Basic Encoding Rules). For example, data being transferred from a 9-bit character machine to a 8-bit character machine could be encoded as 8-bit ASCII, the most significant bit not being used on the 9-bit machine. If this data is to be cryptographically processed prior to transfer, the algorithm implementation would normally need to be aware of this so it doesn't encrypt the unused bits.

To address this problem, the CKMS uses objects of a class 'Bit String' to encapsulate data which is processed by algorithms. Services are provided by which algorithms can treat the data as a bit string (or an octet string) whilst hiding the actual character encoding.

Algorithms used within the CKMS are implemented to use the services of the Bit String class in order to process data. This enables the algorithms themselves to be portable to different machine environments.

7 Object Managers

Within most Object-Oriented systems there is usually the need for a number of "object managers", each responsible for managing the storage, location and retrieval of a particular class of object or a group of object classes.

The CKMS uses a general scheme where a specialised object manager inherits attributes and services from a general 'Object Manager' class, and each managed class inherits from a general 'Managed Object' class.

An Object Manager will contain zero or more (usually one) 'Persistent Store' objects encapsulating the details of how persistent objects are saved and retrieved in long-term storage. Specialised Persistent Store objects may exist to encapsulate different forms of persistent storage (e.g. storage on-board a crypto device). Two of the responsibilities of a managed object (inherited from the Managed Object class) are to be able to provide an Object Manager with a storable representation of the managed object, and to be able to restore the managed object from previously stored information provided to it by the Object Manager.

An Object Manager will also contain an Index used to locate persistent objects (either in persistent store or in memory) and transient objects.

Within the CKMS there are the following specialised object managers:

- Supported Algorithm List
  This maintains a list of Algorithm objects. Algorithm objects are transient, being dynamically created when the CKMS is created;
- Context Instance Manager
  This manages the storage, location and retrieval of persistent and transient Context Instances. The organisation and implementation is encapsulated by the object (e.g. whether secure hardware is used to hold unencrypted keys). The Context Instance Manager is responsible for ensuring that objects held in files are suitably protected (i.e. associated keys are encrypted) and for loading objects into memory when required.
- Context Type Manager
  This manages the storage, location and retrieval of Context Types (which are persistent objects) and responsible for loading objects into memory when required.

Figure 5 shows the Object Manager class, the classes it contains and the specialised object manager classes used in the CKMS.

8 Future Enhancements

The Context Types and Instances concept arose before the use of Object-Oriented techniques to solve the algorithm replacability problem. Further work is in progress to properly integrate these concepts in a more Object-

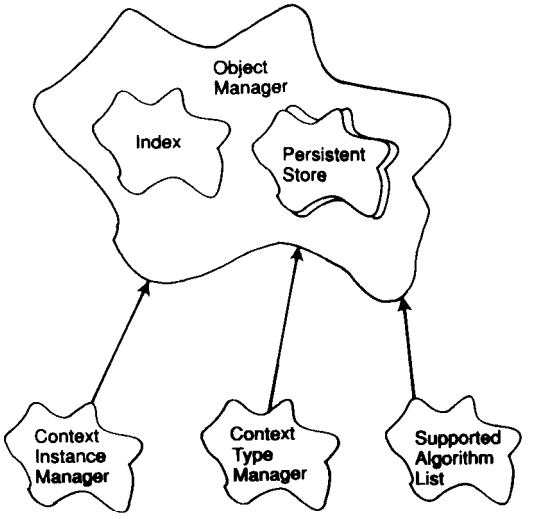


Fig. 5 Object Manager Class Structure

Oriented manner. One possibility is for Context Instances to include configured algorithm sub-objects and which will provide the appropriate services according to the intended functionality of the Context Instance. Thus a confidentiality Context Instance would only offer services to encipher and decipher data.

9 Summary

An algorithm-independent interface can be offered to the clients of a cryptographic facility through the means of Cryptographic Context Instances which encapsulate the information required to provide the required quality of service.

The creation of Context Instances and hence the use of cryptographic algorithms can be controlled through means of Cryptographic Context Types. Context Types can be set up to model the security policy of the domain with regards to the use of cryptographic algorithms.

An algorithm-independent interface can also be offered to administrative applications for the creation of Context Types by Cryptographic Algorithm objects encapsulating the knowledge of how they can be used to provide different levels of service.

The dependency of the cryptographic facility upon the algorithms themselves can be minimised through the use of an Algorithm class hierarchy from which algorithms must be derived, and the use of polymorphism to allow specific algorithm objects to be treated through a more general super-class.

Bit String objects can be used to encapsulate platform-specifics regarding the representation of data and to provide services to algorithms to treat the data as though it were a bit string. This allows algorithms to be portable to different environments.

Finally, a general object manager scheme was outlined which is capable of extension into specialised object managers for different types of managed objects and which can be extended to cater for different types of long term storage.

The concepts outlined in this paper have evolved over a period of 3 years and have been proven in successive implementation releases. The concepts have given rise to 5 patent application registrations in the UK alone.

Acknowledgements

The author would like to thank Brian Dowler, Tom Parker, Charles Lambert and Roy Jones (now retired) of ICL's security technical steering group for their constructive comments and criticisms of the concepts as they evolved; the CSF development team – Mike Beasley, Keith Jenkinson and Rory Caldow for their part in planning to put the concepts into practice; and Derek Guyatt of the ICL patents department.

References

IBM. *Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference.* IBM, 1990.

COAD, P. and YOURDON, E. *Object-Oriented Analysis.* Prentice Hall, 2nd edition 1991.

BOOCH, G. *Object-Oriented Design with applications.* Benjamin/Cummings, 1991.

FAIRTHORNE, S.B. *OPEN*framework *Security Quality.* OFD 41, ICL OPENframework Division, 1991.

PRESS, J. An Introduction to Public Key Systems and Digital Signatures. *ICL Tech. J.* Vol. 6(4), 1989.

ISO 9797. *Data integrity mechanism using a cryptographic check function employing a block cipher algorithm.* ISO, 1989.

ISO 10116. *Modes of Operation for an n-bit Block Cipher Algorithm.* ISO, 1991.

Biography

Jim Press

Jim Press joined ICL in 1980 after graduating from Southampton University with a BSc. (Hons.) in Electronic Engineering. After working for 4 years on customized product development, he began working on aspects of secure networking and the use of cryptography in the commercial sector.

He now works for ICL Mid-Range Systems Division at Reading in the Open Distributed Processing unit. He is responsible for designing the Cryptographic Support Facility (CSF) and is involved in SESAME, the European collaboration between ICL, BULL and SIEMENS-NIXDORF in the area of secure distributed systems. Jim is also a Member of the Institution of Electrical Engineers (IEE) and a Chartered Engineer.

CHISLE: An engineer's tool for hardware system design

A. Jebson, C. Jones and H. Vosper
Corporate Servers Product Group, ICL Corporate Systems, West Gorton, Manchester

Abstract

This paper describes the methods used in formalising the approach to hardware development. It describes a methodology based on hierarchic design decomposition and mixed multi-level modelling using a semi-formal language "CHISLE" (Combined Hardware and Interface Specification Language for Engineers). This language defines both functionality and interfaces and is compiled into executable models. The approach permits the main problems facing hardware designers to be tackled ie. how to specify the design in a concise, unambiguous and understandable manner and how to represent the parallelism and time relationships inherent in complex pieces of hardware. The results and achievements to date are also described and the paper finishes by giving some possibilities for future enhancements.

1.0 Introduction

The methodology currently used for the design of large Mainframe hardware places great emphasis on the specification and verification of the high level design.

It has evolved over the last 20 years from the "build-and-test" methodology employed on the early 2900 mainframes, through the register transfer level (RTL) modelling methodologies used for the Series 39 level 80 and SX ranges to the methodology used for todays mainframes.

The methodology employed on the earliest 2900 mainframes used little or no modelling to check that the implementation of the machine was correct. These machines were designed, built, and then tested. However, given the relatively small number of gates used in such machines (less than 50 000) only a small number of faults remained to be found when the first prototypes were switched on.

Later Series 39 mainframes with much higher gate counts (up to 1 million gates), employed a methodology where test programs were run on a register transfer level model (MSIM [Hodgson, S., 1984]) of the machine and patterns extracted from this model were compared with the actual gate level implementation. This methodology led to a very large reduction (relatively speaking) in the numbers of faults in the first prototypes, and is described in (Abraham, G.P. et al, 1990) and (Eaton, J.R. et al, 1990).

Following completion of the design of the latest SX mainframe, an analysis of this design was undertaken. This analysis identified a number of design errors, and the stage at which each error was introduced. This showed that the approach of extensive testing of the RTL specification and then extracting and running patterns from this on the gate level implementation resulted in a very small number of implementation faults. However, there were a number of significant weaknesses in the earlier stages of the design process:

— the design went in a single stage, from high-level specifications to RTL level models.
— ambiguities in the (English language) specifications meant that a significant number of errors were introduced early in the design process.
— interfaces between major components of the system were poorly specified, leading to interface mismatches.
— the performance of the RTL model of the system severely limited the amount of testing that could be performed.
— much of the testing was based on the design realisation rather than the specification.

Given the increase in complexity of the next generation of mainframes over that of the SX mainframe, these weaknesses would result in an unacceptably high number of faults in the first prototypes. Therefore, a number of improvements in the methodology were suggested:

— formally define interfaces.
— represent the design in a concise and unambiguous manner in relatively small understandable chunks.
— model the design in a top-down hierarchic manner.
— ensure that the low-level design of a component of the system is independent from the design of its neighbours, such that design changes in a component have minimal impact.

These lead to the development of a semi-formal specification language – CHISLE (Combined Hardware and Interface Specification Language for Engineers) which allows executable specifications of both interfaces and functional components to be written in a common language.

2.0 Concepts

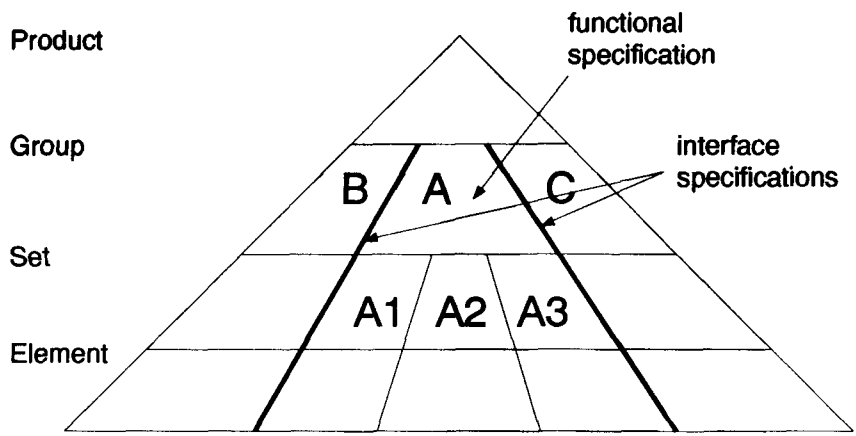
2.1 Methodology

The methodology which has evolved has three key elements:

— hierarchic design decomposition
— "firewall rules" for interfaces
— mixed, multi-level modelling.

These elements are described in the sections below.

2.1.1 Design decomposition When specifying some "thing", this thing is described in terms of a design hierarchy, where each level in the hierarchy is a more concrete "implementation" of its parent, as shown in Figure 1. The top level of this hierarchy is the Product, which is decomposed into three further levels: Group, Set and Element.



Unit "A" at Group level decomposes into units "A1","A2" & "A3" at Set level, referencing the same interface specifications to units "B" & "C".

Fig. 1 Design Decomposition Triangle

The Group Level represents the top level functionality of the machine, and is a collection of "asynchronous" units connected by "asynchronous" interfaces. Note, "asynchronous" does not imply the absence of a "clock". It means that the unit must not be functionally sensitive to the latency of an interface, that is, it must not care how long it takes for an interface request to be serviced. Nor must the unit be sensitive to the latency of any internal "services".

Each group is decomposed into Sets with, roughly, a 1 to 5 expansion. A Set is considered to be a synchronous unit, with asynchronous interfaces. That is, it must not be functionally sensitive to the latency of its interfaces, but knows how long any internal operations will take.

Each Set is decomposed into Elements. An Element is considered to be a synchronous unit, with synchronous interfaces. That is, it is a register transfer level description. Again there is roughly a 1 to 5 expansion from Set to Element level.

2.1.2 Firewall rules These rules were designed to allow a manageable partitioning of the design. They attempt to achieve the key attributes of understandability, verifiability, and efficiency. The rules are:

i) *asynchronous behaviour* – the design of a unit must not be functionally sensitive to the latency of its interfaces or its neighbouring units.
ii) *isolation* – the only information that any unit of design has about another unit's behaviour must be passed across a formally defined interface. No unit should need to have knowledge of why a function has been requested or of what is happening in any other unit.

Use of these rules yields a design where behaviour of a unit is independent from that of its neighbours. That is, there is a large degree of design independence, allowing the design and testing of a unit to proceed in (relative) isolation.

2.1.3 Mixed, multi-level modelling (mMm) This modelling strategy allows the design to be modelled at various levels in the design hierarchy. For example, the Element level of the processor could be modelled with Set Level or Group Level of the memory and IO processor, etc.

Using this strategy, a unit or its decomposition may be tested without changing any other components of the model. This tests that the implementation of a level meets its specification. That is, the decomposition is functionally equivalent to its parent.

2.2 Language Concepts

The CHISLE language is based on the discipline of functional programming, which has a strong, and well understood, logical foundation. (DILLER ANTONI, 1990). The use of such a language for specification purposes has many advantages. For example, it leads to concise, unambiguous and exact specifications that are easy to reason about. Additionally, unlike many other specification languages, the specifications are readable since they are couched in familiar programming terms, rather than in the mathematical notations used by other (more formal) specification languages.

One of the key ideas in specification languages is that specification and implementation are kept separate. The specification should state what the component is supposed to do and not how it is to go about achieving its task. Separating specification and implementation results in the separation of the often conflicting tasks of correctly solving the problem in hand and that of building an efficient component.

The methodology underlying the use of CHISLE stresses the primacy of *declarative* thinking over *procedural* or *imperative* thinking. One of the things that people find most difficult to do when they start writing formal specifications is to stop thinking procedurally. The goal is to describe what a component of a system is going to do, and should not be concerned about efficiency or even implementability. Thus operations are specified by giving their pre-conditions and post-conditions, and not by giving a procedure for how these operations are to be carried out. The technique of ignoring the computational or procedural difficulties associated with a problem and only concentrating on its functional specification is known as *procedural* or *operational abstraction.*

A feature that distinguishes CHISLE from most other specification languages is that it is executable. In the general case, specifications written in other languages cannot be executed. As a consequence of this, CHISLE incorporates many features normally found only within imperative programming languages. However, this has been done within a declarative framework, retaining the principles of functional programming.

Procedural abstraction is one of the two main ways in which abstraction is used in CHISLE to manage the complexity of the problem of correctly specifying a large, complex system. The other main abstraction technique is *representational abstraction.* This involves using high-level structures – like arbitrary sets, functions, queues, and so on – in the specification of systems without worrying about how these are eventually going to be implemented. This has the beneficial consequence that in solving a specific problem designers are allowed to think using the structures most suited to the problem in hand, rather than being forced at a premature stage to think in terms of the data and control structures available at later stages of design.

Once a system has been specified, it must be verified. There are two main approaches to verifying that an implementation meets its specification, namely the proof-theoretic and the model-theoretic approaches. Model-theory does this by executing a model of the implementation and showing that its behaviour fulfils the specification in all circumstances, whereas proof-theory does this by constructing a rigorous mathematical proof that they are identical.

The model-theoretic approach is fundamentally limited by the amount of simulation that can be done and, for a large model, it is not feasible to perform an exhaustive check. Also, many faults can be introduced during

the coding of the model. However, it does have the advantage that the approach is well known and widely used and the techniques are mature.

The proof-theoretic approach has the inherent problem that, though it proves that an implementation is identical to its specification in all circumstances, it states nothing about the validity of the specification itself. The other major problem with this approach is that the construction of the mathematical proofs cannot, in the general case, be done automatically, and is, currently, not feasible for any but the simplest specifications.

One of the fundamental concepts in the methodology is that of design by decomposition. By this, we mean that when specifying a large, complex system the problem should be decomposed in a top down manner from a high-level abstract specification to a low-level concrete specification.

This means that the specification of some component "A" will be decomposed into a number of lower-level specifications "A1", "A2", and so on, interconnected by a number of interfaces, as illustrated in Figure 1.

Consider now, that this component "A" has interfaces I1 and I2 to components "B" and "C" respectively.

In order that the design of components B and C may proceed independently from that of component A, the specifications of A, I1, and I2 must be sufficiently abstract that any changes to the implementation of A (sub-units A1, A2, etc) do not affect the design of components B and C, as seen in Figure 2.

This concept of design independence becomes important when the high level specifications of components B and C are being used with a lower level specification of component A.

Here, an implementation of A is being tested against the specifications for components B and C, and therefore, it is important that the models generated for components B and C have the "loosest" possible behaviour that still fulfils the requirements of the specification of these components. Thus, the implementation of A will be tested against the behaviour of every possible implementation of B and C, rather than the behaviour of a particular implementation.

To achieve this "loose" behaviour, an interface and how a component "drives" the interface must be specified in such a manner that a high level specification of a component is insensitive to low level implementation changes in both the interface and the components that it communicates with. The specification must express the *essential* features of the component rather than details particular to an implementation of the component.

mMm allows the Implementation of A to be simulated with the behaviours of B and C

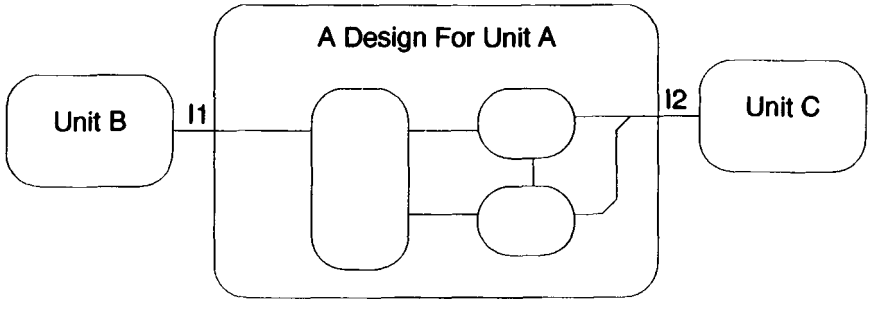


Fig. 2 Mixed Multi-level Structure

Also, interfaces should be specified only once, and this definition must be used for all levels of specification. This does not mean an interface specification cannot change, but that there should be a single specification which all components use.

It is also important that a single language is used for the specification of both interfaces and the functionality of a component, at any abstraction level from the most abstract through to concrete register transfer level specifications.

3.0 Language

The CHISLE language has been designed as a "common" language for the description of both interfaces and the functionality of units. Although the syntax of the language is similar for these two different types of specification, the semantics differ and it is worth considering the two "dialects" separately.

The sections that follow will use the specification of a simple "memory" as an example.

3.1 Functionality

It should be stressed that, as stated in the Concepts section above, the language has been based on declarative rather than procedural thinking. That is, a specification is a description of what a component or function has to do and not how it is done. The functional specifications of units are based on a client/server architecture, where each unit is considered to be a "server" providing services to its neighbouring units. The specification of this server describes the permissible set of services supplied by the unit. In the case of a simple memory which can handle up to 16 concurrent memory requests this might be written:

```
server memory
begin
    unique ((4) bit BuffNum): HandleRequest;
end;
```

A service is a thread of actions that specify how a request from a client is to be processed. In our memory example, the service "HandleRequest" might be written:

```
service HandleRequest((4)bit BuffNum)
concurrent on (BuffNum)
begin
    mem_func Function;
    add_mode Address:
    tag_mode Tag;
    (16)word Data;


    Get_Request(BuffNum, Function, Address, Tag);
<0> if Function in {WriteReal, WriteVirtual}
    then
        Get_Write_Data(Tag, Data);
<-->      Commit_Write(BuffNum, Function, Address, Data);
    else
        Read_Store(BuffNum, Function, Address, Data);
<-->      Send_Read_Data(BuffNum, Tag, Data);
    fi;
end;
```

An action is a base function or set of functions required by the service. For example, the action "Read_Store" might be written:

```
action Read_Store  (
                     (4)bit BuffNum,
                     mem_func Func,
                     add_mode Addr,
                     ref(16)word Data
                   )
concurrent on (BuffNum)
pre-condition SchedControl(schedule, BuffNum, Func, Addr)
post-condition SchedControl(remove, BuffNum, _, _)
begin
    Data := MainStore(Addr);
end;
```

The examples above, highlight the use of some key features of the language, described below.

3.2 *Features*

The key features that increase the expressive power of CHISLE include: concurrency, time relationships, pre- and post-conditions, and powerful data typing.

i) *Concurrency.* That is, the ability to express threads which may have more than one concurrent instance. In the example above, the service HandleRequest, and the action Read_Store are both concurrent, and there may be up to 16 active instances of each. Associated with each instance is a unique identifier, in this case the parameter "BuffNum".

ii) *Time Relationships.* These relationships specify the time taken within a thread. In the above example, two of the many time relationships allowed in CHISLE were shown: the time relation "a <0> b" states that b must start immediately a has finished; "a <--> b" states that b must start some time (conceptually up to an infinite time later) after a has completed.

iii) *Pre- and Post-Conditions.* These are rules that govern when something may start or finish. These can be just a simple boolean expression (e.g x = 5), or may be very complex. Where they are complex, CHISLE allows their specification separately as "conditions".

iv) *Data Typing.* As specifications in CHISLE may be at one of several levels from the highly abstract Group level through to the concrete (RTL) Element level, the data types allowed range from abstract constructs such as queues, sets, polymorphic data types, etc to words and arrays of bits.

3.3 *Support of Test Scripts*

The functionality of a unit can be tested across its interfaces by specifying tests in the same language. A functional unit may have interfaces with units A, B, and C. Therefore, for a test script to fully test this unit it must be able to drive all these interfaces. A test script can be compiled along with the units functional specification, generating a model known as a "Test Harness".

Specifications of interfaces in CHISLE are expressed in terms of seven layers. Proceeding from top to bottom, these layers are:

- *Interface layer.* An interface is considered to be the sum of the total information exchanged by a collection of units.
- *Protocol layer.* Information is exchanged by a collection of two or more units in a "conversation". The protocol layer of the interface describes the legal "conversations" on that interface.
- *Transaction layer.* This specifies the information that will be transferred between the units on an interface for one "conversation". For example a memory read request followed by the data response.
- *Item layer.* An item is information transferred between units in a single direction and consists of an ordered, untimed sequence of packets. For example a 64-byte data response, which consists of eight packets of 8 bytes, and the time taken between packets is unknown.
- *Packet layer.* A packet is information transferred between units in a single direction and consists of an ordered and timed sequence of slices. For example an 8-byte data packet might be specified as taking 8 clock cycles.
- *Slice layer.* A slice is information transferred between two units in a single direction in a single clock cycle.
- *Wires layer.* The wires layer specifies what physical connections exist between units.

When an interface is driven by a functional unit, the level at which that unit is permitted to drive the interface is governed by the specification level of the unit. Group level specifications are only allowed to drive an interface at Item level; Set level specifications at Packet level; and Element level specifications at Slice level.

When specifying an interface in CHISLE the language features are as for functionality; additionally interface specifications have the following properties:

- the specification of an interface is expressed in terms of what information is exchanged between units and of the rules for this exchange, rather than in terms of how a unit is either to drive the interface or to receive data from it.
- from the specification of an interface it is possible to derive the method by which each functional unit must drive the interface (at any level of decomposition). That is, both the method by which data must be sent by a transmitting unit, and how it must be accepted by a receiving unit.
- from the specification of an interface it is possible to derive checks on the legality of a transfer. For example, if an item is sent by a Group Level specification then this will be checked for legality against all

possible transactions and the protocol of the interface, as seen in Figure 3.

- the specification of an interface can include rules governing the legality of a transfer. For example, rules stating that only four read requests may be outstanding at a given instant can easily be expressed.

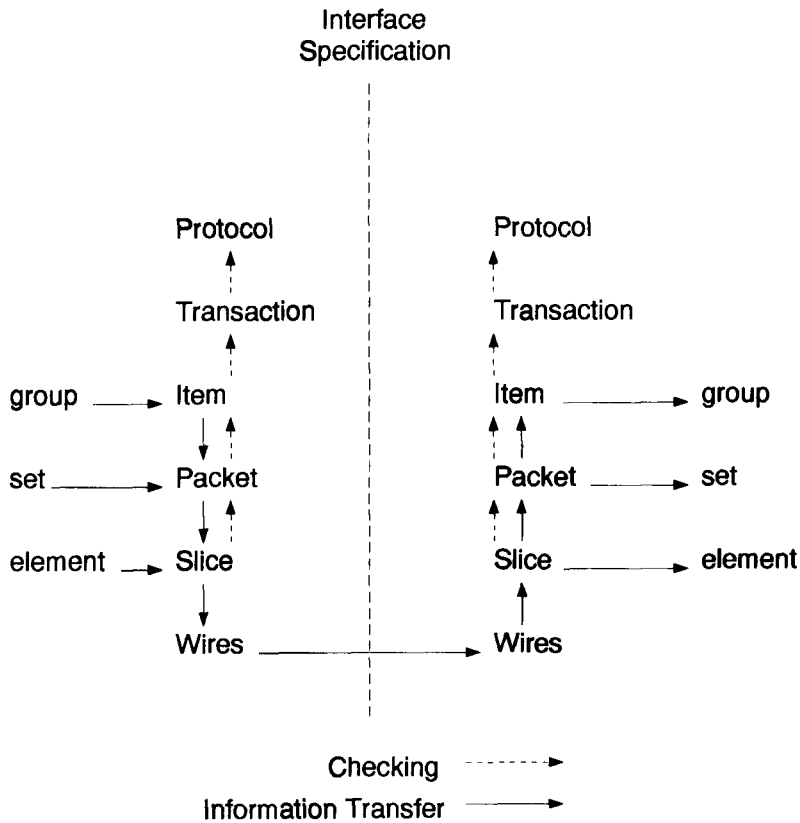


Fig. 3 Interface Model Hierarchy

4.0 Results and Achievements

4.1 Use of the Language

The use of the CHISLE language alone has given major benefits to the current mainframe development project. At the top level of the design (the Group level) there are five major components and nine interfaces between these units. All these have been specified in CHISLE (in roughly 10 000 statements). A number of design errors and interface problems have already been found just through writing the specification in a formal manner.

4.2 Training

One of the major achievements to date was the training of the design engineers (and their managers) in the design methods, as well as in the tools and the CHISLE language itself. The acceptance by everyone of the high level design concepts, design decomposition and formal specifications had major benefits in terms of common understanding of the design, and its subsequent specification. Some 45 designers have now been trained in the use of CHISLE and support documentation (reference manual, user guide, etc) has also been provided.

4.3 Tools

4.3.1 The CHISLE compiler The prototype compiler (roughly 40 000 lines of 'C') has been written and validated. As well as the expected syntactical and type checking the compiler provides a semantic analysis phase which checks for certain types of ambiguity and design rule violations.

The specification for each of the five components of the Group level have been compiled into executable behavioral models. Each of these models has been unit tested by the responsible design teams. The unit tested models have been collected together to form a system model which has been tested using a number of Series 39 defined tests. To date the model has run for more than 6 million clock cycles.

4.3.2 Test harness generation The unit testing of each component of the system model required the generation of a large number of test scripts. These were written in CHISLE, and compiled to produce test harnesses for each component.

These test scripts were written by a separate Design Verification team to give independent functional verification of each design component.

4.3.3 Network generation To support the design decomposition process, a decomposition structure can be specified in CHISLE which is then compiled into a simulatable network instancing the relevant components, and specifying their interconnection.

4.3.4 Design control The compiler has been incorporated into a design support environment which provides mechanisms for version control and configuration management facilities, covering CHISLE source, compiled code, model build, test scripts, and support documentation.

4.3.5 Simulation performance The models generated from CHISLE specifications are run using the MSIM simulator. Currently, the system model achieves roughly 180 clock cycles per second on a single node Series 39/level 80 machine. This compares very favourably with the performance of 5 clock cycles per second for the equivalent RTL model of the SX processor (on the same machine).

4.3.6 Future developments As the CHISLE language and mMm methodology have evolved, the possible enhancements seem endless. We therefore have to be extremely careful in the choice of extensions to the tool-set. Our current plans include:

- an optimisation pass for the compiler which will generate more efficient models.
- a new "back-end" for the compiler which will generate 'C' rather than S3, permitting simulation on multiple hardware platforms (the only S3 compiler available is specific to Series 39).
- extensions to the compiler to include full test cover measurement (ie. which paths in the model have been exercised) into the generated simulation models. This will allow easier identification of untested paths.
- the syntax and sematics of the language need "cleaning up". The current syntax and sematics evolved in a relatively *ad hoc* manner and contain several inconsistencies.

In addition we are considering the problems associated with:

- the generation of software products from high level specifications.
- logic synthesis from register transfer level specifications, including the overlay of testability features.
- tools to aid the decomposition process by analysis of high level specifications and suggesting possible partitions of these specifications.

5.0 Conclusions

In this paper we have described an approach to the development of a powerful methodology and tool set which enables large complex systems to be formally specified. The approach is based on hierarchic design decomposition with specifications of Functional Units and their Interfaces.

Executable models can be generated automatically from these specifications which allows testing of the design and the specifications.

We are successfully using the approach in Corporate Servers Product Group for the development of ICL's future mainframes, with major improvements in both productivity and design quality.

Acknowledgements

The authors would like to thank the various members of Corporate Servers Product Group and Design Automation departments at ICL, West Gorton for their help and impatience in developing and using the techniques and tools described in this paper.

References

ABRAHAM, G.P., FREETH, D.C., and VOSPER, H., SX Design Processes, *ICL Tech. J.* Vol. 7 No. 2 pp. 212–232, 1990.

EATON, J.R., ALLT, G., and HUGHES, K., The SX Node Architecture, *ICL Tech. J.* Vol. 7 No. 2 pp. 197–211, 1990.

HODGSON, S., A Multi-Level, Mixed State Simulator for Hierarchical Design Verification, *IEE European Design Automation Conference* 1984.

DILLER, A., *An Introduction to Formal Methods*, John Wiley & Sons ISBN 047192489X 1990.

Biographies

Chris Jones

Chris Jones joined ICL in 1984 as a sponsored student. He graduated from Manchester University in 1988 with an honours degree in Computer Engineering.

He started work on the development of the internode system for the Series 39 SX System. He also worked on the Processor of the Series 39 DX System.

He is now working on the memory system design for future mainframe development.

Tony Jebson

Tony Jebson joined ICL in 1978 as a sponsored student. He graduated from Bristol University in 1982 with an honours degree in Electrical and Electronic Engineering.

He started work on the development of the Series 39 Level 30 Low Speed Computer. Since then he has worked as a team leader on the Flagship project (an ALVEY programme), and on the Series 39 SX System.

He is now responsible for I/O and Inter-Node System design for future mainframe development.

Harry Vosper

Harry Vosper joined ICL in 1962 as a student apprentice. He graduated from Queen's University Belfast with an honours degree in Electrical Engineering and Electronics. He transferred to Development in 1970 working originally in Microsystems on thick film and integrated circuit design.

He became Hardware Quality Manager for the Estriel project in 1983 and subsequently Future Products (Mainframe Systems) in the Essex Project.

He is currently the manager of Development Route & Tools Team in Corporate Servers Product Group which provides the development processes, database environment and tools necessary to support mainframe development.

Distributed Detection of Deadlock

Steve Hilditch
ICL Bracknell UK
Tom Thomson
ICL Manchester UK*

Abstract

Data consistency in distributed database management systems is often implemented by transactions which separately preserve consistency. Concurrent transaction execution can be ensured by locks on data and the discipline of two-phase locking. Transactions can be forced to wait for others to release locks. It is possible that a complete cycle of waiting transactions can form deadlock. In a distributed memory environment, some deadlock can be detected locally, i.e. without using the communications network. But the most difficult case to detect is when a number of transactions on more than one network node all wait for each other so that none can proceed. This is called distributed deadlock.

We present two simple algorithms for the detection of distributed deadlock on a shared-nothing distributed database system. They are designed to be scalable and minimise network message traffic. An object-oriented approach is used.

1 Introduction

Much use has been made in recent times of databases which are distributed over a number of processors connected by a communications network. Whether the communication links are just across the room or are intercontinental, the main difficulty arises because of the communication delays. The consistency of the distributed database and the liveness of the distributed database management system must be ensured by careful communication, allowing for possible delays in information.

1.1 Two-Phase Locking and Deadlock

Given the decision that data is only represented once in the distributed database, transactions and Two-Phase Locking can be used to ensure data

*The authors were part of the European Declarative system, ESPRIT project EP2025.

consistency. Two-Phase Locking is an agreement by all transactions to execute in two phases: during the first locks can only be taken, during the second locks can only be released. Locking of shared resources can enforce a fair treatment of the transactions but at the cost of deadlock.

Deadlock is caused by a complete cycle of transaction dependencies. One transaction is said to be *dependent on* or *waits for* another transaction if the former must wait until the latter has finished with some shared resource. Transaction dependencies and deadlock can be modelled by a directed graph. A transaction waiting for another transaction is modelled by a graph edge from a graph node representing the former transaction to a graph node representing the latter transaction. Any cycle in the so-called *global dependency graph* indicates the presence of deadlock which must be resolved. Aborting a transaction corresponds to removing a node from the dependency graph, together with all the graph edges to and from it. This, of course, will remove the cycle which corresponds to resolving the deadlock.

Some deadlock can be detected locally, i.e. without using the communications network. But the most difficult deadlock to detect is when a number of transactions on more than one network node all wait for each other so that none can proceed. The difficulties of detecting distributed deadlock are all caused by network communication delays:

- maintaining up-to-date information on transaction dependencies
- allowing for the late arrival of messages
- implementing the detection process to avoid clogging the network
- implementing the detection process to resolve deadlock as quickly as possible.

1.2 Deadlock Resolution and Victims

When deadlock has been detected, resolution demands that at least one transaction must be told to cancel a lock request or release a lock. That transaction is called the *victim* of the deadlock resolution process. In most cases the victim is forced to release all its locks, abort and wait to be restarted as if it had never existed. If a transaction were allowed to release one lock and then obtain it again later, then the transaction would have violated the two-phase locking strategy. This would allow data to become inconsistent. This paper assumes for simplicity that victims are forced to abort and roll-back to their starting points.

It is possible to prove that no transaction is 'starved' of resources if we have a fair method of choice of victim once deadlock is found. In order to be able to choose fairly a victim, we assume that each transaction has a unique identifier which consists of its source processor or site identifier as the least significant bits, and the local timestamp as its most significant bits. We only assume that local timestamps have to be kept approximately the same. This can be achieved using an algorithm such as that of Lamport [10]. Since a

transaction identifier (tid) uniquely determines a transaction, and all tids can be totally ordered, when faced with the choice for victim we can always choose the transaction with the highest tid. The transaction with the lowest tid is the one which 'has been on the system for the longest time' or the 'oldest' transaction.

We also make sure that a transaction's tid is kept if the transaction is restarted or 'reincarnated' after being chosen as a victim and aborted.

To show that no transaction is perpetually starved of resources we have only to notice the following two points:

1 The oldest transaction will never be chosen as a victim.
2 Every transaction will eventually become the oldest transaction unless it finishes first.

1.3 Distributed Algorithms for Detecting Distributed Deadlock

Many have attempted in recent times to give algorithms which detect deadlock between transactions which run in a distributed environment [7, 9, 11, 3, 2, 15, 6, 1, 5, 4, 14]. Of these detection algorithms, some are not fully distributed themselves.

In this paper we present two variants of a distributed algorithm to detect distributed deadlock. It, like many other deadlock detection algorithms, uses the concept of a global waits-for directed graph to model the transaction dependencies.

Avoiding Bottlenecks: In contrast to the algorithm of Ho and Ramamoorthy [9], our algorithms do not assume any logical hierarchical node structure between the distributed constituent parts of the deadlock detection process. There is no difference between the nodes of the network, there are no management hierarchies or special nodes. Therefore, communication bottlenecks are avoided in our algorithms.

Avoiding Global Synchronisation: In contrast to the algorithms of Belik [1], Obermarck [11], Bracha and Toueg [2] and Gafni [6], our algorithms do not involve any global synchronisation of message passing. In such a case, if communication with one site or node in the system is slow, the whole algorithm is slowed. Therefore, our algorithms offer the advantages of concurrency and scalability.

Independent Decision Making: In contrast with the algorithm of Obermarck [11] whose deadlock detection is done at intervals, to detect all deadlock up to that time, our algorithms attempt to detect deadlock by immediate probes. Belik [1], Sinha and Natarajan [15], Roesler and Burkhard [14] and Choudhary, Kohler, Stankovic and Towsley [4] also adopt the same approach. This results in a more immediate detection of deadlock. Our

algorithms detect deadlock without any artificial delay, and provide early resolution to reduce lock existence times, the number of lock conflicts and hence response times.

Scalable Distribution: The distributed algorithms we present only involve network nodes which are directly involved in a particular lock conflict. Therefore, our algorithms can be run on massively parallel machines with optimal performance.

Communication Economy: In contrast with the algorithms of Chandy, Haas and Misra [3] and Bracha and Toueg [2] whose transactions send out deadlock detection probes when they fear deadlock, our algorithms only send detection probes when a new cycle may have been completed. Probes are not needed at other times. There is no need to send a repeat 'reminder' message in the absence of system failure. This means that under heavy load, our deadlock detection process will not involve escalation of communication traffic. Also, each deadlock message sent has fixed maximum length.

Tidiness: Our algorithms offer the removal of old deadlock detection information. This tidiness saves storage and processing time.

Conceptual Simplicity: We provide algorithms which are simple to describe and verify, involving few operations (two at most three) and few data structures (one or two lists). We therefore provide C++-like code for each algorithm. A Temporal Logic type proof of their correctness is available from the authors.

1.4 Our Network Requirements and Model

In this section we state the problem which we solve and the assumptions on the reliability of the network we make. We refer to Raynal [13] for a helpful classification of distributed algorithms and their assumptions.

The general problem is that of resource allocation scheduling. Each resource is stored in the memory of exactly one processor; there are no duplications.

We will assume that the processors are fully-connected. Our algorithms assume this, and the number of network messages could well be more on other configurations of processors, unless the resource requirements of most transactions are readily accessible, i.e. reachable directly through the network. According to the definitions of Raynal [13], our algorithms have "strong symmetry", in that the same algorithm runs at each processor.

The main network assumption is that of the reliability of the network when delivering messages. We assume that every message sent from one processor to another will eventually arrive at its destination. Given this assumption of reliability, we are able to make estimates on the number of network messages which our algorithm requires. It is, of course, possible to ensure

this reliability by implementing a communications protocol on the top of a more unreliable network, but we shall not discuss this here. We do not need to assume that there is a fixed bound on the amount of time that a message takes to arrive across the network.

We also assume that no duplicate messages are created by the network and that messages arrive in the order in which they are sent. Our algorithms can be modified to cope with messages out of sequence as we shall show below. Out-of-sequence messages may occur, for example if a message corrupted in transmission had to be re-transmitted. Note that it is difficult to ensure the order of arrival of messages sent through different third parties: it is easy for a message to 'overtake' another given a different (indirect) communication channel. However, we assume that every message can be sent directly without the use of a third party.

1.5 The Lock Manager's Rôle

The detection of deadlock is dependent on the collection of the correct transaction "waits-for" information: each time a transaction is forced to wait for another transaction to release a lock before it can obtain the lock, a new edge is added to a directed graph whose vertices are transactions. Our algorithms, like most in the literature, keep track of the state of this "waits-for graph". The presence of a directed cycle in the waits-for graph indicates deadlock involving all the transactions within the cycle. The rôle of the system lock manager in our model is to inform the deadlock detection process (in the form of one deadlock detection manager per transaction) of the existence of new waits-for edges and the removal of old waits-for edges.

A simple algorithm for informing the deadlock detection process can be given as follows:

- If a transaction **S** requests a lock on some database resource which is incompatible with an existing lock held by another transaction **T** then the deadlock detection process should be notified by the following message:

  change_graph(T,ADD).

  This message is sent to the deadlock manager associated with the transaction **S**.

- Each time a transaction **T** releases a lock, a queued lock request may be granted. This lock request's transaction **S** will no longer be dependent on **T**. Therefore, a

  change_graph(T,SUBTRACT)

  message should be sent to the deadlock manager for the transaction **S**.

Furthermore, each of the transactions **R** which sent outstanding queued lock requests are now dependent on **S** and for each such **R** the following messages should be sent to the deadlock detection manager for **R**:

change_graph(T,SUBTRACT)
change_graph(S,ADD).

We should note that each ADD edge message which is not sent could result in undetected deadlock. Each SUBTRACT message which is not sent might result in detection of non-existent deadlock, resulting in unnecessary transaction abortion, and certainly would cause obsolete transaction dependency information to be retained in the deadlock manager, wasting storage space and slowing the detection process.

We believe that it is possible significantly to reduce the number of messages sent to the deadlock detection process by more careful analysis of the contents of lock requests queues. This has been described in detail in [8] but is not considered here.

2 Deadlock Detection

We shall first describe our basic distributed algorithm for deadlock detection. Later we describe a more complicated probe-remembering algorithm.

We present the algorithms in C++ object-oriented style. In order to be able to describe distributed algorithms we shall assume the following two extra enhancements to the semantics of C++:

1 Each instantiation of a class is a separate process which can be scheduled by the process scheduler.
2 Communication is by asynchronous messages (rather than by RPC – Remote Procedure Call). Each instantiation of a class has its own messages received queue to which it responds one by one. The messages it understands are precisely calls to its members with appropriate parameters.

For simplicity of presentation, we assume:

- Each transaction is created with its own instantiation of the deadlock deadlock manager class. When a transaction terminates its associated deadlock manager is removed also.
- Each change of transaction conflict is sent separately from the lock manager to a transaction's deadlock manager. Such a change is sent as follows:

  change_graph(transaction,change),

  where the *transaction* parameter is the transaction which first requested

the lock and the deadlock manager called is associated with the later transaction whose request caused the lock conflict.

- The numerical value of the identifier of each deadlock manager object is assumed to be an accurate representation of the transaction's relative age on the system. Therefore, each deadlock-resolution victim can easily be selected by the fact of its relatively large deadlock manager identifier. As each probe is sent between the deadlock managers, *current_victim* always records the identifier of the deadlock manager of the youngest transaction involved in the corresponding deadlock chain.

2.1 The Basic Detection Algorithm

The basic deadlock detection algorithm consists of a collection of instances of deadlock detection managers, one for each transaction in the database management system. Each deadlock manager sends either *probe* messages to other deadlock managers or abort messages to the database management system's transaction scheduler. Each message receives no reply, and the communication is asynchronous.

We give below the *class* definition of the deadlock manager abstract data type. Each instance of the deadlock manager class will therefore have its own state consisting of a list of transactions for which it is waiting – *conflicting_trans*. Access to the state is restricted to the two *member* operations: **change_graph** and **probe**. The former member is only called by the lock manager, the latter is only called by other deadlock manager instances.

The only output from the deadlock detection algorithm occurs when a victim is chosen and a deadlock manager instance sends the message: **abort(victim)** for some transaction *victim*.

The algorithm assumes that the relevant deadlock manager is told of any changes to the waits-for graph involving its associated transaction: either by a

change_graph(transaction,ADD)

message or a

change_graph(transaction,SUBTRACT)

message. In particular, even if the deadlock detection algorithm chooses a transaction for abortion, (a *victim*), references to the aborted transaction in other deadlock managers must be removed by explicit

change_graph(victim,SUBTRACT)

calls from the lock manager.

The transaction_list *conflicting_trans* is allowed to have duplicate transactions if the deadlock manager's transaction has conflicted with another transaction over lock requests to more than one resource. Duplicates are essential for the correctness of this algorithm. Therefore, the standard list manipulation function *list_remove(conflict,conflicting_trans)* is assumed to remove **only one** reference to the element *conflict* in the transaction list *conflicting_trans.*

```
typedef enum {
    SUBTRACT,
    ADD
} change;

class deadlock_manager {
    transaction_list conflicting_trans;
public:
    void change_graph(transaction conflict,change add);
    void probe(transaction initiator,transaction current_victim);
};
```

2.1.1 change_graph

The first operation or class member to be described is **change_graph**. It is called by the lock manager when there has been a change in the waits-for graph; either an edge has been added or one has been removed. Each edge of the waits-for graph is directed; there is a *waiting* transaction and a *waited-for* transaction. The **change_graph** message is sent to the deadlock manager of the waiting transaction with parameter the waited-for transaction and a boolean flag indicating whether the edge is to be added or removed. **change_graph** either adds or subtracts the waited-for transaction to its list of conflicting transactions. If the edge is new, a **probe** message is sent to the waited-for transaction's deadlock manager.

Example For example, if a transaction **a** requests a lock that conflicts with a lock held by transaction **b** then the message **a.change_graph(b,ADD)** is sent. This causes the transaction **b** to be added to the *conflicting_trans* list of the deadlock manager for **a**. Then the message **b.probe(a,a)** is sent to the deadlock manager for the transaction **b**. See figure 1, messages marked*.

```
void deadlock_manager::change_graph(transaction conflict,change add)
{

// Note: change_graph is called by the lock manager when either a
//       conflict has been resolved (add = SUBTRACT) or when a
//       new conflict has been noted (add = ADD). If add = ADD
//       then a new probe is initiated and sent.
```

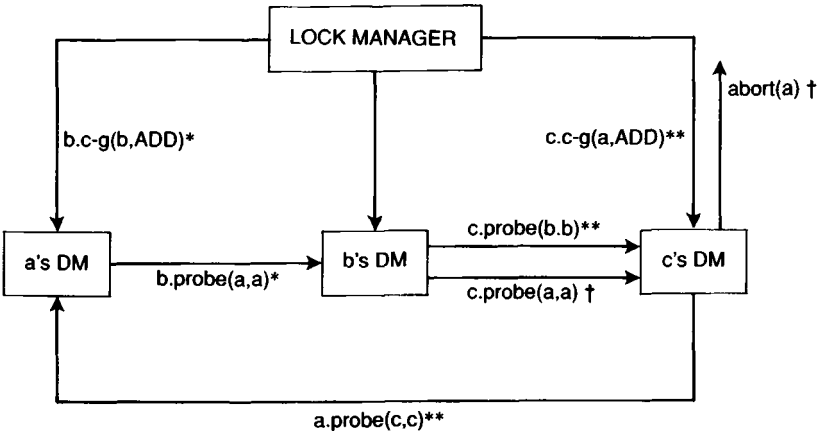


Fig. 1 Deadlock detection: basic algorithm

```
    if (add = ADD) {
        conflicting_trans = list_cons(conflict,conflicting_trans);
        conflict.probe(this,this);
    }
    else conflicting_trans = list_remove(conflict,conflicting_trans);
};
```

2.1.2 probe

The second operation is **probe**. It is called by another deadlock manager when deadlock is suspected. The parameters consist of the *initiator* of the probe and the current choice of victim for deadlock resolution should deadlock be detected. Intuitively, probe messages are initiated each time a new edge is added to the waits-for graph. Probes are sent breadth-first down the waits-for graph from waiting transactions to waited-for transactions until either deadlock is detected or a non-waiting transaction is encountered. A transaction that is not waiting has a deadlock manager with an empty *conflicting_trans* list.

probe never alters the *conflicting_trans* list, but sends **probe** messages to the deadlock manager of every transaction in its list of conflicting transactions.

1 **probe** decides whether or not the transaction it represents is more worthy of abortion than the current victim transaction passed as a parameter. Therefore, it may make its own transaction the current victim.
2 **probe** checks to see if a deadlock cycle is present by seeing if the probe's initiator is in its list of conflicting transactions. If so, it transmits **abort(current_victim)**.
3 If deadlock is not detected then **probe** sends on the probe to all the deadlock managers of transactions in its conflicting transactions list.

4 Note that if *conflicting_trans* is empty then **probe** sends out no messages; its associated transaction cannot be involved in any deadlock.

```
void deadlock_manager::probe(transaction initiator,
                             transaction current_victim) {

// Note: probe is called by another deadlock manager. It updates the
//       current_victim, then detects deadlock or sends on the
//       modified probe to all the deadlock managers in
//       conflicting_trans.

   transaction conflict;
   transaction_list message_list;

   if (this > current_victim)

// Note: here we assume that the deadlock manager identifiers are
//       arranged in ascending order according to the start-times of
//       their associated transactions: youngest high. current_victim is
//       the youngest transaction encountered so far by the probe.

         current_victim = this;
   if (list_element(initiator,conflicting_trans))
      abort(current-victim)
   else {
         message_list = conflicting_trans;
         while (message_list != NULL) {
              conflict = head(message_list);
              message_list = tail(message_list);
              conflict.probe(initiator,current_victim);
         };
   };
};
```

The authors have a Temporal Logic style proof of detection of deadlock which was not able to be included due to lack of space. It can be obtained from the authors.

Example If three edges are added to the waits-for graph indicating that transaction **a** waits-for transaction **b**,**b** waits-for transaction **c** and **c** waits-for **a**, then deadlock is detected as follows:

1 Sooner or later three **change_graph** messages will be sent to the deadlock managers of each of the three deadlocked transactions. This will result in **b** being added to the list of conflicting transactions of **a**'s deadlock manager, **c** being added to **b**'s *conflicting_trans* and **a** being added to **c**'s

conflicting_trans. Also, three probes will be initiated: **b.probe(a,a)**, **c.probe(b,b)** and **a.probe(c,c)**. See diagram 1, messages marked * and **.

2 Sooner or later the initial probe messages will be received and executed. At this point let us assume that **a** is the best choice of abortion victim of the three. The *conflicting_trans* lists will be examined and at least one will be non-empty. For each non-empty *conflicting_trans* list the probe will be sent on. Let us assume that **b.probe(a,a)** is processed after **b.change_graph(c,ADD)**, then **c** will be in **b**'s *conflicting_trans* list when **b.probe(a,a)** arrives. Therefore, the message **c.probe(a,a)** will eventually be sent. See messages marked † in figure 1.

3 When **c.probe(a,a)** arrives at the deadlock manager for **c** it should find **a** in its list of conflicting transactions. Then deadlock will be detected and the message **abort(a)** will be sent. The authors have shown that at least one of the three probes initiated will detect the deadlock cycle. Whichever probe detects deadlock, note that the same victim **a** will be chosen.

4 Deadlock might be detected three times, and three **abort** messages might be sent, but the same victim will be aborted each time.

2.1.3 An Observation

From this example we can see that the transaction scheduler must be able to cope with a number of abort requests for the same transaction. Only one **abort** message will be sent if each probe completes traversing the waits-for graph before the next **change_graph(,ADD)** message arrives. We expect that this will be the normal scenario. The probe-remembering deadlock detection algorithm which we describe later ensures that only one abort message is ever sent for each deadlock cycle.

A number of refinements of the basic algorithm can be seen:

Grouping of change_graph messages: A list of conflict changes could be sent together from the lock manager to the deadlock manager of the same transaction if upon requesting a lock it conflicts with more than one other transaction. This might occur when locks are allowed to be shared. It would not affect the correctness or tidiness of the program.

Reference Counts in the *conflicting_trans* list: Instead of recording duplicates in the *conflicting_trans* list, a reference count could be kept for each conflicting transaction. This would stop duplicate probes being sent. The use of negative reference counts would also allow the algorithm to cope with messages arriving out-of-order: a **change_graph(,SUBTRACT)** message arriving before its corresponding **change_graph(,ADD)** message. Reference counts of the transactions in the list of conflicting transactions could also be used to reduce the number of probes sent: a probe would only be sent if the reference count increased from 0 to 1.

2.2 A Probe-Remembering Algorithm

The probe-remembering algorithm, like the basic deadlock detection algorithm, consists of a collection of instances of deadlock detection managers, one for each transaction in the database management system. Each deadlock manager sends either **probe** messages or **antiprobe** messages to other deadlock managers or **abort** messages to the database management system's transaction scheduler. Once again each message receives no reply, the communication is asynchronous.

In a manner similar to the algorithm of Roesler and Burkhard [14], each deadlock manager can remember all the probes which it has received in a new transaction list of probe initiators. Each **probe** would only be removed by a corresponding **antiprobe** when the conflict has been resolved. Duplicate probes could be allowed to be stored in a deadlock manager's probe list and each antiprobe message would only remove one probe from the list.

We give below the *class* definition of the deadlock manager abstract data type. Each instance of the deadlock manager class will therefore have its own state consisting of two lists of transactions:

1 the transactions for which it is waiting – *conflicting_trans*
2 the initiating transactions of the probes it has received – *probes.*

Access to the state is restricted to the three *member* operations: **change_graph, probe** and **antiprobe.**

The only output from the deadlock detection algorithm occurs when a victim is chosen and a deadlock manager instance sends the message: **abort(victim)** for some transaction **victim.**

Some things should be noted:

- Care has to be taken that exactly the same number of antiprobes is sent as probes in order to maintain the tidiness of the deadlock detection algorithm.
- Because all the probes are remembered, unresolved deadlock will result in the same probes being sent no matter what order the internal detection messages are sent in or what order they arrive in (this is contrary to our basic algorithm where the number of probes sent depends on their order of arrival).
- One advantage of remembering probes is that the number of probes and antiprobes sent can be reduced by a factor of two using an idea of Obermarck [11]: a **probe(s)** is only sent to deadlock manager U if the initiator S is younger than U. Unfortunately, since each probe has to be removed by a corresponding antiprobe, it is not clear whether messages are saved overall. However, it appears that, on average, fewer probes and antiprobes would be sent in our probe-remembering

algorithm than probes sent in the probe-forgetting algorithm. See section 2.3 below.

- Another advantage of the probe-remembering algorithm is that exactly one **abort** message will be sent for each deadlock cycle. In the probe-forgetting algorithm more than one **abort** message could be sent for each deadlock cycle detected.
- Since probes are only sent to a deadlock manager if the probe's initiator is younger than it, this also means that the probe's message can be simplified: only the probe initiator need be included in the parameter since the probe initiator will be the same as the *current_victim*. This also means that a probe can be represented by its initiator transaction. In fact, each deadlock manager remembers the probes it has received as a list of initiating transactions.

```
typedef enum {
        SUBTRACT,
        ADD
} change;

class deadlock_manager {
        transaction_list conflicting_trans;
        transaction_list probes;
public:
        void change_graph(transaction conflict,change add);
        void probe(transaction initiator);
        void antiprobe(transaction initiator);
};
```

2.2.1 change_graph

The first member is **change_graph**. It is called by the lock manager when there has been a change in the waits-for graph; either an edge has been added or one has been removed. As in our basic algorithm, the **change_graph** message is sent to the deadlock manager of the waiting transaction with as parameter the waited-for transaction and a boolean flag indicating whether the edge is to be added or subtracted.

change_graph either adds or subtracts the waited-for transaction to its list of conflicting transactions. However, it reads but does not write to its list of probes received.

If the edge is new:

1 The new waited-for transaction is added to the list *conflicting_trans*.
2 Deadlock is checked by seeing if the new waited-for transaction has initiated a probe in the past which has reached this deadlock manager. If deadlock is found an abort message is sent. The victim is the new

waited-for transaction, since it is by construction the youngest transaction in the deadlock cycle.

3 If no deadlock is found a new probe may be initiated and sent to the waited-for transaction's deadlock manager. The probe is only sent if the deadlock manager's associated transaction is younger than the transaction it is now waiting for. This is the trick of Obermarck [11].

4 If no deadlock is found then each probe which the deadlock manager has received might be sent on to the new waited-for transaction. Each probe is sent on if its initiator is younger than the new waited-for transaction.

If the edge is to be removed:

1 The transaction which is no longer waited for is removed from the list *conflicting_trans*.

2 If the deadlock manager's transaction is younger than the new waited-for transaction then an antiprobe message is sent to the waited-for transaction's deadlock manager.

Example The call **a.change_graph(b,ADD)** causes the transaction **b** to be added to the *conflicting_trans* list of the deadlock manager for **a**.

If **a** is younger than **b**, then the message **b.probe(a)** is sent.

If the *probes* list only contains the transaction $\mathbf{c} \neq \mathbf{b}$ or **a**, and **c** is older than **b** then the message **b.probe(c)** is sent. If however, $\mathbf{b} = \mathbf{c}$ then the only message sent will be **abort(b)**.

```
void deadlock_manager::change_graph(transaction conflict,change add) {

// Note: change_graph is called by the lock manager when either a
//       conflict has been resolved (add = SUBTRACT) or when a
//       new conflict has been noted (add = ADD).

   transaction_list old_probes;
   transaction initiator;

   if (add = ADD) {

// Note: we add the new transaction to the conflicting_trans list.
//       Deadlock is detected or alternatively a new probe might
//       be sent and any relevant existing probes may be sent on.

         conflicting_trans = list_cons(conflict,conflicting_trans);
         if (list_element(conflict,probes)) abort(conflict)
         else {
              if (this > conflict) conflict.probe(this);
```

```
                old_probes = probes;
                while (old_probes != NULL) {
                    initiator = head(old_probes);
                    old_probes = tail(old_probes);
                    if (initiator > conflict) conflict.probe(initiator);
                    }
                }
        }
        else {

    // Note: we remove the transaction from the conflicting_trans list.
    //       An antiprobe is constructed and sent.

            conflicting_trans = list_remove(conflict,conflicting_trans);
            if (this > conflict) conflict.antiprobe(this);
        };
};
```

2.2.2 probe

The second member is **probe**. It is called by another deadlock manager when deadlock is suspected. The only parameter consists of the *initiator* of the probe. As with the basic algorithm, probe messages are initiated when a new edge is added to the waits-for graph. Probes are sent breadth-first down the waits-for graph from waiting transactions to waited-for transactions. A probe is only sent down an edge of the waits-for graph if the probe's initiator is a younger transaction than the transaction at the other end of the edge. All probes that arrive are remembered by adding their initiator to the *probes* list.

probe reads from but never alters the *conflicting_trans* list. However, it does add a transaction to the *probes* list.

1 The probe's initiator is added to the *probes* list.
2 **probe** checks to see if a deadlock cycle is present by seeing if the probe's initiator is in its list of conflicting transactions. If so, it sends a message **abort(initiator)**.
3 If deadlock is not detected then **probe** sends on the probe to all the deadlock managers of transactions in its conflicting transactions list, providing the probe's initiator is younger than the conflicting transaction.

```
void deadlock_manager::probe(transaction initiator) {

// Note: probe is called by another deadlock manager. It detects
//       deadlock or relays the probe to all the deadlock managers in
//       conflicting_trans which have transactions older than the
//       probe initiator.
```

```
        transaction conflict;
        transaction_list message_list;

        probes = list_cons(initiator,probes);
        if (list_element(initiator,conflicting_trans)) abort(initiator);
        else {
                message_list = conflicting_trans;
                while (message_list != NULL) {
                        conflict = head(message_list);
                        message_list = tail(message_list);
                        if (initiator > conflict) conflict.probe(initiator);
                };
        };
};
```

2.2.3 antiprobe

The third member is **antiprobe**. It is called by another deadlock manager when a conflict has been resolved and a probe must be removed. The only parameter consists of the *initiator* of the antiprobe. Antiprobes are sent breadth-first down the waits-for graph from waiting transactions to waited-for transactions. An antiprobe is only sent down an edge of the waits-for graph if the antiprobe's initiator is a younger transaction than the transaction at the other end of the edge. Each antiprobe that arrives causes its initiator to be removed from the *probes* list.

antiprobe reads from but never alters the *conflicting_trans* list. However, it does remove a transaction from the *probes* list.

1 The antiprobe's initiator is removed from the *probes* list.
2 **antiprobe** sends on the antiprobe to all the deadlock managers of transactions in its conflicting transactions list, providing the antiprobe's initiator is younger than the conflicting transaction.

```
void deadlock_manager::antiprobe(transaction initiator) {

// Note: antiprobe is called by another deadlock manager. It relays
//       the antiprobe to all the deadlock managers in
//       conflicting_trans which have transactions older than the
//       antiprobe initiator.

        transaction conflict;
        transaction_list message_list;

        probes = list_remove(initiator,probes);
        message_list = conflicting_trans;
        while (message_list != NULL) {
```

```
            conflict = head(message_list);
            message_list = tail(message_list);
            if (initiator > conflict) conflict.antiprobe(initiator);
        };
    };
```

Example If three edges are added to the waits-for graph indicating that transaction **a** waits-for transaction **b**, **b** waits-for transaction **c** and **c** waits-for **a**, and if **a > b > c** then deadlock is detected as follows:

1 Sooner or later three **change_graph** messages will be sent to the deadlock managers of each of the three deadlocked transactions. This will result in **b** being added to the list of conflicting transactions of **a**'s deadlock manager, **c** being added to **b**'s *conflicting_trans* and **a** being added to **c**'s *conflicting_trans*. Also, two probes will be initiated: **b.probe(a)** and **c.probe(b)**. See figure 2, messages marked *. The former probe will eventually detect deadlock, the latter will not be sent on by its receiving deadlock manager. Therefore, let us consider just the **b.probe(a)** probe.

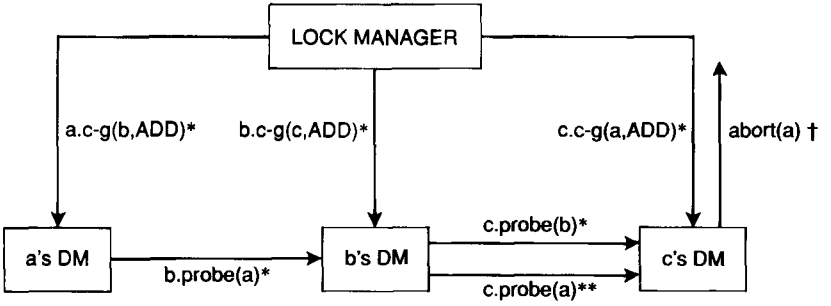


Fig. 2 Deadlock detection: probe remembering algorithm

2 Sooner or later the initial probe message **b.probe(a)** will be received and executed. Transaction **a** will be added to the *probes* list in the deadlock manager for transaction **b**. This probe will be executed either before of after the message **b.change_graph(c,ADD)**:

- If **b.probe(a)** is executed before **b.change_graph(c,ADD)**, then when the latter operation is called the probe initiated by **a** will be sent on to **c**: **c.probe(a)** since **a** will be in **b**'s deadlock manager's *probes* list.
- If **b.probe(a)** is executed after **b.change_graph(c,ADD)**, then when the former operation is called the probe initiated by **a** will be sent on to **c**: **c.probe(a)** since **c** will be in the deadlock manager's *conflicting_trans* list.

See diagram 2, message marked **.

3 Eventually, the probe **c.probe(a)** is executed at the deadlock manager for the transaction **c**. This probe will be executed either before of after the message **c.change_graph(a,ADD)**:

- If **c.probe(a)** is executed before **c.change_graph(a,ADD)**, then when the latter operation is called deadlock will be detected because the transaction **a** will already be in **c**'s deadlock manager's *probes* list.
- If **c.probe(a)** is executed after **c.change_graph(a,ADD)**, then when the former operation is called deadlock will be detected because the transaction **a** will already be in the deadlock manager's *conflicting_trans* list.

4 Therefore, eventually, an abort message **abort(a)** will be sent. See figure 2, message marked †.

A fuller Temporal Logic style proof of deadlock detection is available from the authors.

Further refinements can be made to the probe-remembering algorithm:

Grouping of change_graph Messages: Similarly to the basic algorithm, a list of conflict changes could be sent together from the lock manager to the deadlock manager of the same transaction if upon requesting a lock it conflicts with more than one other transaction. This would not affect the correctness or tidiness of the program.

Reference Counting Conflicting Transactions: Like the basic algorithm, reference counting of the transactions in the list of conflicting transactions could: save storage space, cope with messages arriving out-of-order, and reduce the number probes and antiprobes sent. A probe would only be sent to a conflicting transaction when its reference count increased from 0 to 1. Conversely, an antiprobe would only be sent to a conflicting transaction when its reference count decreased from 1 to 0. The use of negative reference counts would also allow the algorithm to cope with messages arriving out-of-order: a **change_graph(,SUBTRACT)** message arriving before its corresponding **change_graph(,ADD)** message.

Reference Counting Probes: Duplicate arriving probes could be remembered by incrementing a reference count to the probe and antiprobes could simply decrement the appropriate reference count. When the reference count was zero the probe could be forgotten. Also, a further antiprobe would only be sent on when the reference count reached zero. If messages might arrive out-of-order between two deadlock managers, then the algorithm may stumble because a *probe* message might arrive after its corresponding *antiprobe* message. The solution is to allow the probe reference counts to take all possible negative as well as positive values (a zero value is not needed!).

Note that this might require a deadlock manager to survive the end of its transaction in order to be able to pair up its *probe* and *antiprobe* messages. Each time the probe count becomes positive the probe would be sent on to the relevant waiting transactions. Each time the reference count decre-

ments to zero an antiprobe would be sent on to the relevant waiting transactions.

2.3 *The Efficiency of the Algorithms*

The efficiency of distributed algorithms depends on the communication network as much as the constituent processors. Some communication networks naturally provide a broadcast facility but others are implemented by point-to-point communication. The massively parallel architectures tend to have networks which more closely resemble the second variety. We shall therefore assume that broadcast messages ought to be avoided for efficiency's sake.

The time taken to execute deadlock detection code on any one processor is negligible compared with the typical latency time of a network message. Also, the amount of memory used by a deadlock detection algorithm is usually minimal. Therefore, the efficiency of deadlock detection algorithms on distributed-store multiprocessors can be measured according to two key criteria:

- the percentage of network bandwidth used by the deadlock detection process under a range of loads: this can be measured as the average message size times the total number of messages sent
- the time taken to resolve any particular deadlock: this can be measured by the maximum number of network messages in any one thread of execution involved in detecting a particular deadlock cycle.

Therefore, we shall concentrate on these two efficiency criteria.

The number of network messages sent by our deadlock detection algorithms and the time to detect deadlock will depend on the number of conflicting transactions of which each transaction is aware. What is certain is that for each new transaction dependency noted by the lock manager two messages will have to be sent to the same deadlock detection manager:

change_graph(,ADD)

and

change_graph(,SUBTRACT).

In order to be able to do calculations we make the following simplifying assumptions:

1 After each **change_graph** message is sent, all deadlock probes and antiprobes are sent, arrive and are executed before the next **change_graph** message is sent. In general, without this assumption, it is impossible to deal with all the different cases of message arrival orderings. The assump-

tion means that the deadlock detection process is only given one stimulus at a time. In the probe-remembering algorithm, the average total number of probes (and therefore antiprobes) sent will be the same with or without this assumption. In the basic algorithm, the number of probes sent could be higher without this assumption, e.g. the same deadlock cycle could be detected by many probes.

2 The average number of transactions for which a transaction is waiting at any time (called E) is small i.e. $E < 1$. This is equivalent to saying that the transaction conflict rate is low. Without this assumption it is difficult to put a bound on the number of probes generated by the basic deadlock detection algorithm.

3 The average number of transactions for which a transaction is waiting from a given time until the end of the transaction (called F) is small i.e. $F < 1$. Note that $F \geq E$ since the transaction might wait for new transactions in the future.

The Total Number of Messages Sent: The number of probes sent will depend on the detection algorithm and the number of conflicting transactions in each deadlock detection manager:

The Basic Algorithm: Considering the abstract waits-for directed graph of transactions, a probe is sent down each edge reachable from the probe initiating transaction vertex at the time of the probe's arrival at each vertex. Since probes are forgotten, any new edges added to the waits-for graph after the probe has finished traversing the graph will not be traversed by that probe. One probe is always sent by the basic detection algorithm. However, when that probe arrives, the number of further copies of that probe sent is equal to the number of graph edges reachable from that vertex. Therefore, the total number of probes sent is $1 + P$ where P is the average number of edges reachable from a vertex. Let p_i equal the probability that a transaction is waiting for i other transactions, then p_i equals the probability that there are i out edges from a vertex. Notice that $\sum_{i \geq 0} p_i = 1$. $P = \sum_{i \geq 0} i \cdot p_i \cdot (P + 1)$ which can be rearranged as $P = \frac{E}{1 - E}$ where $E = \sum_{i \geq 0} i \cdot p_i$ is the expected number of out edges at a vertex. The second assumption ensures that $E < 1$. We conclude that the total number of probes sent for each transaction dependency is $1 + P = \frac{1}{1 - E}$, and the total number of messages sent is $2 + \frac{1}{1 - E}$.

The Probe Remembering Algorithm: Considering the abstract waits-for graph again, a probe is sent down each edge reachable from the initiating vertex at any time in the future providing the edge is from a higher transaction identifier to a lower transaction identifier. Probes are remembered until expressly forgotten by the arrival of an antiprobe. Let q_i equal the probability that a transaction is or will wait for i other transactions, then q_i equals the probability that there are i out edges from a vertex.

Notice that $\sum_{i \geq 0} q_i = 1$ and $F = \sum_{i \geq 0} i \cdot q_i$. We can assume that a probe is sent down a graph edge approximately half the time, so our calculations are as follows: the number of probes Q equals $\sum_{i \geq 0} i \cdot \frac{q_i}{2} \cdot (Q + 1)$ which can be rearranged as $Q = \frac{F}{2 - F}$. The total number of antiprobes sent is the same. Therefore, the total number of messages sent for each transaction dependency is $1 + \frac{2 \cdot F}{2 - F} = 2 + \frac{F}{1 - F/2}$.

Comparison: It can be seen from this analysis that it is difficult to choose between the efficiency of the two algorithms: the basic algorithm sends fewer messages in that it forgets probes immediately, the other sends fewer messages in that it only sends them from a higher transaction identifier to a lower transaction identifier. However, given the assumption that each transaction always asks for all its locks when it starts, this will mean that $E = F$. It is easy to show that $2 + \frac{1}{1 - E} > 2 + \frac{E}{1 - E/2}$ for all E. Therefore, we conclude that for transactions which immediately ask for all their locks, the probe remembering algorithm generates fewer messages than the basic algorithm.

The Response Time To Detect Deadlock: It can be seen that the probe-remembering algorithm detects deadlock more quickly: in any cycle the last probe sent by the basic algorithm must be sent along each edge of the cycle, but only along half the edges of the cycle in the probe-remembering case. This faster response time might prove invaluable. Therefore, the number of sequential messages sent to detect deadlock in the probe-remembering algorithm is on average half the number of sequential messages sent to detect deadlock in the basic algorithm.

Conclusions: When considering the number of sequential messages required in order to detect deadlock, the probe-remembering algorithm out-performs the basic detection algorithm. With regard to the total number of messages sent, the algorithm which is the more efficient depends on the average timing of transaction lock requests.

3 Comparisons with other Algorithms

3.1 Obermarck

Obermarck's algorithm [11] is roughly as follows. The checker starts by local managers gathering local knowledge. For a synchronised period messages are sent concerning the suspected global cycles and then the local managers add the knowledge they received during the message period to their local knowledge, looking for local cycles and suspected global cycles.

Another agreed synchronised period of message passing between the local managers follows. So the algorithm alternates local checks and synchronised message passing periods until no more messages are sent.

When no more messages are sent all deadlock has been detected. The local managers then forget all they know, waiting for the next call of the algorithm. All detected local deadlock cycles are broken by aborting and retrying a victim before the next message phase starts.

Our algorithms have similarities with the algorithm of Obermarck [11]. There are, however, significant differences:

- Obermarck assumes that each transaction is requesting at most one access at any time, we allow a transaction to request more than one resource access at once.
- Obermarck's algorithm detects all deadlock in the system *up to a given time* by synchronising a sequence of local deadlock checks and message passing between the checks. After the deadlock process each local manager forgets all that it learned during the process. Our algorithms detect deadlock *as it arises* and each local deadlock manager contains information as up-to-date as possible: the deadlock managers working asynchronously.
- We provide in our basic algorithm an efficient removal of information about committed/aborted transactions from the deadlock detection process, since probe information is either forgotten immediately or explicitly removed.

3.2 *Chandy, Haas and Misra*

A distributed deadlock detection protocol for shared resources was described in [3]. They also described a detection protocol for detecting deadlock in communicating processes. The protocol of [3] assumes deadlock detection managers for each network node. However, these managers remember for each local transaction all the transactions which are known to depend on it, even indirectly.

The principle differences of this from our algorithm are:

- The algorithm of [3] assumes that a local deadlock detector sends out a deadlock checking probe whenever it *suspects* deadlock. This may result in many probes being sent by a suspicious transaction, and more so when the system is under heavy loading. Our algorithms initiate only one probe message each time a deadlock cycle could be completed. Furthermore, our probe-remembering algorithm issues the same number of probes no matter what order the messages arrive.
- In [3], each manager keeps information about all the transactions that it is told about. We make sure that the deadlock detection managers

forget about transactions which are not in a direct waits-for relationship with its transaction. So, our algorithm restricts the storage overheads.

- The previous difference means that when a transaction is either committed, aborted or restarted it is easier with our algorithm to inform the relevant detection managers, i.e. fewer network messages need to be sent.

It should be noticed that both our algorithms and theirs detect all deadlock and false deadlock is only possibly detected when at least one transaction in the detected deadlock cycle has been aborted since the start of the detecting process.

3.3 Elmagarmid, Soundararajan and Liu

Elmagarmid, Soundararajan and Liu [5] present an algorithm which groups the control of "interacting" transactions dynamically as transactions compete for resources. One controller will control a number of transactions on different processing elements or sites, which have interacted with each other until they interact no longer. The idea is to amalgamate control when resource requests conflict and fork two controllers when two subsets of transactions form a partition and no longer interact. The algorithm is flexible but entails control of a transaction being switched across the network to sites of interacting transactions, all the control information being sent as a message and enough information left behind to tell other inquiries where the control was sent.

Apart from the extra complications and storage overheads, concurrency can be impaired. If a transaction consists of a large number of concurrent resource requests then each request can be sent in parallel with instructions to reply to the source processor. If, however, the transaction's control migrates across the network, each reply will have to chase the control mechanism, following like a paper chase. This will significantly increase the number of network messages per transaction. Only if the resource requests are sequential will the replies be given up-to-date return addresses.

In our algorithms, communication only occurs between the deadlock managers of interacting transactions. This can be seen to be similar to their grouping of interacting transaction management.

We summarise the difference between their algorithm and our algorithms:

In [5], the destination of replies to resource requests can be out-of-date because the transaction's controller has moved a number of times. Each out-of-date reply destination causes extra network messages, the cost being proportional to number of times the transaction's controller has moved since the request was sent. In our algorithm the transaction's controller location is fixed, avoiding disastrous 'paper chases'.

In [14] a distributed deadlock detection algorithm is presented which is similar in a number of aspects to our probe-remembering algorithm:

- Deadlock detection is initiated each time locks conflict.
- The algorithm runs the same code on each of the sites.
- The algorithm is essentially object-oriented.

However, there are significant differences:

Lock Managers Involvement as well as Transaction Managers: The involvement of two types of deadlock detection manager in the transmission of probes means that twice as many deadlock detection agents participate. Therefore, their algorithm will generate twice as many messages as our probe-remembering algorithm.

A Probe-Forgetting Alternative: Our basic detection algorithm offers a simpler probe forgetting algorithm which is easier to implement and which is still more efficient than theirs.

4 Conclusions

Firstly, the algorithms proposed to deal with distributed deadlock in the literature can themselves have different degrees of concurrency.

The algorithms of Ho and Ramamoorthy [9] involve the choice of one or more central control centres where global deadlock detection is collected and acted upon. The centre(s) send out information to all sites and wait for all the responses. These algorithms send few but lengthy messages across the network, but they are not truly distributed.

The interesting algorithm of Belik [1] assumes that each time a resource request is queued, and there is no deadlock, all sites or processors have to agree to allow one of them to update its dependency graph and all then wait for the results of this update. This surely is not truly distributed. Belik argues that his algorithm does not deteriorate under heavy load, but the normal communication overheads seem too high.

The algorithms of Obermarck [11], Bracha and Toueg [2] and Gafni [6] involve the coordination of message passing within the deadlock detection process. The fact that any coordination is required indicates that these algorithms are not truly distributed. For instance, communication time delays may vary between the sites. In this case the speed of the algorithm depends on the longest communication time delay, even if deadlock might just be caused between two sites which are very close, e.g. two processors within the same cabinet.

On the other hand, both our algorithms and that of Sinha and Natarajan [15], modified and corrected by Choudhary et al. [4] require no central control or global synchronisation.

Secondly, there are a number of approaches as to when a deadlock detection process should be started.

One approach says "when a process or transaction is worried that it has been waiting a long time, it should send a probe out to see if there is deadlock". The problem with this approach [3, 2] is that when the system is working hard it will receive more such worried probes and have less time to deal with them.

A second approach says "let us detect all deadlock at regular time periods, or when some global sensors suspect a lot of deadlock". This approach, which is the approach of Obermarck [11], tends to lead to attempts to synchronise deadlock-finding iterations at each of the processors in parallel. The synchronisation of time periods is to allow message passing phases between the deadlock detection iterations. The algorithm provides that all deadlock is detected by the time of the end of the last iteration. Performing deadlock detection at fixed intervals to detect all deadlock means that deadlock is not resolved as soon as possible thus escalating the amount of deadlock on the system.

Our approach says "whenever we know of a new dependency between two processes or transactions, we do as much of the deadlock check as is necessary". This ensures that there is no extra work when the system is under high load and waiting processes might get paranoid about being deadlocked. Also it ensures that deadlock is not left blocking the system until some regular or imprecise decision is made to test for all deadlock at once. The algorithms of Belik [1], Sinha and Natarajan [15] and Choudhary et al. [4] also adopt the same approach.

We further believe that our algorithms exhibit a simplicity which allows semi-formal proofs of their correctness.

References

1 BELIK, F., "A Distributed Deadlock Avoidance Technique", *Lecture Notes in Computer Science*, Vol. 312, J. van Leeuwen (ed.) Springer Verlag, pp. 144–154, 1987.

2 BRACHA B. and TOUEG S., "A Distributed Algorithm for Generalised Deadlock Detection", *Proceedings Third Annual ACM Symposium on Principles of Distributed Computing*, Vancouver, August 27–28, pp. 285–301, 1984.

3 CHANDY, K.M., HAAS, L.M. and MISRA, J., "Distributed Deadlock Detection", *ACM Transactions on Computer Systems*, Vol. 1, no. 2, pp. 144–156, 1983.

4 CHOUDHARY, A.N., KOHLER, W.H., STANKOVIC, J.A. and TOWSLEY, D., "A Modified Priority Based Probe Algorithm for Distributed Deadlock Detection and Resolution", *IEEE Transactions on Software Engineering*, Vol. 15, no. 1, pp. 10–17, 1989.

5 ELMAGARMID, A.K., SOUNDARARAJAN, N. and LIU, M.T., "A Distributed Deadlock Detection and Resolution Algorithm and its Correctness Proof", *IEEE Transactions on Software Engineering*, Vol. 14, no. 10, pp. 1443–1452, 1988.
6 GAFNI, E., "Perspectives on Distributed Network Protocols: A Case for Building Blocks", *IEEE MILCOM*, 1986.
7 GLIGOR, V.D. and SHATTUCK, S.H., "On Deadlock in Distributed Systems", *IEEE Transactions on Software Engineering*, Vol. 6, no. 5, pp. 435–440, 1980.
8 HILDITCH, A.S. and THOMSON, C.M., "Distributed Algorithms for Detecting Distributed Deadlock", University of Manchester Computer Science Department Technical Report UMCS-91-3-2, 1991.
9 HO, G.S. and RAMAMOORTHY, C.V., "Protocols for Deadlock Detection in Distributed Database Systems", *IEEE Transactions on Software Engineering*, Vol. 8, no. 6, pp. 554–557, 1982.
10 LAMPORT, L., "Time, Clocks and the Ordering of Events in a Distributed System", *Communications of the ACM*, Vol. 21, no. 7, pp. 558–565, 1978.
11 OBERMARCK, R., "Distributed Deadlock Detection Algorithm", *ACM Transactions on Database Systems*, Vol. 7, no. 2, pp. 187–208, 1982.
12 PNUELI, A., "The Temporal Semantics of Concurrent Systems", *Theoretical Computer Science*, Vol. 13, pp. 45–60, 1981.
13 RAYNAL, M., *Distributed Algorithms and Protocols*, Wiley, 1988.
14 ROESLER, M. and BURKHARD, W.A., "Deadlock Resolution and Semantic Lock Models in Object-Oriented Distributed Systems", *Proc. SIGMOD Int. Conf. on Management of Data*, Chicago, pp. 361–370, 1988.
15 SINHA, M.K. and NATARAJAN, N., 'A Priority Based Distributed Deadlock Detection Algorithm', *IEEE Transactions on Software Engineering*, Vol. 11, no. 1, pp. 67–80, 1985.

Biographies

Steve Hilditch

Steve Hilditch had a PhD in pure mathematics (Algebraic Topology) and a BA in Theology before turning to Computer Science in 1987.

From 1987 until 1991 he worked in Manchester University on joint projects with ICL: Flagship and EDS, both distributed-store multiprocessors. During this time he completed his MSc in Computer Science. After a year learning French, he joined ICL Mid Range Systems Division (MRSD) in August 1992. He is a member of a Design Authority team within MRSD.

Tom Thomson

Tom Thomson is a Chartered Mathematician and a Chartered Engineer. A graduate of Oxford and Bristol Universities, he worked at the Rutherford Laboratory, at English Electric, at the University of East Anglia, and at CTL before joining ICL in 1971. After 4 years in ICL Dalkeith working on communications processors he joined ICL Kidsgrove and spent 10 years working on mainframe systems. For the last 7 years he has been the senior system designer in ICL's Fifth Generation and Parallel Systems group in Manchester, working on research collaborations under the Alvey and Esprit programmes and on the exploitation of this research in ICL products.

Book reviews

*OPEN*framework: *The Systems Architecture – An Introduction*, Ron Brunt and Andrew Hutt (Eds), Prentice Hall, 1992. ISBN 013 560186X pp. 60 £22.95

This book gives an overview of the OPEN*framework* architecture from ICL. Its chapters precis the components of OPEN*framework:* Perspectives (of users, applications developers ...), Qualities (availability, usability), Elements (user interface, information management, networking services ...), Specializations and Advice and Guidance (on business and technology trends ...). These are all to be expanded in further books in the series.

On first reading I found the rather general statements of the first two chapters difficult to get to grips with – a simple example or two or a clearer definition of what was meant by the word 'architecture' would have helped. However, having found the first sentence of Chapter 7, "The overall objective of OPEN*framework* is to provide know-how and guidance on best practice to those concerned with ensuring that information systems serve the business needs of an enterprise", I had more confidence in the practical possibilities for OPEN*framework* and found a backtrack through the previous chapters worthwhile.

The book has a simple, coherent structure and the diagrams should play an important role in illustrating the concepts.

Occasionally, I found the diagrammatic notations misleading. An early example on the Perspectives seems to imply that Application Developers talk to Enterprise Management but not to users. "Aha!" I thought. "No wonder systems are often user unfriendly." However the text immediately below the diagram belies this by stating that 'application developers need to work with users.'

As a footnote, and in view of the key role of the diagrams, I would have preferred them to be larger and more carefully drafted.

The chapter on Qualities could be used as a sophisticated checklist, to be considered from the inception of a project. For example, the authors emphasise the need to set quality targets at an early stage, and the importance of service level agreements, with an aim to develop systems which are neither over- nor under-engineered. The further reading promised should expand on the standards and products which can be used, but readers will find

helpful even the precised version here. The reference models in particular give some structure to the potentially messy business of discussing qualities such as usability. The inclusion of potential for change as a Quality is particularly interesting.

The claims for OPEN*framework* are big. Coming from an environment in which we have decided that concentration on "UNIX" and PC/MAC platforms is just about all we can cope with if we are to have an integrated but distributed system, I have great respect for the authors of this book who are prepared to tackle more heterogeneous environments, for example those involving different models of filestore. I have some doubts that products following protocols they recommend (for example OSI's FTAM) can at this moment deliver all that is required but maybe I will be proved wrong when reading the books on the Elements. There is in any case a healthy sprinkling of pragmatism (for example SNA and TCP/IP in the Networking Element), which bodes well for some practical solutions.

OPEN*framework* would seem to be truly vendor-neutral. It is not about ICL products and I came across only one "ICLism" (a reference to FTF) in the book.

The potential for OPEN*framework* to make a radical improvement in the design and use of information systems is exciting. For example, on security the promise of distributed security services, such as authentication, fills me with anticipation. I look forward to seeing the further books in the series and finding out if in fact there are enough building blocks (including genuine products that work) in the marketplace to make the potential a reality.

I would recommend the book to those with some strategic responsibility in information systems who feel the need to sit back and try to unravel where their organisation's information system is, where it may be going and what needs to be done to set it up so as to be flexible to change. The further books in the series would of course need to be read before practical steps could be taken.

Annette Haworth
Annette Haworth is Director of Computer Services at the University of Reading. She has a particular interest in networking and is a member of the University Funding Council Advisory Committee on Networking and recent Chairman of the Inter-Universities Networking Committee.

X/Open and Open Systems by Colin Taylor. The X/Open Company Limited: 117 pp ISBN 1–872630–55–3 £20: from bookshops and directly from X/Open Company (Publications), P.O. Box 109, Penn, High Wycombe, Bucks HP10 8NP, UK.

The importance of Open Systems to both users and suppliers is increasingly realised. The X/Open Company is a keystone in the Open Systems movement: X/Open offers an integrated set of open standards drawn from other bodies, and knowledge of its role and activities is therefore of fundamental importance to anyone who wants an understanding of the modern IT industry.

This very readable book describes the objectives, origin, present structure, current coverage and future tasks of the X/Open Company. The book is highly authoritative, as might be expected both from its being an official publication of X/Open and from its author, who has been ICL's Technical Manager for X/Open from its inception. It covers a wide subject both clearly and succinctly: so providing a lot of information in an easily assimilated form.

It starts with an introductory chapter on Open Systems, explaining what they are and the business reasons behind them. This chapter of itself could be a reason to acquire the book. The origins, benefits and some of the issues on Open Systems are clearly laid out and the role of X/Open is established in this context. The cover here is good, thought I would have liked to see more emphasis in this section on the aspect of public change control.

The author then takes us through the history of X/Open from its formation to the present day. In the course of this the various current structures and products are brought out, in the context of how they came about. This approach works well, even for someone who only wants to know the organisation and its work as it is today, as the history provides a good way of explaining why things are the way they are, and how the various business and technical interests have been taken care of. In fact only chapters 2, 3 and 6 are really history, chapter 4 explains XPG3 – the set of interfaces and conformance level which is still the basis of what nearly all users will have installed. Chapter 5 then sets out the fundamental structure of the X/Open Company.

From chapter 7 (p. 31) onwards we see the increasing emphasis on the user input – and the increasing user support by procurement policies, and the way that X/Open has widened its original remit to cover the general field of Open Systems and the issues of branding/conformance.

Chapter 12 addresses the issue of conformance and branding of products supporting the interfaces and in chapter 13 the challenges of conformance testing and branding of applications using the interfaces are set out. In the latter the issues of conformance of products supporting the interfaces are

rightly painted by contrast as "solved" – but the reader should not think this means "no problems" even for the former. Most of the relevant issues are indeed covered in chapter 12, but a little more could have been said here or later in the book about interoperability and the relation with SPAG's Process to Support Interoperability (PSI) – SPAG is the European based Standards Promotion and Application Group. (Note that the second book in the X/Open series is entitled "X/Open and Interoperability".)

Chapters 16 onward bring us up to date with the current specification level, XPG4, and what is coming after this, and with the latest developments in X/Open Company structure, which offer those with less resources than the major shareholders more opportunities to participate in areas of particular interest. Some of the steps forecast in this section have now taken place – X/Open is a dynamic organisation and a second edition of this book will no doubt be needed within a year or two! The last chapter (20) summarises X/Open's achievements and points the way forward.

There is a foreword by Andrew Roberts, at the time of publication Chairman of X/Open and Managing Director of ICL's Mid Range Systems Division.

Colin's colleagues will know him as an enthusiastic advocate of X/Open for many years. Not only has he been ICL's Technical Manager since the word go but his advocacy goes back before its conception! He pioneered the strategy of adopting UNIX as a standard operating system for commercial mid range systems in ICL (before this was company policy) and ICL under Robb Wilmot subsequently instigated the X/Open collaboration. X/Open's results have since become extremely important to ICL not only for its mid-range systems but more recently for its corporate systems business too (ICL VME was the first mainframe operating system to achieve X/Open XPG branded status). Earlier, Colin had a long history of work in producing operating systems providing compatibility across a variety of different hardware. With this book he has made another major contribution on Open Systems to the IT community.

His book could be advocated as a set book for Computer Science courses, and as course material for core training in major companies. It is written in a factual way, and avoids getting involved in areas of political controversy. Someone looking for a bloodthirsty account of the "UNIX Wars" will not find it here (indeed the author explicitly says in chapter 11 "much has been written on this industry split ... and it is not proposed to discuss it again here but only to address the implications for, and their effect on, X/Open"). The book thus avoids getting dragged into taking sides in such arguments, but the reader will be able to see where such issues lurk beneath the surface.

The book is number one in X/Open's new series "X/Open in Action". If the monographs on specific subjects which follow it are as clear and as readable, then we have much to look forward to.

Brian G Millis
The writer worked for ICL for many years. He managed the evolution of ICL's Information Processing Architecture over several years of adoption of Open Systems standards. He is now working as a consultant to ICL, and also on Open Systems to other clients.

New Technology and Practical Police Work: The social context of technical innovation by Stephen Ackroyd, Richard Harper, John A Hughes, Dan Shapiro and Keith Soothill. Open University Press Buckingham and Philadelphia (pp. 178) ISBN 0–335–09459–7 (hb) (£37.50) ISBN 0–335–09458–9 (pb) (£15.99)

This book is written in academic style and is intended for graduate students in management or IT related subjects. It examines "some of the ways in which the police have made use of IT" and investigates "why some information systems 'succeed' in contributing to performance and gaining the acceptance of users, while others 'fail' to do so." This book also has potential value to IT specialists within the police service and their suppliers.

The authors are, or were at the time of the study, members of Lancaster University, and their research force was the Lancashire Constabulary, with whom the University has a long established relationship. The research force is a major police customer for ICL in the UK. The research was funded by the Joint Committee of the Science and Engineering Research Council and the Economic and Social Research Council as part of their initiative on The Successful Management of Technological Change.

The study pays considerable attention to the management revolution within the police, and the change from custodial management (an apt name referring to the protectionist style of past ways of doing things) to managerialism, the new style with dependencies on the prompt availability of information to support decision making. The preface makes reference to general public impressions about the use of technology by police but "So far, at least as information systems are concerned, the reality that we describe is patchy, with some effective systems but other cumbersome, limited and even limiting ones. Yet we must point out at once that Lancashire is in fact one of the leaders among British police forces in the adoption and effective use of information technology ..." The early chapters review recent police history from the mid 1960s up to the present day (1991). To students of this subject chapters 2 and 3 provides a concise view, all the more valuable for coming close to the present day.

Subsequent chapters look at specific applications which have been developed, the context in which development and implementation took place and an assessment of the relative 'success' of the projects. The case study approach loses some impact as the studies relate to projects which occurred in different

technological eras as well as in different stages of the research force's implementation experience. The first development system was an example of using 'data processing' techniques to provide management information, but the application was too 'technical' for end-users to exploit fully. The second case, built on the experience of the former, exploited 'information processing' and used tools which allowed prototyping. The later project addressed the weakness identified by the authors who challenged the original 'specification approach' and found it made users vulnerable to, and dependent upon the designers and builders of systems.

A chapter examines the impact of technological innovation and, drawing on previous research, concludes "The impact of IT (on the police service) has been so poor that in the UK there may be an over-reaction in which the technology will be seen as totally useless ..." If this is truly the case, it is surprising that a service which has grasped other technologies, telecommunications, forensic support, Vascar and specialist equipment for surveillance to list just a few, has failed so badly in its use of information technology. There are clear examples of success across the service. The Police National Computer has made central records widely and securely available 'on demand'. The Holmes system introduced to the service in 1986–7 is described in some detail and various critical comments made, specifically about the 'linking' of incidents, the effort required in data capture and the 'primitive' nature of the application. Thankfully the authors acknowledge that '... honour is more suitably served if everyone simply agrees that Holmes is better than the manual system... while recognizing it is not being used to its full potential". It is not unusual to find technology being used below its potential, although there will be pockets of excellence where greater exploitation of the system is achieved.

The structure of the police service is likely to inhibit innovation across all forces. Best practise is disseminated, but relatively slowly. While individual forces retain the freedom to choose how they implement IT, the adoption of best practise supported by IT will be dependent upon capital programmes and the availability of resource. The speed of technological innovation is currently moving faster than the police services ability to exploit them. The consequence is a proportion of forces are always struggling to catch up.

The authors recognise that their offering addresses a part only of the subject but hope that it provides a contribution towards understanding in this area. This it surely does. New Technology and Practical Police Work addresses complex organisational issues and the use of IT; it contains some rich seams, though the extraction of the lessions will prove a challenge.

S.B. Southerden
The reviewer has worked for ICL as an industrial consultant since 1987 and currently manages the development of the Intelligence Analyst Workbench (see paper in *ICL Tech. J.*, Vol. 8 no. 2 1992). Before joining ICL he served 16 years as a police officer with Kent County Constabulary with a secondment to the University of Kent where he took a degree in public administration and management.

THE ICL COMPUTER USERS ASSOCIATION

There are now over 2000 organisations which are members of ICL user groups and are affiliated to the ICL Computer Users Association. These range from large government departments, local authorities, public utilities, large corporate companies through to smaller private companies and individuals.

The Association has over 40 user groups meeting regularly, some on a regional basis, others because they have similar professional interests, while other users meet because they share a common application or operating system and wish to gain maximum benefit from their investment.

The benefits of membership of a user group can be listed as follows:

- Exchange of views and experiences which help companies to exploit their investment in IT systems to the limit of their potential.
- Regular communication with the supplier to influence the future developments of products and services.
- Economical training and up-to-date information on the latest releases of software and documentation.
- Discounts on many of the volume products.
- A regular news magazine.
- Personal development of active members and new informal contacts through social events.

The CUA holds a major conference and exhibition each year where the latest products can be seen. In addition, much technical information is exchanged and interesting presentations for a whole range of users are given. This is arranged via "streams" ranging from IT management, through to application development, service delivery, end users and special interests.

Members join the CUA and any number of individual user groups in which they have an interest. In this way they can maximise the value of the investment to their organisation.

John Gardner, Chairman of ICL (UK) plc, and his staff are fully supportive of the CUA, and the user groups that form it value the relationship that exists between the CUA and ICL (UK) and strongly recommend that all users seriously consider becoming members.

For more information about the ICL Computer Users Association contact: **CUA Communications Manager, PO Box 18, Stevenage, Hertfordshire SG1 2PU. Tel: (0438) 747313; Fax: (0438) 314858.**

For ICL staff contact: **CUA Liaison Department, ICL UK, Waterside Park, Cain Road, Bracknell, Berkshire RG12 1FA. Tel: (0344) 711000; Fax: (0344) 711746.**

ICL TECHNICAL JOURNAL

Guidance for Authors

1. CONTENT

The *ICL Technical Journal* has a large international circulation. It publishes papers of high standard having some relevance to ICL's business, aimed at the general technical community and in particular at ICL's users and customers. It is intended for readers who have an interest in the information technology field in general but who may not be informed on the aspect covered by a particular paper. To be acceptable, papers on more specialised aspects of design or applications must include some suitable introductory material or reference.

The Journal will usually not reprint papers already published, but this does not necessarily exclude papers presented at conferences. It is not necessary for the material to be entirely new or original. Papers will not reveal matter relating to unannounced products of any of the ICL Group companies.

Letters to the Editor and reviews may also be published.

2. AUTHORS

Within the framework defined by §1 the Editor will be happy to consider a paper by any author or group of authors, whether or not an employee of a company in the ICL Group. All papers are judged on their merit, irrespective of origin.

3. LENGTH

There is no fixed upper or lower limit, but a useful working range is 4000–6000 words; it may be difficult to accommodate a long paper in a particular issue. Authors should always keep brevity in mind but should not sacrifice necessary fullness of explanation to this

4. ABSTRACTS

All papers should have an Abstract of not more than 200 words, suitable for the various abstracting journals to use without alteration. The Editor will arrange for each Abstract to be translated into French and German, for publication together with the English original.

5. PRESENTATION

5.1 Printed (typed) copy

Two copies of the manuscript, typed $1\frac{1}{2}/2$ spaced on one side only of A4 paper, with right and left margins of at least 2.5 cms, and the pages numbered in sequence, should be sent to the Editor. Particular care should be taken to ensure that mathematical symbols and expressions, and any special characters such as Greek letters, are clear. Any detailed mathematical treatment should be put in an Appendix so that only essential results need be referred to in the text.

5.2 Diagrams

Line diagrams will if necessary be redrawn and professionally lettered for publication, so it is essential that they are clear. Axes of graphs should be labelled with the relevant variables and, where this is desirable, marked off with their values. All diagrams should have a caption and be numbered for reference in the text, and the text marked to show where each should be placed – e.g. "Figure 5 here". Authors should check that all diagrams are actually referred to in the text and that all diagrams referred to are supplied. Since diagrams are always separated from their text in the production process these should be presented each on a separate sheet and, *most important*, each sheet must carry the author's name and the title of the paper. The diagram captions and numbers should be listed on a separate sheet which also should give the author's name and the title of the paper.

5.3 Tables

As with diagrams, these should all be given captions and reference numbers; adequate row and column headings should be given, also the relevant units for all the quantities tabulated. Short tables can be given in the text but long tables are better submitted on separate sheets and these, as for diagrams, must carry the author's name and the title of the paper.

5.4 Photographs

Black-and-white photographs can be reproduced provided they are of good enough quality; they should be included only very sparingly. Colour reproduction involves an extra and expensive process and will be agreed to only exceptionally.

5.5 References

Authors are asked to use the Author/Date system, in which the author(s) and the date of the publication are given in the text, and all the references are listed in alphabetical order of author at the end.

e.g. in the text: "... further details are given in [Henderson, 1986]"

with the corresponding entry in the reference list:

> HENDERSON, P. Functional Programming, Formal Specification and Rapid Prototyping. *IEEE Trans. on Software Engineering* SE-12, 2, 241–250, 1986.

Where there are more than two authors it is usual to give the text reference as "[X et al ...]".

Authors should check that all text references are listed, and only text references; references to works not quoted in the text should be listed under a heading such as "Bibliography" or "Further reading".

5.6 Style

A note is available from the Editor summarising the main points of style – punctuation, spelling, use of initials and acronyms etc. – preferred for Journal papers.

6. REFEREES

The Editor may refer papers to independent referees for comment. If the referee recommends revisions to the draft the author will be asked to make those revisions. Referees are anonymous. Minor editorial corrections, as for example to conform to the Journal's general style for spelling or notation, will be made by the Editor.

7. PROOFS, OFFPRINTS

Printed proofs are sent to authors for correction before publication. Authors receive 25 offprints of their papers, free of charge, and further copies can be purchased; an order form for copies is sent with the proofs.

8. COPYRIGHT