ICL TECHNICAL JOURNAL

Volume 7 Issue 4 November 1991

Published by
International Computers Limited
at
Oxford University Press

ICL TECHNICAL JOURNAL

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

Editor
J.M.M. Pinkerton
ICL, Lovelace Road, Bracknell, Berks RG12 4SN

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.

1991 subscription rates: annual subscription £50 UK and Europe and $110 rest of world; single issues £30 UK and Europe and $66 rest of world. Orders with remittances should be sent to the Journals Subscriptions Department, Oxford University Press, Pinkhill House, Southfield Road, Eynsham, Oxford OX8 1JJ.

Printed by H Charlesworth & Co Ltd, Huddersfield ISSN 0142-1557

ICL TECHNICAL JOURNAL
Volume 7 Issue 4

Contents

Résumés

Phil Barthram et Tim Howling
ICL Mid Range Systems Division, Basingstoke, Royaume-Uni
Gestion de distribution — l'approache OUVERTE de ICL

L'ensemble des produits de gestion de distribution ICL concerne la gestion de la distribution de tous les types d'objets de "gestion de système": logiciel, documents, données de configuration, etc., dans une communauté d'ordinateurs interconnectés. La transmission peut se faire soit par réseau, soit par support interchangeable.

Au coeur de cet ensemble de produits, on trouve une application conforme aux normes existantes on encours d'adoption et capable d'assumer la gestion de la distribution d'objets relatif à la "gestion de système", sur un réseau multilivreur composé de plate-formes Unix™ et non-Unix. Les produits sont basés sur une architecture qui favorise un changement d'éhelle répondant ainsi aux besoins des réseaux de toutes tailles.

Cet article présente les exigences du marché, l'approche adoptée en termes d'analyse des besoins et d'architecture, et la fonctionnalité du produit, et décrit la manière dont la solution a été élaborée et les produits sur lesquels elle a été mise en oeuvre.

George Brown
ICL Mid-Range Systems Division, Basingstoke, Royaume-Uni.
Gestion des systèmes: un défi pour les années 90 pourquoi maintenant?

Le faible coût des systèmes informatiques et l'accroissement permanent de leur puissance sont aujourd'hui tels que l'on risque, à terme, de ne plus être à même d'en faire bon usage. Les communications en large bande et les réseaux permettent d'augmenter leur puissance pratiquement à l'infini. Paradoxalement, il reste cependant difficile d'exploiter cette puissance à 100%, principalement en raison de notre incapacité à l'organiser et à la gérer. Toutefois, une technologie de gestion des systèmes commence à s'attaquer à ces problèmes et à offrir des outils de valeur pratique. Après une brève description des problèmes, cet article résume par le biais d'illustrations les composantes essentielles des produits ICL OSMC (Open Systems Management Centre — centre de gestion des systèmes ouverts).

J.W.S. Carmichael
ICL — Defence Technology Centre, Winnersh, Berks
Origines de PERICLES — une interface en ligne universelle

A l'instar des autres grandes sociétés, ICL dépend totalement des ordinateurs pour la marche de ses affaires. Au fil des années, la société a développé un certain nombre

de systèmes, principalement adaptés à ses propres besoins, mais integrant certain degrés de généralité. Parmi ceux-ci, l'un des plus puissants, doté de ce que l'on nommerait aujourd'hui une interface homme-machine particulièrement conviviale, était PERICLES, Développé pour la série ICL 1900, il a vu le jour en 1975. Cet article résume l'historique de ce système et ses principales caractéristiques.

Tony Gale
Architecture, Conformance and Verification Division, ICL, Basingstoke, Royaume-Uni
Evolution chez ICL d'une architecture de gestion des systèmes

L'évolution de la gestion des systèmes pour les systèmes informatiques distribués présente de nombreuses similitudes avec celle des systèmes d'exploitation des ordinateurs. Cet article présente la perspective ICL en la matière, en mettant l'accent sur la nécessité d'atteindre des objectifs commerciaux spécifiques, tout en contribuant à notre compréhension du sujet. Il décrit l'architecture générale et applique les principes fondamentaux de gestion à la conception, à la livraison et au contrôle opérationnel d'un système informatique complet.

Le système à gérer est composé du matériel, du logiciel et des personnes qui utilisent les services fournis. les réseaux, les ordinateurs, les systèmes d'exploitation et les applications sont tous concernés et la gestion des systèmes doit faire en sorte qu'ils contribuent tous à la réussite de l'entreprise dans son ensemble.

Dave Hacker
ICL Systems Management Product Center, Basingstoke, Royaume-Uni
Gestion des opérations

Au début de l'année 1991, ICL a lancé une série de logiciels appelée OSMC (Open Systems Management Centre — centre de gestion des systèmes ouverts). Cet article décrit la partie gestion des opérations du logiciel constituant OSMC et explique ce que les clients de ICL doivent faire pour atteindre un meilleur niveau de "facilité de gestion" en utilisant la version actuelle de OSMC.

Le programme OSMC reconnaissait que la "*facilité de gestion*" devait faire partie intégrante du processus de développement de toutes les applications futures. C'est la raison pour laquelle ICL travaille à l'établissement d'un certain nombre de normes pertinentes pour les systèmes ouverts. Dans l'entrefaite, certaines fonctions de gestion des opérations, qui n'exigent aucune modification des applications concernées, pourront être mises en service. Une telle facilité de mise en oeuvre améliore considérablement la souplesse et la rapidité de réalisation de la gestion des systèmes sur un réseau conforme aux normes des systèmes ouverts.

Lorsque les systèmes ne sont pas ouverts — c'est-à-dire lorsqu'ils se conforment à des protocoles propriétaires — et que l'infrastructure OSMC existe, il est toujours possible de garantir la facilité de gestion grâce à la gestion des opérations, bien que cela nécessite l'écriture d'un peu de code d'application personnalisé. Cet article montre comment le gestionnaire des opérations OSMC permet à un prestataire de services d'être averti des problèmes avant qu'ils n'influencent le niveau du service fourni au client. Il peut ainsi agir de manière préventive plutôt que réactive. Ce comportement est essentiel lorsque l'on souhaite améliorer tant le niveau réel que perçu du service offert par l'informatique.

Gareth I. Jenkins
Systems Management Product Centre, ICL Mid range Systems Division, Basingstoke, Royaume-Uni
Facilité de gestion d'un système distribué

A l'instar des autres formes de gestion, la gestion des systèmes nécessite une participation tant des éléments à gérer que du système de gestion. Cet article étudie la relation entre les applications de gestion et ce qu'elles gérent (ou les ressources gérées) du point de vue des ressources gérées, et propose un cadre dans lequel elles peuvent toutes prendre place. Outre l'impact sur le processus général de conception, l'article aborde également la relation vis-à-vis du travail externe sur la normalisation et des accords des personnes chargées de la mise en application quant à l'interfonctionnement des ressources gérées.

Michael H. Kay
ICL Fellow, Reading, Royaume-Uni
Peter J. Rivett
ICL, CASE Product Center, Basingstoke, Royaume-Uni
Présentation du système de base de données orienté objet Raleigh

Raleigh est un système de base de données orienté objet, comprenant un modèle de données fonctionnel, ainsi que son propre langage complet du point de vue calcul, OODL. Initialement, il est développé pour l'usage interne dans les programmes ICL de développement d'applications et de gestion de système.

Cet article donne un aperçu du système Raleigh, en mettant l'accent sur son modèle de données et son langage, mais en décrivant également l'architecture de mise en oeuvre.

Tony Maynard-Smith
Network Systems, ICL Secure Systems, Stevenage, Royaume-Uni
Domaine de la gestion des réseaux

Bien que la gestion des réseaux et la gestion des systèmes présentent de nombreuses similitudes et un grand nombre de points de recouvrement, une bonne compréhension de leurs différences est essentielle si l'on souhaite apprécier correctement les orientations prises dans les deux domaines. Cet article décrit l'historique du développement de systèmes de gestion de réseaux, la position actuelle en matière de normes et les différents types d'intégration appropriés aux diverses circonstances. Il présente également l'approche ICL en matière de fourniture de systèmes de gestion de réseaux.

Brian Moore
ICL Secure Systems, Bracknell, Royaume-Uni
Création d'un système bureautique sûr

Cet article décrit l'utilisation du système Secure UNIX^R de ICL, comme base de développement d'une application majeure — un système bureautique sûr. Il présente la politique de sécurité mise en oeuvre par Secure UNIX et donne un aperçu des extensions qui doivent y être apportées pour réaliser le système bureautique sûr.

Enfin, il décrit certains aspects du système développé, plus particulièrement la capacité des utilisateurs dans un environnement de bureau à faire face aux contraintes qui leur sont imposées par la politique de sécurité.

Ian Pickworth
ICL Retail Systems, Bracknell, Royaume-Uni
Expérience de gestion des flux de données en informatique distribuée dans le commerce de détail

Une grande expérience acquise avec quelques-unes des principales sociétés de distribution du monde a amené ICL à créer une gamme de produits spécialisés dans la gestion du flux des données au sein d'une organisation de distribution. Ces produits sont basés sur les produits ICL de gestion des systèmes, mais spécialement adaptés au marché de la distribution. Cet article commence par présenter de manière générale le traitement des données du commerce de détail. Ensuite, il décrit l'évolution des produits au cours de la dernière décennie, avant de terminer par les orientations futures susceptibles d'être prises par les produits.

M. Small, J.D. Mitcalf, K.J. Johnstone, J.W. Doores
ICL Strategic Systems, Cardinal house, Manchester, Royaume-Uni
Gestionnaire de contrôle des opérations OSMC

Les sociétés qui utilisent un système informatique important possèdent bien souvent plusieurs main frames et systèmes de bureau. Ceux-ci peuvent être localisés en différents emplacements, chacun d'eux supporté par une équipe informatique. Partie intégrante du système OSMC, le gestionnaire de contrôle des opérations (OCM — Operations Control Manager) permet aux responsables de contrôler ces systèmes à partir d'un point central.

Le système OCM utilise un poste de travail graphique pour afficher une vue d'ensemble du réseau. Il indique l'état d'objets définis, permettant ainsi à l'opérateur central d'effectuer des recherches et de tapper des commandes aux systèmes gérés. Lorsque cela s'avère nécessaire, certaines parties du réseau peuvent être agrandies et affichées plus en détail.

Le système OCM a été développé conformément à la méthodologie ICL qui intègre marketing et conception. Ainsi, des facteurs humains sont inclus dans les spécifications et la conception du produit, ce qui se traduit par des produits nettement plus conviviaux pour leurs utilisateurs.

Jim White
ICL Systems Management Product Centre, Basingstoke, Royaume-Uni
Génération de configurations — un travail d'équipe

La configuration des nombreux systèmes différents intégrés dans les systèmes interconnectés et distribués, ainsi que de toutes les applications qu'ils utilisent, peut s'avérer une tâche présentant des risques d'erreur importants et nécessitant un niveau de connaissance considérable. Les problèmes qui en découlent peuvent nuire au développement de l'usage du système informatique d'une entreprise et/ou réduire sa capacité d'évolution.

Les aspects "génération" de l'architecture ICL de gestion des systèmes (Systems Management Architecture) s'attachent tout particulièrement à ces problèmes. Cet article décrit la façon dont ICL a développé des prototypes de son approche en collaboration avec un de ses clients importants. Il présente cette approche et justifie les choix effectués.

Kam-Fai Wong
European Computer-Industry Research Centre (ECRC) GmbH, Arabellastrasse 17, 8000 Munich 81, Allemagne
Architectures des machines à base de connaissances

Les systèmes de bases de données classiques fondés sur un modèle relationnel sont largement utilisés dans de nombreux domaines d'application. Néanmoins, les bases de données relationnelles ne sont pas adaptées aux applications évoluées (par exemple, la conception assistée par ordinateur, CAO). Cela s'explique essentiellement par la puissance de modélisation limitée et l'absence de capacité d'inférence du modèle relationnel. Pour résoudre ces problèmes, des modèles de bases de données évolués — généralement appelés bases de connaissances — sont développés. Etant donné la complexité des modèles à base de connaissances, les systèmes qui les utilisent ne peuvent pas fonctionner de manière efficace sur des ordinateurs ordinaires, cela s'accentuant à mesure de l'augmentation de la taille des bases de données traitées. Des machines spécialement conçues pour traiter ces bases de connaissances de façon efficace ont ainsi été proposées. Cet article décrit l'architecture de cinq machines à base de connaissances: KBM (Knowledge Base Machine, Japon), DDC (Delta Driven Computer, France), CLARE (CLAuse Retrieval Engine, Royaume-Uni), PRISMA (Pays-Bas) et EDS (European Declarative System, Europe).

Zusammenfassungen

Phil Barthram und Tim Howling
ICL Mid Range Systems Division, Basingstoke, Großbritannien
Distributions — Verwaltung — ICL's Offener Ansatz

ICL's Distribution Management Software verwaltet die Verteilung der verschiedenen Systemverwaltungs-Objekte innerhalb einer Gruppe von vernetzten Computern, d.h. Software, Dokumente, Konfigurationsdaten usw. Die Übertragung erfolgt entweder über das Netz oder über austauschbare Datenträger.

Der wichtigste Bestandteil dieses Produkts ist eine Verwaltungsanwendung, die den existierenden und den zukünftigen Normen gerecht wird und welche die Verteilung von Systemverwaltungsobjekten über ein Netzwerk von Unix- und Nicht-Unix-Plattformen verschiedenster Hersteller verwalten kann. Das Produktpaket baut auf einer Architektur mit flexibler Skalierung auf, so daß sie die den Anforderungen sowohl sehr kleiner als auch sehr großer Netze angepaßt werden kann.

Dieser Artikel beschreibt die Marktanforderungen, den gewählten Ansatz zur Analyse dieser Anforderungen und die Architektur und Funktionalität des Produkts, sowie Art und Weise, wie eine entsprechende Lösung ermittelt wurde, und die daraufaufbauenden Produkte.

George Brown
ICL Mid-Range Systems Division, Basingstoke, Großbritannien
Systemverwaltung: Eine Herausfarderung für die Neunziger Jahre-

Computer werden heute immer preiswerter und ihre Leistungsfähigkeit nimmt so schnell zu, daß wir Gefahr laufen, diese Fähigkeit nicht mehr genügend ausnützen zu können. Breitband-Kommunikation und leistungsstarke Netze steigern diese Leistung potentiell fast ins Grenzenlose. Dennoch scheint es merkwürdigerweise immer noch schwierig zu sein, diese Leistungsfähigkeit voll auszuschöpfen, was vor allem daran liegt, daß wir nicht in der Lage sind, diese richtig zu organisieren und zu verwalten. Die Technologie zur Verwaltung von Systemen ermöglicht es jedoch inzwischen sich mit diesen Problemen zu befassen und praxisgerechte Werkzeuge zu liefern. Dieser Bericht gibt eine kurze Beschreibung der Probleme, gefolgt von einer beispielhaften Zusammenfassung der wichtigsten Komponente des ICL Open Systems Management Centre.

J.W.S. Carmichael
ICL Defence Technology Centre, Winnersh, Berks, Großbritannien
Die Entstehungsgeschichte von Pericles — Eine gemeinsame Online-Schnittstelle

Wie bei allen großen Firmen ist auch bei ICL der laufende Geschäftsbetrieb völlig von Computersystemen abhängig. Die Firma hat im Laufe der Jahre eine Reihe von Systemanwendungen entwickelt, die zwar in erster Linie auf die eigenen Anforderungen zugeschnitten waren, die jedoch in verschiedenen stufen auch allgemein eingesetzt werden können. Eines der leistungsfähigsten Anwendungen, mit einer — wie man es heute nennen würde — besonders benutzerfreundlichen Mensch-Maschine-Schnittstelle war PERICLES, die für die ICL 1900-Serie entwickelt und das erste Mal 1975 eingesetzt wurde. Der Artikel gibt eine Zusammenfassung der Entwicklungsgeschichte dieses Systems und seiner wichtigsten Eigenschaften.

Tony Gale
Architecture, Conformance and Verification Division, ICL, Basingstoke, Großbritannien
Die ICL-Interne Evolution einer Architektur für die Systemverwaltung
Die Evolution der Systemverwaltung für verteilte Informationssysteme weist starke Parallelen zu der Evolution von Betriebssystemen für Computer auf. Dieser Artikel beschreibt ICL's Perspektive zu diesem Thema, wobei auf die Notwendigkeit ganz spezifische Geschäftsziele zu lösen eingegangen wird, als auch auf unser allgemeines Verständnis zu diesem Thema. Der Bericht gibt eine Beschreibung der allgemeinen Architektur und erläutert, welche Verwaltungsprinzipien auf die Planung, Lieferung und Kontrolle eines kompletten Informationssystems angewandt werden.

Ein zu verwaltendes System besteht insgesamt aus Hardware, Software und den Personen, die die gelieferten Dienstleistungen benutzen. Netze, Computer, Betriebssysteme und Anwendungen spielen dabei ebenfalls ein wichtige Rolle und die Systemverwaltung muß gewährleisten, daß all diese Komponenten zu dem Erfolg des gesamten Unternehmens beitragen.

Dave Hacker
ICL Systems Management Product Centre, Basingstoke, Großbritannien
Operations Management

Anfang 1991 hat ICL ein Paket von Software-Produkten unter dem Namen "Open Systems Management Centre" (OSMC) auf den Markt gebracht. Dieser Artikel beschreibt die einzelnen Komponente dieser OSMC-Software und erläutert, was ICL-Kunden tun müssen, um mit Hilfe der aktuellen OSMC-Freigabe eine besser kontrollierbare systemverwaltung erreichen zu können.

Das OSMC-Programm basiert auf der Erkenntnis, daß *Kontrollierbarkeit* ein Bestandteil des Entwicklungsprozesses aller zukünftigen Anwendungen sein muß. Deshalb arbeitet ICL an der Etablierung einer Reihe von relevanten Normen für offene Systeme. Inzwischen konnten verschiedene Funktionen für die Betriebsverwaltung freigegeben werden, die keine Änderungen der beteiligten Anwendungen erfordern. Ein derartig unproblematischer Einsatz verbessert die Flexibilität und die

Schnelligkeit der Implementierung der Systemverwaltung eines Netzes, das mit den Normen offener Systeme übereinstimmt, in erheblichem Maße.

Selbst wenn die End-Systeme nicht offen sind, d.h. wenn sie proprietären Protokollen unterliegen, und wenn die OSMC-Infrastruktur existiert, kann eine Kontrollierbarkeit der Betriebsverwaltung erreicht werden. In diesem Fall müssen jedoch entsprechende kurze Anwendungs programme geschrieben werden.

Es wird beschrieben wie der OSMC Operations Manager einem Serviceunternehmen die Möglichkeit bietet, Probleme zu erkennen, bevor sie sich auf die Dienstleistungen auswirken, die dem Endanwender zu liefern sind. Dadurch kann man bereits im Voraus handeln, anstatt erst hinterher zu reagieren. Diese Fähigkeit ist ausschlaggebend für die Verbesserung der tatsächlichen und erwarteten Leistung, die ein Computer Dienstleistungsunternelunen anbieten kann.

Gareth I. Jenkins
Systems Management Product Centre, ICL Mid Range Systems Division, Basingstoke, Großbritannien
Kontrollierbarkeit eines Verteilten Systems

Wie bei jeder anderen Art der Verwaltung umfaßt auch die Systemverwaltung neben den Ressourcen, die verwaltet werden, auch das Verwaltungssystem selbst. Dieser Artikel befaßt sich mit der Beziehung zwischen der Verwaltungsanwendung und den Ressourcen, die verwaltet werden (oder den verwalteten Betriebsmitteln), und zwar vom Standpunkt der verwalteten Betriebsmittel aus. Ferner empfiehlt er eine Rahmenstruktur, in welcher die verwalteten Betriebsmittel eingefügt werden können und befaßt sich mit der Frage, welche Auswirkungen dies auf den gesamten Design-Prozeß hat. Die Beziehung zu externer Arbei zur Standardisierung und den Vereinbarungen zwischen verschiedenen Ambietern hinsichtlich der Interoperabilitat der verwalteten Betriebsmittel wird ebenfalls erörtert.

Michael H. Kay
ICL Fellow, Reading, Großbritannien
Peter J. Rivett
ICL, CASE Product Centre, Basingstoke, Großbritannien
Ein Überblick über das Objektorientierten Datenbanksystem Raleigh

Raleigh ist ein objektorientiertes Datenbanksystem, das ein funktionales Datenmodell und seine eigene Verarbeitungsprache OODL umfaßt. Es wird zunächst für den internen Gebrauch innerhalb von ICL's Produktprogramme für das System Management und der Anwendungsentwicklung eingesetct.

Dieser Artikel bietet einen Überblick über Raleigh, mit Schwerpunkt auf dessen Datenmodell und der Datenbanksprache, gibt aber auch eine kurze Beschreibung seiner Implementierungsarchitektur.

Tony Maynard-Smith
Network Systems, ICL Secure Systems, Stevenage, Großbritannien
Die Netzwerkverwaltungs-Domäne

Die Netzwerkverwaltung und die Systemverwaltung ähneln und überschneiden sich in vielerlei Hinsicht; um jedoch wirklich zu verstehen, in welche Richtungen sich diese zwei Bereiche entwickeln, muß man die Unterschiede zwischen den beiden Systemen verstehen. Der Artikel beschreibt den Hintergrund für die Entwicklung der Netzwerkverwaltung, die aktuelle Position hinsichtlich der Normen, sowie die verschiedenen Formen der Integrationsmöglichkeiten. Außerdem wird beschrieben, welches konzept ICL bezüglich der Lieferung von Netzwerkverwaltungs-Produkten gewählt hat.

Brian Moore
ICL Secure Systems, Bracknell, Großbritannien
Die Schaffung eines Sicheren Bürosystems

Dieser Artikel beschreibt die Verwendung von ICLs Secure UNIX^R als Basis für die Entwicklung einer wichtigen Anwendung — die eines sicheren Bürosystems (Secure Office System). Er beschreibt die von Secure UNIX auferlegte Sicherheitspolitik und umreißt die notwendigen Erweiterungen für das sichere Bürosystem. Zum Schluß werden noch einige Aspekte des daraus resultierenden Systems beschrieben, vor allem die Möglichkeit der Benutzer, in einer Büroumgebung mit den durch die Sicherheitspolitik auferlegten Einschränkungen zurecht zu kommen.

Ian Pickworth
ICL Retail Systems, Bracknell, Großbritannien
Erfahrung mit der Verwaltung von Datenflüssen bei Verteilter Verarbeitung in Einzelhandelsbetrieben

Umfassende Erfahrungen mit einigen der weltweit größten Einzelhändler haben zu der Entwicklung einer ICL-Produktlinie geführt, die Datenflüsse innerhalb eines Einzelhandelsbetriebes verwaltet. Die Produkte basieren auf ICL's System Management-Produkte, sind aber speziell auf den Einzelhandelsmarkt ausgerichtet. Der Artikel gibt zunächst einen kurzen Hintergrund zur Datenverarbeitung im Einzelhandel, gefolgt von einer Beschreibung der Entwicklung dieser Produkte im letzten Jahrzehnt. Abschließend wird ein Einblick in die mögliche Weiterentwicklung der Produkte gegeben.

M. Small, J.D. Mitcalf, K.J. Johnstone, J.W. Doores
ICL Strategic Systems, Cardinal House, Manchester, Großbritannien
OSMC Operations Control Manager

Eine Firma, die mit einem großen IT-System arbeitet, verfügt häufig über mehrere Großrechner und Bürosysteme. Diese befinden sich möglicherweise an verschiedenen Standorten und werden wahrscheinlich jeweils von einem eigenen IT-Team unterstützt. Der Operations Control Manager (OCM), Teil des OSMC-Systems bietet die Möglichkeit, diese verschiedenen Systeme von einer Zentralstelle aus zu steuern.

Der OCM benutzt einen graphischen Arbeitsplatzes um eine Übersicht über das gesamte Netzwerk auzuzeigen. Er zeigt den Status einzelner Objekte an, so daß der zentrale Bediener in der Lage ist, Detaills zu überprüfen und Befehlt an die verwalteten Systeme zu geben. Wenn nötig, kann die Bildschirmanzeige erweitert und spezifische Teile des Netzes genauer eingesehen werden.

Der OCM wurde mit Hilfe von ICLs 'Marketing to design'-Methode entwickelt. Diese Methode schließt die menschlichen Faktoren in die Produktspezifikation und deren Entwickelung mit ein, was zu Produkten führt, die von den Benutzen leichter akzeptiert werden.

Jim White
ICL Systems Management Product Centre, Basingstoke, Großbritannien
Generierung von Konfigurationen — ein Gemeinschaftsprojekt

Das Konfigurieren der vielen verschiedenen Systeme in einer vernetzten und verteilten Systemumgebung kann eine Fehlergefährdet und der darauf laufenden Anwendungen Aufgabe sein, die bedeutende Fachkenntnisse voraussetzt. Die damit verbundenen Probleme können reduzieren den Einsatz von zusätzlichen Anwendungen verhindern und/oder die Fähigkeit notwendige anderungen schnell und problemlos durchführen zu können.

Die Generierungs möglichkeiten der Systems Management Architektur von ICL sind speziell auf diese Probleme ausgerichtet. Dieser Artikel beschreibt, wie die Firma ICL Prototypen in Zusammenarbeit mit einem ihrer größten Kunden entwickelt hat. Er gibt einen Überblick über diese Methode, sowie einige Gründe für die im Einzelnen getroffenen Entscheidungen.

Kam-Fai Wong
European Computer-Industry Research Centre (ECRC) GmbH Arabellastraße 17, 8000 München 81, Deutschland
Architekturen von Wissensbasierten Maschinen

Normale relationale Datenbanksysteme werden auf breiter Ebene in vielen Anwendungsbereichen eingesetzt. Für spezielle Anwendungen (wie etwa rechnerunterstütztes Konstruieren, CAD) sind relationale Datenbanken jedoch ungeeignet. Dies liegt vor allem an der begrenzten Modellbildungs- und der mangelnden Inferenzfähigkeit innerhalb des relationalen Modells. Um dieses Problem zu überwinden, werden erweiterte Datenbankmodelle — gewöhnlich als Wissensbasen bezeichnet — eingeführt. Da die Wissensbasis-Modelle äußerst komplex sind, können sie auf konventionellen Computern nichteffizient laufen. Mit dem wachsenden Umfang von Daten, die solche Systeme zu bewältigen haben, wird dieses Problem zunehmend größer. Es wurden daher spezial-Hardware systeme für die effiziente Handhabung von Wissensbasen vorgeschlagen. Dieser Artikel beschreibt die Architektur von fünf Wissensbasis-Maschinen. Dazu gehören die Knowledge Base Machine (KBM, Japan), der Delta Driven Computer (DDC, Frankreich), die CLAuse Retrieval Engine (CLARE, Großbritannien), die PRISMA-Maschine (Holland) und das European Declarative System EDS (Europa).

Editorial

Aspects of Management

This issue carries ten articles by ICL authors on Systems Management, a subject rapidly becoming very important to users running more than a single isolated computer. It also includes a review of an important book, "The Corporation of the Nineties", derived from a major sponsored research programme at the Sloan School of Management at MIT which was actively supported by ICL. Together these contributions have promoted reflection on the idea of management and on the agencies human or inanimate that perform it.

Between business management and system management there are in fact common strands. Both must begin by agreeing the purpose or purposes to be achieved, continue by deciding on practical policies for serving those purposes, then assigning responsibilities to various agents to execute tasks that together should attain the desired purpose, giving each agent powers and means matched to its specific responsibility, defining and delimiting that responsibility so as to avoid confusion or overlap but without leaving any essential task unassigned and, finally, setting up mechanisms for reporting back to a central body charged with seeing that the agreed purposes are in fact being achieved.

In practice ***management*** is a composite process performed by a variety of agencies, some human and some automatic. (The way the word "manager" is used can leave it rather uncertain whether a person is or is not involved). In the business case the highest level functions are invariably left to people, a board of directors, aided and guided of course by computers. With system management too ***people*** must be involved because people designed the hardware and software in the first place and because someone had to set up the system to behave in a way appropriate to that user selected from the wide range of ways envisaged by the designers. Designers will have aimed to reduce the need for frequent or low-level human intervention as far as they can, but there must always be opportunities for people to seize control if automatic behaviour is seen to be dangerous.

It has often been said that if the word system is not to become meaningless, it is essential to draw a line around those entities thought of as within it so distinguishing them from the external environment. The criterion for deciding where to draw the line is whether a given component entity is or is not subject to closely programmed direction or regulation. In a completely automatic system this is not too difficult. When human beings are included distinguishing between system and environment can be awkward. Can they be assumed to behave in a prescribed or at least a predictable way so that they can safely be regarded as components of the system or is it wiser to assume that they may (sometimes) act in a way that must appear to the system to be meaningless or random? In that case they should be kept outside it.

There also seems to be two significantly different ***classes*** of system management. The first concerns what can be planned in advance by a human manager when there is no immediate operational pressure. It could be deciding the topology of the interconnections between the computers and terminals comprising a network or fixing the relative priorities for the different routine interactions between central and local nodes. To use antique civil service jargon, this might be called the ***administrative*** mode of management.

The other class is when the network is, or appears to be, misbehaving because of some unknown failure of a component or some unforeseen human act or external occurence like a lightning strike. Clearly this kind of management is more difficult. If the system has collapsed totally then system management cannot be effective; but much more often its functioning is merely impaired. Then, for the sorts of reasons given in Pickworth's paper, it is extremely urgent to bring it back quickly to full efficiency. To recognise rapidly what is wrong, the forms of presentation described in the papers by Hacker and by Small et al may allow a human operator to recognise what is at fault and where. Reconfiguring the network itself or reallocating tasks to other nodes may allow operations to proceed, either unimpaired or at reduced efficiency, while repairs are made. If the failure is non-trivial human intervention is likely to be essential and details of that intervention must be carefully logged. This type of management could, using the same civil service terminology, be called ***executive*** mode.

This brief analysis indicates that the essence of computer system management, as it is of IT in business management, is putting ***people*** in a better position to control events in the system. It must do this by giving them a synoptic view of on-going processes and by enabling them to predict, perceive and monitor both normal and abnormal behaviour of the system as a whole or of any selected component part of it – never forgetting possible misbehaviour by other people.

Therefore, training people to understand and so to use the concepts of system management embodied in its hardware and especially in software designs will be vital to ICL's success in launching OSMC in the marketplace.

SYSTEM MANAGEMENT

Foreword

System Management: The Open System Management Centre

For those of us who are closely associated with the development of Information Technology, it is all too easy to overlook the revolution which has taken place over the last ten years. Not surprisingly, this has given rise to new problems and challenges. This edition of the Technical Journal focuses on our improving responses to some of these newer factors in IT.

If we go back just ten years, the typical installation was mainframe centred. There were terminals, sometimes in profusion, but there was no doubt about where the data, the applications and the control resided — on the mainframe. To make things even simpler, the average organisation had only one or two mainframes. Although network control, performance optimisation and system management seemed hard enough, they were tractable problems which a few simple tools could tame, if not quite domesticate.

Today's picture is painted in very different colours. The majority of the organisation's power is spread across the employees' desks; the mainframe is a complex of processors, sometimes split across sites; there is a new tier of midrange systems performing tasks which would hardly have been recognised ten years ago. Adding to this is the presence of products from many vendors and standard software which has often been selected, purchased and installed without the intervention of the traditional data processing experts.

It is a sad but inescapable fact that we human beings have not advanced in our mental powers to the same degree over this period. This leaves us very much more in need of powerful tools to control a more complex and intricate set-up.

ICL early recognised the need in this area and has, since 1982, resourced a Systems Management programme which covered research, prototypes and early, focussed product developments. Around three years ago, ICL took up the challenge by planning an integrated set of products known as the *Open Systems Management Centre*, (OSMC). This provides direct control of the basic hardware and software in the network. It embraces maintenance of the myriad applications spread around the network; the containment and correction of errors; and help and guidance to the humans at the terminals. Overlaid on all this are facilities to optimise performance. OSMC makes this all possible using a small number of experts located at one, or a few, locations so that central control is maintained and costs contained.

1991 is a key year in this programme for ICL, because this year the first deliverables for OSMC are made available. ICL is showing that it is at the forefront of the development of the advanced, Open facilities which users must have in order to take the fullest business advantage of their widespread IT investments. But there are other important aspects which should be noted as well. OSMC products are being developed for a wide range of platforms, some of which are from other vendors. This has encouraged ICL to collaborate with a new range of software houses and systems integrators, an experience which is going to be of advantage to all concerned for many years to come.

OSMC is an important step along the way of keeping IT firmly under control while delivering the business advantages which justify the investment. ICL is proud to have been a partner with so many customers and collaborators in achieving this and looks forward to maintaining this innovative position. The articles which follow provide an excellent description of what has already been achieved and reveal the thinking which will take us all further on this important journey.

A.J. Boswell
Director
Architecture, Conformance and Verification Division
ICL Product Operations.

Systems Management: a challenge for the Nineties – why now?

George Brown
ICL Mid-Range Systems Division, Basingstoke, UK

Abstract

Computing power is now so cheap and growing so fast that it is threatening to outstrip our ability to make good use of it. Wide bandwidth communications and networks potentially increase this power almost without limit, but it still remains curiously difficult to exploit this power to the full, chiefly because of our inability to organise and manage it. However, a technology for managing systems is beginning to address these problems and to provide tools of practical value. After briefly describing the problems, the paper summarises the main component parts of the ICL Open Systems Management Centre by way of illustration.

1 Introduction

Management is most effective when invisible. When we visit a hotel or go to an event where everything runs smoothly and things appear when they should, we know that it is well managed. It is not the appearance of the Manager but the effect of management that we notice. Conversely, we look to "Management" to fix things if there is a problem, so that we are more likely to see the Manager when things are going wrong than when they are going right.

The management of distributed computer systems is no different. Since the subject of systems management is now so much under discussion, we need to ask what is going wrong. There are a number of simplistic answers to this question. "I don't know what is going on out there in the network". "I can't seem to recruit the number of UNIX experts I need to keep this system running". "The Help Desk is swamped every time we try to change anything". But these are only symptoms. We need to dig deeper to find the underlying causes and their origins.

In this paper, the term "Systems Management" or sometimes just "Management" is used to cover all the processes and tools that must be applied to the management of a set of IT resources in order to deliver an agreed service in a cost-effective manner. A number of other terms are now being used in the description of Systems Management technology which are defined in Gale (1991), which describes the architecture.

That all forms of technology are advancing at an ever increasing pace is a familiar idea and in information technology and computing this is very evident. The reason may have more to do with technology than with market demand. Developments in different technologies have combined to multiply the overall speed of development. But now our ability to exploit the unexpected bonus is becoming limited by the lack of tools to control and co-ordinate the linking together of large numbers of traditionally autonomous computers.

The main task of systems management is therefore to unlock the potential of Information Technology by making it more easily available to users and to ease the problems caused by size and complexity. This requires facilities for both planning and for crisis management or problem solving. However, systems management solutions are now starting to be developed and will be a major point of focus in Information Technology during the present decade.

We can draw a useful parallel with the fifties and sixties when there was a need to control the growing power of the computer itself, which resulted in the development of operating systems. There are naturally many differences of detail between the design of operating systems and that required for systems management. But the basic problems are very similar - sharing resources amongst many users, security and protecting users from the complexity of the technology.

2 Relevant trends in base technology

Many advances in the physical sciences have been directed towards the development of computer technology. The main objective of these has been that computers should become both faster and cheaper, and store and process ever more data. The rate of progress has been exceedingly fast and operating systems technology has had to respond by continuous development of new techniques and methodologies. But the trend today is towards the distribution of computing power rather than the continuous development of bigger and faster computers, except perhaps for some very specialised applications.

There is one area where the demand for more power is still growing and that is at the desk top. Powerful workstations and high resolution screens are revolutionising the interface between man and computer. The spread of personal computers has opened our eyes to what may be possible. This type of computing creates a heavy demand for power to provide the graphics and satisfy the need for very rapid response. It has been claimed that the faster the computer responds, the faster the man will work. It is therefore inevitable that intelligent workstations, with highly sophisticated, local man/machine interface software, will become the normal method of input and output to corporate and other remote systems. Naturally, users who become accustomed to this type of rapid interaction with their workstations are not likely to tolerate a much lower standard when linked to remote systems.

The availability of greater computing power is affecting the ways in which we now build applications. While using the power of the computer to sort out the logic, relational databases, fourth generation programming tools and the employment of a more declarative style to express requirements, all ease the task and increase the speed of application development, the disadvantage of this trend is a rapid increase in overall complexity. This complexity certainly includes installation and configuration procedures. In many systems these need a high degree of skill and expertise, which cannot realistically be supplied locally. One answer to this problem is to develop systems management tools which combine elements of both centralisation and devolution of control. The objective is, firstly, to do as much as possible automatically at the remote end by making use of the intelligence which is there, while the second requirement is to provide control facilities at a central location to allow a human expert to operate the remote systems on an exception basis.

Developments in communications technology are making more bandwidth and greater speed available at ever lower cost. It is therefore becoming cost-effective to distribute computing power to places of work. Such local or departmental systems are frequently provided with links to remote corporate systems or corporate data. This type of setup is now becoming common in industry, commerce and government.

But however user-friendly and easy to use departmental systems become, there are always some tasks which need a "guru", particularly when something goes wrong. There are just not enough skilled staff to go round, so it is essential to provide tools to do the necessary work from a distance. The trend towards soft engineering instead of hard, wherever possible, has meant that faults can be diagnosed and often corrected from a distance, thus opening one area of application for systems management.

Universal access to departmental systems which are connected to remote corporate systems is being implemented everywhere through highly sophisticated, intelligent workstations. But the costs of provision and maintenance of service are beginning to become a limiting factor, which is obliging users in many places to give higher priority to systems management.

3 Trends in Working practices

Early uses for computers in business administration were to support, improve or speed up existing activities in those organisations. Gradually this gave way to finding new ways of doing things; this trend continues, as organisations depend increasingly on Information Technology. There are many examples now of businesses which could not function without their computing facilities. But, more significantly, there are also examples of organisations which owe their position of leadership to the way that they have exploited IT.

The effect of the greatly increased importance of computing to business is that loss of availability can now lead directly to loss of business. It used to be possible to fall back on the old manual system. But for many modern systems, there is no manual back-up. Retailers now employ bar codes rather than individually marking goods with the price. This offers great flexibility, in that prices can be changed during the day without touching the actual goods. But it does mean that if the system goes down, the store may have to stop trading altogether rather than risk angering customers by causing huge delays. The significance of Systems Management to the retail industry is more fully described in (Pickworth, 1991).

Availability is a central feature of systems management. Operations Management is above all aimed at optimising availability. Some organisations now use computers to interact directly with their customers. The most obvious examples of this are the financial institutions with the familiar automatic teller. But the current reductions in staffing levels taking place in the banks are going a stage further. New technology is making large numbers of jobs unnecessary, but this means that any thought of falling back on a manual system is out of the question. Availability is both fundamental and expected.

4 Technology – an Agent of Change

IT is now enabling insurance companies, for instance, to offer with increasing frequency new types of policy, in order to enter new market sectors and to keep up with the competition. There is consequently a greater variety of different types of life assurance policy available, compared with ten or fifteen years ago. There are many similar examples, where IT has provided the ability to respond quickly, so that it has been used to provide a competitive advantage. The result has been that competitors have had to follow suit in order to survive and the pace of business generally has quickened.

The implications of this for computing technology are far-reaching. Computer systems have always needed to respond quickly to changing business requirements, but with the spread of distributed systems and the growing numbers of end users, change is becoming continuous. Staff movements, software changes, equipment upgrades and the introduction of new systems must all be planned, approved and controlled. Change management is therefore growing rapidly in importance and is a significant application for Systems Management. It starts with planning but also includes the basic mechanisms for initiating, carrying out, tracking and monitoring changes.

5 Open Systems

"Open Systems" means different things to different people. But with the rapid growth in the numbers of distributed systems, which will certainly take place during the nineties, the pressure to be able to build systems and networks supplied by multiple vendors will continue to grow. Most computer users today use equipment from a multiplicity of vendors. This can be for a variety of reasons and circumstances, but is often the result of a deliberate policy of using more than one source of supply for expensive and critical assets. Sometimes multi-sourcing means acquiring mainframe computers, office equipment or departmental systems from different suppliers. But the distinctions between these types of equipment are becoming difficult to sustain, where they are all needed to be networked together to form part of a corporate-wide system.

The requirement is therefore to be able to assemble distributed systems with components conforming to Open standards with complete freedom to choose the most appropriate suppliers. This is where Systems Management and Open Systems Interconnection are both involved. The ability to communicate freely between components is a universal requirement. But without a management system providing overall control and administration, there will be severe limitations.

Systems no longer have the convenient boundaries which they used to have. Corporate systems no longer end at the factory gate. There are already many communications links with the outside world, for instance with suppliers and customers, with public directories and information systems, with workers based at home and with other organisations by electronic mail.

Corporate computing resources are likely to consist of a series of overlapping and interconnected distributed systems. Management of these will be by means of a series of well defined domains of control (Gale, 1991) and (Maynard-Smith, 1991), where each domain will provide a defined set of services using a combination of resources. Some will be under direct control but many will be supplied as services purchased from other domains. This is similar to the DP Manager providing an application service running on his own computer centre but delivering it over a telecommunications service which he has purchased from another supplier.

The concept of *domains of control* is fundamental in bringing order to the system. Along with it is the concept of *service level agreements*. The important boundaries therefore are those between domains of control. Their existence will result in the provision of defined services possibly at contracted rates. Physical boundaries may become meaningless and invisible. Users will only be interested in the service that they get and will not care where it comes from. There is a clear analogy here with the telephone system. If you make a call from London to New York, you do not care how it is routed, you only care about a swift connection, a clear line and the charge. It makes no difference whether it goes by satellite or under the sea.

Corporate systems will become less easy to make secure by physical means. This is an aspect of systems management on which we may not see rapid progress in the short term, for two reasons. The first is that security must logically follow systems management by being implemented on top of the fundamental processes of operations management, generation, distribution and change management. It must follow when the infrastructure is all firmly in place. The second reason is that market pressure for security is not yet strong.

6 Open Systems Management Centre (OSMC)

In 1991 in response to these needs ICL launched a suite of products called the Open Systems Management Centre (OSMC). This suite of applications gives users the capability to manage their distributed IT equipment. Considerable flexibility is available to centralise management where it is needed or to devolve it as appropriate. By this means, management tools can be made available wherever the skills to use them may be located.

6.1 OSMC Problem Manager

OSMC Problem Manager is provided to help identify and expedite problems as they occur. Problems can be of many kinds from major equipment or software failure to a call for help from a user in difficulties. The objective is to optimise system availability by detecting problems early, diagnosing the cause and applying corrective action as soon as possible.

OSMC Problem Manager provides the ability to record and resolve a user call for help or service request. *Call logging* and *progression facilities* allow calls to be:

- updated with details of their progress
- referred to specialist support
- linked to similar problems
- closed and reopened.

Alerts generated anywhere in the network can be automatically logged and progressed as normal calls. A knowledge database of information is maintained holding:

- known fault procedures
- location information
- equipment at each location
- problem urgency levels
- problem resolution authorities.

Calls can be passed to identified problem resolution authorities and a priority system monitors the progress of resolution of more serious problems. There is also a procedures database with standard and recommended actions against specific problems. These facilities combine to increase the level of knowledge which may be used to resolve a call and speed up the resolution of problems.

6.2 OSMC Change Manager

Changes made to any of the components of a distributed IT system, whether to correct a fault or to provide a new service, may affect other components. Careful management and control of all such changes is therefore vitally important.

OSMC Change Manager provides the capability from one central location to log, plan and progress all planned changes to equipment, software, user locations or communications links within a distributed IT system. Facilities consist of a database of all change requests and a process for managing the agreed procedure for change authorisation. These are based on the process described in the Central Computer and Telecommunication Agency (CCTA) IT Infrastructure Library on Change Management. Flexibility also exists to tailor the system to support the customer's existing procedures and organisational structure.

For each change request the following information is held:

- identity and location of System to be changed
- change request identifier
- location which raised the request
- item to be changed, e.g. library, file or hardware item
- a statement of the impact of the change
- the time the request was raised
- the originator's name.

6.3 OSMC Operations Manager

OSMC Operations Manager provides the facilities required for remote management of networked systems. The facilities include the presentation of status information received from individual managed systems and the ability to exercise operational control over those systems.

A key feature of OSMC Operations Manager is the ability to display information from and to manage a number of different types of system from a single point. Control is exercised through a high-resolution graphics workstation which can display status information of the managed systems both logically and geographically. The detail of the display can be expanded to view the network of managed systems in greater detail and the display may be accessed at any level within this hierarchy. It also includes a knowledge base of the objects being managed.

The graphical display is based on the "Open Look" user interface standards and provides:

- standard icons to represent managed objects
- colour definition of managed systems
- display layout editable by the user
- display expansion facilities
- automatic propagation of status change
- menus of actions available for each managed object
- terminal emulation access to DRS/NX systems
- automatic operation of VME systems
- real time status, prompt and alert monitoring
- real time performance and threshold monitoring
- archiving and housekeeping remote systems.

A number of OSMC Operations Manager centres can be distributed across an organisation and networked together to create a regional management capability. For more detail see (Hacker, 1991).

6.4 OSMC Capacity Manager

OSMC Capacity Manager assists in planning that adequate capacity is available within the distributed IT system and that existing capacity is used effectively. The ultimate aim is to measure capacity in units of work which have a direct relevance to the users of the service. In the meantime usage and performance statistics are gathered from managed systems across the network into a single database for analysis. This analysis, together with information from OSMC Change Manager, provides an effective tool for planning future requirements.

The functionality provided includes:

- local extraction and collation of usage statistics
- automatic collection into a central database
- central facilities for analysis, reporting and capacity planning.

Collection and transfer of statistics can be scheduled flexibly and analysis tools can be freely selected.

6.5 OSMC Distribution Manager

The distribution of software and data around a network is growing in both importance and difficulty. In large distributed networks it is sometimes essential that all users employ the same versions of software at the same time. OSMC Distribution Manager provides facilities for both the collection and retrieval of files from end-systems. It can also employ intermediate servers to act as gateways for onward distribution. A database on the central management system holds information on distribution schedules and the location of files in the network.

OSMC Distribution Manager provides the operator with menu screens to assist in:

- selection of software and data for distribution
- defining how, when and where it will be distributed
- controlling the actual distribution function
- remote installation and activation of distributed software
- displaying information from the database.

The Distribution database holds information on:

- the end systems within the network
- distribution routes
- delivery schedules
- machine groups to which deliveries may be made concurrently
- delivery status
- locations of software and data files.

Facilities are also provided for automatic regression to revert to the previous version of the software in the event of any failure of distribution.

ICL's approach to Distribution is fully discussed elsewhere in this issue (Barthram and Howling, 1991).

7 Conclusions

We have seen that there are several trends in IT which are causing a significant shift in its use and in the expectations it generates. Some of these trends are not new, but today they are combining, so that the exponential growth in computing power may not be fully exploitable, were we not able to build better tools for its management and control. However, systems management technology is emerging and is already beginning to provide solutions to some of the problems.

But the analogy with operating systems should not be forgotten. It has taken nearly forty years for competing suppliers to offer equipment running the same operating system. The task facing us now is to manage a wide variety systems and services irrespective of supplier or underlying equipment. That is the challenge for the Nineties.

References

BARTHRAM, P. and HOWLING, T.D.: Distribution Management – ICL's OPEN Approach. ICL Tech. J. Vol. 7 No. 4, pp. 702–717, 1991.
GALE, A.C.: The evolution within ICL of an architecture for Systems Management.
HACKER, D.: Operations Management ICL Tech. J. Vol. 7 No. 4, pp. 741–750, 1991.
MAYNARD-SMITH, A.: The Network Management Domain. ICL Tech. J. 1991 Vol. 7 No. 4, pp. 764–779, 1991.
PICKWORTH, I.R.: Practical Experience in the Management of Retail business data flows in Distributed Computing Environments. ICL Tech. J. Vol. 7 No. 4, pp. 718–731, 1991.

Biography

G. E. G. Brown

Having gained a degree in Classics from Reading University in 1962, George Brown joined English Electric at Kidsgrove. After some years working on basic software, including a year with RCA in New Jersey, he joined CAP. This provided a varied experience of working on real user problems in many environments. He then found his way back into ICL via STC and is now Programme Manager of ICL's Systems Management activities.

The Evolution within ICL of an Architecture for Systems Management

Tony Gale
Architecture, Conformance and Verification Division ICL, Basingstoke, UK

Abstract

The evolution of Systems Management for Distributed Information Systems has strong parallels with the evolution of Operating Systems for computers. This paper provides an ICL perspective on this subject drawing on the need to meet specific business objectives while contributing to our understanding of the subject as a whole. The overall architecture is described and the underlying principles of management are applied to the planning, delivery and operational control of a complete Information System.

A System that is to be managed comprises hardware, software and the people who use the services supplied. Networks, Computers, Operating Systems and Applications are all involved and Systems Management must ensure that they all contribute to the success of the enterprise as a whole.

1 Introduction

An Information System comprises hardware, software *and* the people that use the services supplied; Computers, Networks, Operating Systems and Applications are all integrated into a whole. Management is the processes and tools that must be applied to the Information System in order to deliver an agreed service in a cost-effective manner.

The processing power, communications bandwidth and storage capacity available are still multiplying every few years. The limitations upon our ability to apply technology to the problems we wish to solve are human, not technological. As well as using the services supplied, people are involved in the development of systems and the provision of service on a day by day basis. This paper describes some of the activities of the Service Provider and describes how tools can be built that help him/her to do the job.

A brief historical perspective is provided to show how the architecture has evolved to meet the needs of specific types of enterprise.

The architecture is described at its highest level and the underlying principles are identified. Examples are provided of some of the solutions to management problems that have been solved by tools and processes based on this architecture.

2 Historical Perspective

As ICL moved into the world of Open Distributed Computing it became apparent that management of the computer resource had to be supplemented by other types of management. In particular the network that links computers together had to be managed to ensure that sufficiently reliable links were available across a wide geography. The most likely point of failure was the network and Network Management was there to repair service as quickly as possible. Although ICL did not market any significant amount of network products initially, some important concepts were well established in the field of Network Management and these became the basis for our approach to Systems Management:

Remote Operation – as distributed computing evolved it would be necessary to use the technology to bridge the gap between the skilled person and the component which needed attention.
Duplication – The time taken to fix a problem is usually too long and duplication of components in key areas is used to maintain service.
Monitoring – with many components unattended and geographically spread, the need for formal monitoring increases significantly.

2.1 Retail Management

One of the types of network ICL began to install in the early 1980's was the Retail network and, in particular, ICL followed a strategy of moving computing resource into the store linked by the network to a central mainframe. The reliable movement of information such as prices and stock requirements is vital for retailers and they require particular management features of their networks such as:

Automated Scheduling – the retailer requires to contact all stores every night and does not want to be dependant upon human resources being significantly involved in that activity.
Resilience and Recovery – the files moved are generally quite small and intermittent failures in the network infrastructure can be recovered by retry. The numbers of stores (several hundred in some cases) mean that a form of recovery will be necessary somewhere quite regularly.
Integrated Communication and Processing – much of the data is preprocessed, transferred, aggregated, analysed and then derived information is transferred elsewhere. This whole activity must be managed as a system-wide activity without human intervention (unless required by catastrophic failure).

Systems that meet these requirements must use concepts such as:

Prevention – by encouraging the analyst who designs the task to ask the "what if?" questions and providing an appropriate environment to encode the answers many operational problems can be solved without the need for operations staff always to be present.
Unattended Operation – the equipment exists in an environment where one cannot assume any degree of computer literacy or training. People are available to do straightforward tasks under direction, however.
Centralised Control – everything does not have to be done centrally but policy and coordination of its implementation must be focused centrally.

2.2 Management of Mail Networks

The development of mail networks in the 1980's has made everyone aware of the dependence they have upon a working system. When the standard infrastructure for communicating within and between enterprises is electronic it is imperative that adequate tools exist to help the service providers do their job. A particular feature of systems management that is emphasised by the mail network is the need to generate consistent parameters for many nodes whenever changes are needed to the configuration.

The generation of configuration parameters involves collecting a complete definition of the networked application into a database. Configuration rules implemented as analysis algorithms are applied to the database looking for potential problems in the definition, deriving routes on the way, and extracting the relevant parameters for each node in the network. Parameters are then delivered and installed over the network itself. The configuration of mail networks using this approach highlights some additional principles that underlie the development of tools for systems management:

Scale – the size of a processor and filestore to support Systems Management reflects much more the number of nodes being managed and the complexity of relationships between them than the size of the nodes themselves. This is because large nodes are already very well managed by the operating systems within them and it is only their relationships with other nodes, large and small, which contribute to the scale of a Systems Management solution.
Deskilling and Productivity – It is much easier to make a skilled person productive with Systems Management tools than it is to de-skill the task (en-skill the person!). Some problems of complexity do not go away very easily but it is possible to ensure that a skilled person is made much more productive by the development of Information Technology tools for Systems Management.
Culture Change – the move from an operational fix-it type environment to a more administrative preventive environment is a difficult change to make. Asking skilled people to use preventive technology is a threat to their power base which has been built on being the 'only one who could' in a fix-it

environment. This will happen in time as new opportunities become obvious but it can affect the buying patterns for Systems Management tools.

The experience gained in the development of generation management tools for Electronic Mail systems has been used to develop a specific Generator of parameters for the Inland Revenue. In their move to use the Government Data Network, Inland Revenue needed to move large numbers of users from one network infrastructure to another without loss of service. The tool developed in collaboration with the customer is described in (White, 1991).

3 ICL's Systems Management Architecture

The overall Systems Management Architecture that has evolved to meet the requirements and principles described earlier in this paper can be expressed quite simply.

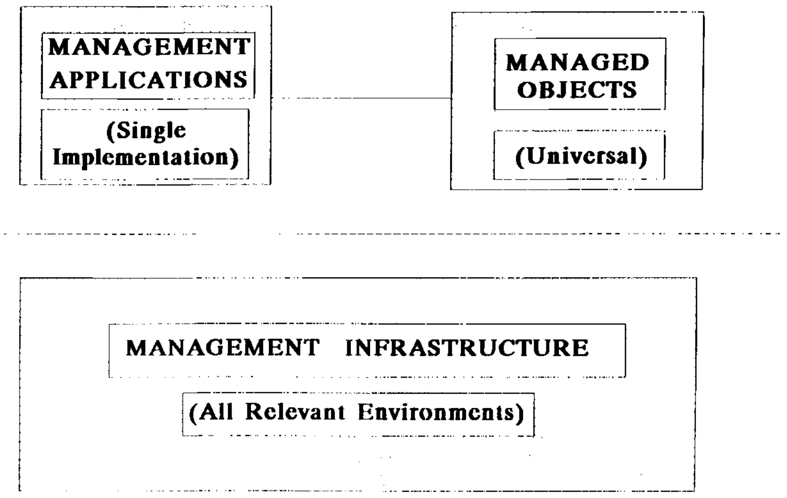


Fig. 1 Systems Management Architecture

Management Applications are tools which support all aspects of the Service Provider's Process (described in section 3.1). Each application has its own quite distinctive architecture. (Other papers in this journal describe some of these specific architectures)
Management Infrastructure promotes reuse of common components within applications and objects as well as providing the various communications services between applications and objects. These include Management Messaging (via a variety of protocols) to maintain real time contact, Bulk Data Transfer (via File Transfer and Mail protocols) to support major update and Virtual Terminal to allow for detailed interaction and diagnosis.

Managed Objects are well-defined, management-specific views of all the resources within a distributed Information System. By use of common libraries and appropriate application programming interfaces it is possible to make managed objects easier to develop, easier to extend to meet the needs of new management applications, and more common in their overall approach to management thereby reducing unnecessary diversity.

It is an important feature of the architecture that it can accommodate different management policies using a common set of tools; these will vary from high levels of management control and monitoring, centralised in one place, to distributed authority systems with feedback into the centre as and when necessary.

The policies adopted by an enterprise determine whether the processing involved takes place at the managing end or the managed end.

3.1 Management Applications

Systems Management Applications are a set of tools used by Service Providers in an enterprise to ensure that all users get the service they expect. To arrive at a specification for what tools are required it has been necessary to analyse the process of Service Provision and the goals that have to be achieved. As with similar technologies Systems Management is an agent for change within the organisation and so it has not been adequate simply to analyse what service providers are doing today but to forecast how they might operate if the right technology were available. This analysis has generated the Service Provider's Process for Systems Management which is at the heart of our chosen architecture. It is targeted at helping service providers meet the requirements of their users. It enables service providers to measure their achievements, take long term corrective action when solving problems that arise and to go on meeting the changing requirements day after day, year after year. The population of the process with tools involves the development and procurement of applications that make the service provider both productive and effective. These tools are the most visible output of the strategy but they cannot be built without the other two parts.

In summary the process is a continuing cycle of activity – there are no loose ends, each part feeds naturally on to the next. Which cycles are most active at any point in time reflects the part of the lifecycle that a system is in. The parts of the process are:

Operations – Controlling and monitoring the managed objects in the information system.
Problem – Diagnosing, fixing and preventing further occurrence of problems.
Capacity – Aggregating and reviewing information that reflects the performance and availability of the information system.

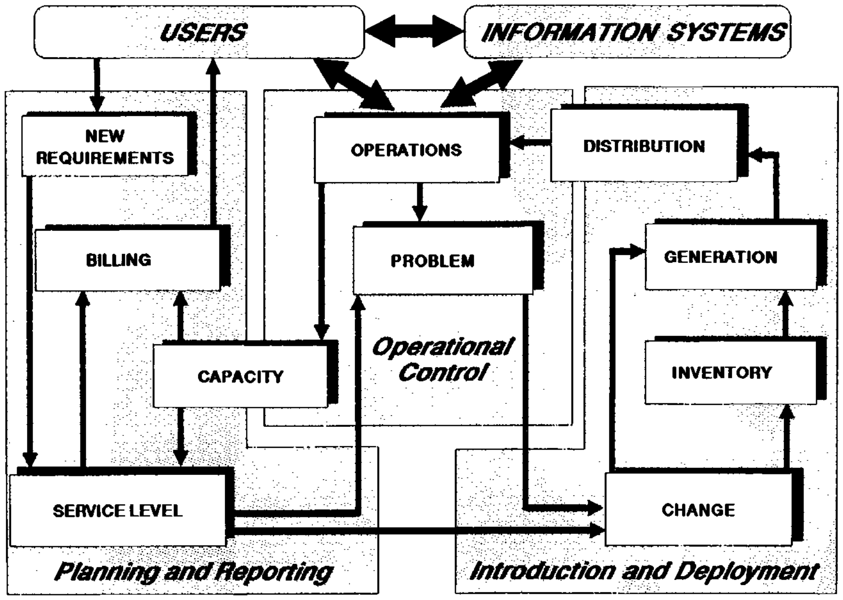


Fig. 2 Service Provider's Process

Distribution, Installation, Activation – Providing facilities to deliver the components of the information systems and bring them into operation.
Change – Managing the process of authorisation and review whereby all changes are made in a controlled manner.
Inventory – Maintaining up to date records of all logical and physical components that comprise the information system.
Generation – Providing the means to define, generate and subsequently change the configuration for the information system.
Billing – ensuring that users are charged appropriately for the services they receive.
Service Level Agreements – Maintaining and monitoring a clear statement of what service various (classes of) users require of the information system.
New Requirements – Responding to requests for new levels of service to be provided by the information system.

To populate this process with tools that meet all conceivable requirements is a major ongoing ICL strategy. It does not make sense to try and build the whole process at once and so priorities have been set according to our perception of our customers' most pressing needs.

Some customers have entered into a collaboration with us giving us access to their understanding of their most urgent requirements. One such collaboration is that between ICL and the Inland Revenue where we have been developing Generation Management technology with the Distribution Management required to support it as an agent for change within a very large network. This collaboration has lead us into the development of advanced datastores based on Object Oriented concepts and new graphical interfaces targeted to make the configuration designer a more productive and effective person.

Other parts of the process are being populated with tools developed against a generic statement of requirement, expressed by many customers and our internal service providers managing the ICL corporate network worldwide. To meet these requirements we have both internal development teams and good relationships with other suppliers who are developing components that help populate the process as a whole.

Many of these elements have been integrated into a single offering – the Open Systems Management Centre (OSMC) launched in 1991.

3.2 Management Infrastructure

The two major parts of the Management Infrastructure are Management Messaging and Bulk Data Transfer. In both these areas the requirement has been to add features to the basic communications facilities of the network that make the distributed system more manageable and more suited to the hosting of management applications.

3.2.1 Management Messaging The Management Messaging infrastructure which is released as a product under the name "Community Alert Management" provides facilities for:

Distributed Filtering – it is recognised that notification of all events is not needed in all circumstances from all nodes. However, it is important to encourage all components to generate events of note without making value judgements. Local distributed filtering supports this requirement by ensuring that only "important" information is routed to a manager (the manager, in turn, decides what is important).
Context Dependant Routing – it is recognised that not all messages need to go to one place and so routing facilities are provided to ensure that messages can be routed flexibly to one or more nodes around the network.
Connectionless working – although the facilities run over an underlying connection-oriented network, the facilities provide an application datagram interface to the components which use them. Events can be routed to whomever wants them, commands can be received from various managers, information can be routed to a specific destination.
Adaptability – additional functions can be added to the infrastructure to maintain logs of messages, present information on screens, generate automatic replies to events and any other facility that is required generally.

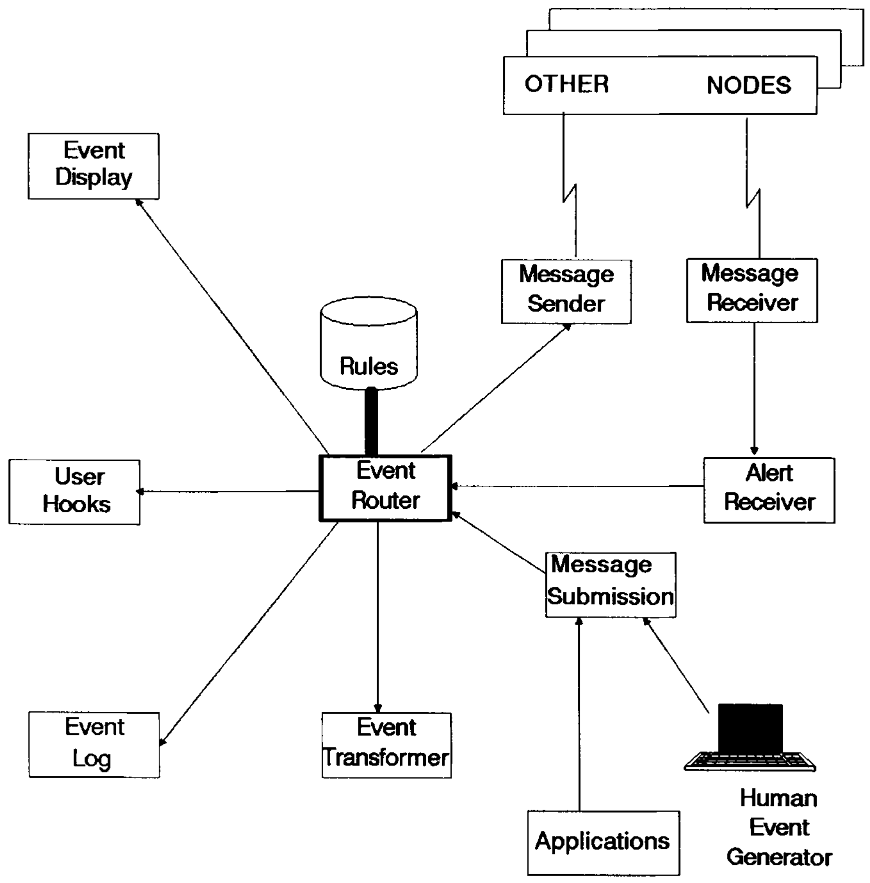


Fig. 3 Management Messaging

3.2.2 Bulk Data Transfer The Bulk Data Transfer infrastructure which is released as a product under the name "Community File Transfer" provides facilities for:

Schedule definition – a set of tasks including both the transfer and processing of data as well as message generation and receipt can be defined to happen with a desirable level of parallelism and a mandatory level of sequentiality. These tasks can operate on any number of nodes in the network with the schedule itself being distributed if required. The set is called a BATCH while individual sequences of tasks are called ACTION LISTS which comprise REQUESTS for the type of action to be performed. Special tasks can be specified to run at the beginning and/or end of each BATCH, ACTION LIST or REQUEST. These features are illustrated in Figure 4.

Schedule Control – the state of the whole BATCH is monitored at all times and facilities are provided to suspend/abandon parts of the schedule if required by external events.
Schedule Recovery – failures in any specific task can be recovered by automated retry with due consideration being given to timeliness (target time for completion).
Multiple protocols – the forms of file transfer used by the product are many and varied and are not preordained by the structure of the product. Thus it has been possible to migrate from ICL's original File Transfer Facility to the OSI FTAM standard and the other facilities such as NFS and RFS supported by UNIX.

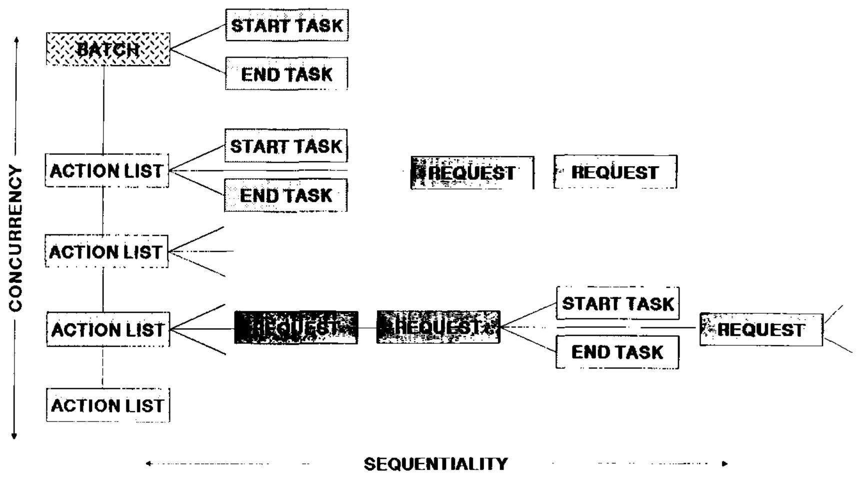


Fig. 4 Community File Transfer Model

Both Management Messaging and Bulk Data Transfer underlie all the management applications and provide the important link between the tools and the managed resources. The existence of a ubiquitous messaging facility makes it possible to design applications that expect to report incidents to a manager even if they are expected to run on an unattended machine. An example of this has been the Retail application GMS which runs in the store; this ensures that service providers at the centre of the network are always informed as to what is going on removing an earlier need for person to person telephone contact. Positive messages reflecting progress through a task are sent as well as alarms when something has gone wrong. The Bulk Data Transfer facilities are used whenever there is a need regularly to move data in many files between many nodes; the scheduling capabilities ensure that all tasks are completed without the need for human checks to be made.

3.3 Managed Objects

Managed Object is the term given to a view of a resource that has to be managed. Specific interfaces have to be built into the resource whether it is network, computer, operating system or application to enable it to be managed – the Managed Object is a specification of those interfaces.

An early example of the paramount need for ALL developers of systems components to build manageability into their designs is the MCU1. The MCU1 is an X.25-to-OSLAN gateway offering facilities for nodes on a LAN (generally large ones but not exclusively) to communicate over an X.25 WAN. The capacity of this node is such that many customers choose to run several of them on their networks in one or more locations. Because they are intermediate nodes in the network no one is very interested in them (until they go wrong!). As well as providing the many specific protocol services that are required by such a gateway the designers included many manageability features using standard Systems Management Infrastructure, such as:

- Software download from one location.
- Remote Configuration from one location.
- Alerting to one location.
- Remote Control from one location.
- Supply of statistics to one location.
- Flexibility to talk to one of many locations (for resilience).

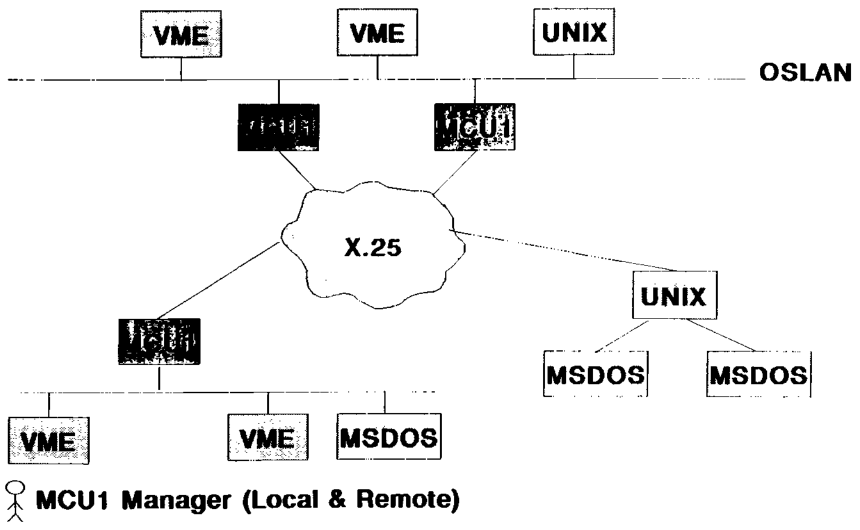


Fig. 5 Managed MCU1's in a network

It is fair to say that these features made the MCU1 much easier to live with in a distributed multi-nodal environment and supported our customers plans to run a more distributed network with mainframe servers in many geographic locations.

Without a commitment to Manageability, Systems Management does not get off the ground – to gain that commitment, *all* systems designers and application designers must envisage their components operating in a distributed system with only a limited skilled resource able to give them the attention they deserve.

MCU1 was an early example of how we learnt the value of manageability as was GMS2. One comment from a Retail customer using this product is that it tells him when things are going right, reflecting the need for Systems Management to give Enterprise Managers confidence that the complex systems they have built are operating as planned.

A Managed Object is a formal model of the various Management Interfaces supported by a managed resource; there may be one or more managed object models for a single resource based on the needs of various managers.

4 External Influences

As well as the need to meet the needs of our customers we have been influenced by external factors as well. In the mid 1980's IBM launched Netview and Netview PC indicating to the world that a more complete solution to Network Management was required. The major focus of this initiative was to link the management of network components into the existing set of applications developed to manage the SNA world; similar initiatives have emerged from Digital with Enterprise Management Architecture and AT&T with Unified Network Management Architecture.

Alongside these announcements OSI standards for Management have been under development. The speed of development was very slow in the early 1980's but more recently a focused work programme has resulted in various standards being ratified. A movement to implement these standards is now underway with the various OSI workshops working together to develop profiles. The UK GOSIP procurement handbook is being updated to provide advice on procuring Systems Management in an open form and NIST (the National Institute of Standards Technology) has a similar activity underway in the USA.

In line with its position as an Open Systems supplier, ICL has contributed through the British Standards Institute, European Workshop for Open Systems, European Computer Manufacturers Association, X/Open and others to the definition of open standards for systems management.

All three parts of our chosen Systems Management Architecture are increasingly affected by standards.

- Management Infrastructure uses FTAM, X.400 messaging and CMIP (Common Management Information Protocol) as well as the various other protocols required by existing and new customers.
- Manageability is based on the Guidelines for Definition of Managed Objects although at this time only a limited number of managed object models are available.
- Management Tools populating the overall process use the Management Functions standardised by ISO wherever possible but this is a very sparse set at present and is the area where most attention is now required.

5 Conclusion

This paper has shown, through historical analysis, how ICL has arrived at the Systems Management Architecture that it uses to drive through its strategy in this area. The effective and efficient use of various Information Technologies is dependant upon the provision of cost effective tooling for Systems Management; without that enterprises will not be able to accommodate or justify their planned uses of Information Technology at the current level of evolution let alone what the 1990's has to bring. The evolution of Systems Management will continue in all 3 areas with:

- Improved tools meeting the increasing needs of the Service Providers.
- Improved Infrastructure making the communications links between components and managers less obvious.
- Improved manageability with all designers putting manageability and functionality side by side in priority.

The goal of Systems Management must be for Service Providers to feel happy with placing all the components of an Information System wherever the enterprise requires them and for Enterprise Managers not to regard Service Providers as a cost which limits their vision of what can be achieved for their enterprise.

References

FULLER, A.R., Community Management for the ICL Networked Product Line. ICL Tech. J. Vol. 5 No. 4, pp. 652–664, 1989.

National Institute of Standards and Technology (NIST). Government Network Management Profile.

ISO/IEC 10040 Information Processing Systems – Open Systems Interconnection – Systems Management Overview.

WHITE, J.B. Generation of Configurations — a Collaborative Venture, ICL Tech. J. Vol. 7 No. 4, pp. 732–740, 1991.

Acknowledgement

UNIX is a registered trademark of AT&T in the USA and other countries.

Biography

Tony Gale

Tony Gale graduated from Trinity College, Cambridge and has spent 20 years with ICl in a wide range of roles – marketing, support, development and research. Since 1982 he has been responsible for the Architecture, Technical Strategy and Design of Systems Management within ICL.

He is the vice chairman of the Expert Group on Network Management in European Workshop for Open System (EWOS) and represents EWOS at the Network Management Special Interest Group within the OSI Implementors Workshop, USA.

Manageability of a Distributed System

Gareth I Jenkins
Systems Management Product Centre Mid Range Systems Division ICL

Abstract

As with any other form of management, Systems Management requires the commitment of the things that are being managed as well as the management system. This paper looks at the relationship between the management applications and the things being managed (or managed resources) from the viewpoint of the managed resources, and proposes a framework into which any managed resources can be slotted. The impact of this on the overall design process is also considered. The relationship to external work on standardisation and implementors agreements on interoperability of managed resources is also considered.

1 General

1.1 Scope

This paper sets in context the background to the manageability of distributed systems. A *distributed system* is any collection of computer systems, usually of different architectures from a variety of manufacturers, which are required to work together to provide services to their users in order to achieve the business objectives of the organisation to which those users belong. *Manageability* is what is required to enable the components of the distributed system to be managed remotely. All components will need to be manageable.

The main purpose of this paper is to set in context the background to manageability in relation to ICL's Systems Management Architecture and external standards, and in particular to describe what component developers will need to consider about manageability in their overall design process.

1.2 Introduction

The first question to answer is "why manageability?". That is, why is it useful to make a distributed system manageable? This subject is discussed more fully in [ACG], but the key reasons are as follows:-

- increasing dependence on IT for business advantage
- move towards distributed computing
- scarcity of skills

The increasing dependence on IT for business advantage implies that any loss of service is no longer just a headache for the IT department, but is a potential loss of profitability for the entire enterprise. For example, in the retail environment, if the automated checkout systems were to fail, it would be difficult to sell any goods, which would bring the whole business to a standstill.

The current trend is more towards a network of distributed IT systems rather than relying entirely on a central mainframe. These distributed systems also tend to be distributed geographically, into the various local offices or stores. This brings the power closer to the users. However, there is still a need for the distributed systems to interwork so that various work groups within the organisation can communicate (eg via electronic mail).

With the lowering of costs of hardware and software, the major cost now for an IT system is that of the skilled people required to operate the system. In addition there is a shortage of such skilled staff and so the more that can be automated then the more productive use can be made of this scarce resource. A simple way of aiding this productivity is to have a central pool of skilled staff who are responsible for the administration and operation of the whole distributed system, leaving the individual systems to run unattended with minimal local skills.

To achieve this way of working it is necessary to provide functionality in each component of the distributed system such that it can be operated and administered remotely. The provision of this functionality is what makes the components manageable.

The other key question is "why standardise?". Since the majority of distributed systems are made up of components from different suppliers, it is important that the central pool of skilled staff are able to access and manage all the components of the distributed system. This is much simplified if the same management applications can be utilised regardless of the origin of the component to be managed. This argument is similar to the one that has been generally accepted in support of OSI* for interworking between different vendors computers.

ICL's Systems Management Architecture (SMA) which is described in [SMA] consists of three basic building blocks as illustrated in Figure 1.

*Open Systems Interconnection.

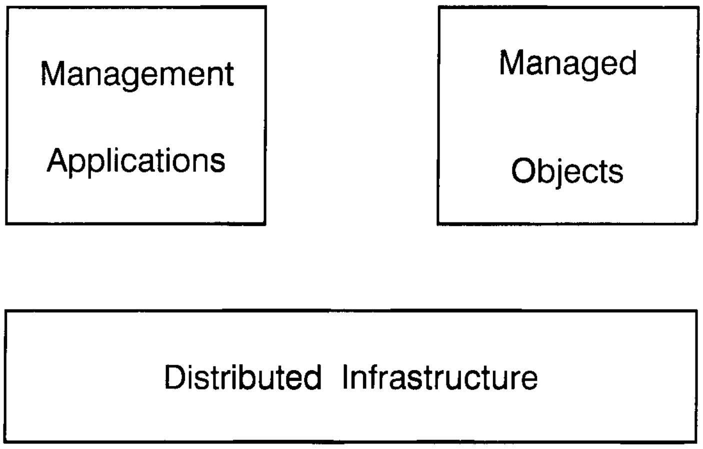


Fig. 1 ICL's Systems Management Architecture

The *management applications* are briefly outlined in [ACG] and some of them are described in more detail in related papers in this journal. The *infrastructure* is also described in [ACG]. The main subject of this paper concerns the third aspect of the architecture, namely that of *managed objects*, or to be more precise, how parts of a distributed system can become manageable.

2 OSI Model For Systems Management

ICL's Systems Management Architecture is aligned with the OSI standards for systems management, and so this section describes the OSI management architecture, and in particular the OSI management information model.

Over the last few years ISO† and CCITT‡ have been working together to define open standards for management of OSI components. The basic OSI management model has been developed to describe the way in which a managing process interworks with the things that are to be managed. This section provides an overview of that management model, followed by a more detailed look at the way in which the components of the distributed system that are to be managed need to present themselves to the management system.

2.1 Overview

The basic OSI systems management model (described in [SMO]) is illustrated in Figure 2. It has the concept of a *managing process* which manages a number of *managed objects* through *agent processes*. This managing process

†The International Standards Organisation.
‡The International Telegraph and Telephone Consultative Committee.

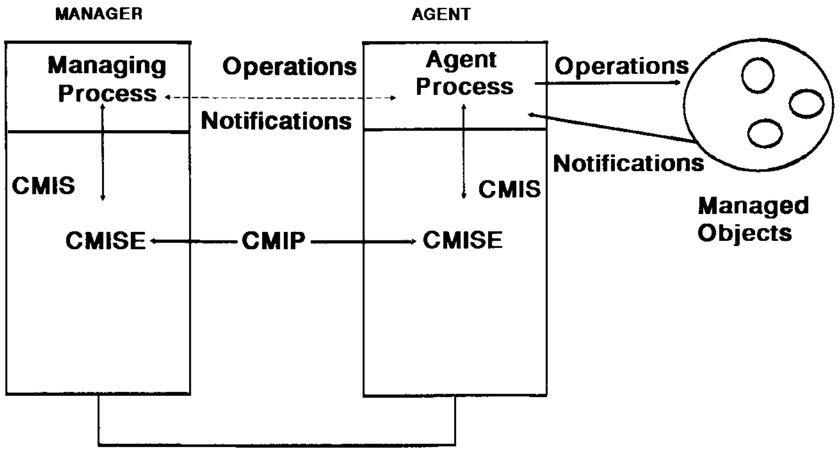


Fig. 2 Basic OSI Management Model

consists of one or more *management applications*, combined with various manual processes. Each of the managed objects presents a management view of some of the *managed resources* in the distributed system. As far as the OSI standards are concerned, this model applies to the management of OSI resources, such as transport connections, message handling systems, X.25 virtual circuits etc., however the OSI standards also recognise that the basic principles of the architecture can be extended to the management of other resources such as filestore, applications processes, modems etc.

The main area for OSI standards has been in the definition of standards for the communications between a managing process and an agent process. This is the *Common Management Information Service* (CMIS), which is supported by the *Common Management Information Protocol* (CMIP).

The CMIS services are defined in [CMIS] but may be summarised as follows:-

M-GET	requests attribute values
M-SET	alters attribute values
M-CREATE	creates a new managed object
M-DELETE	deletes an existing managed object
M-ACTION	performs an class specific action on a managed object
M-EVENT-REPORT	notifies a manager of the occurrence of an event
M-CANCEL-GET	cancels a previous GET

Each of these CMIS services is defined in terms of the managed objects upon which it operates (or in the case of M-EVENT-REPORT, from which it is emitted), therefore in order to be able to utilise the CMIS services it is necessary to have a common definition of the managed objects which are being used to model the managed resources.

The remainder of this section concentrates on the *management information model* which defines the managed object view presented by the managed resource to the management system. The standard which defines this model [MIM] is part of the *Structure of Management Information*; the other main part of which, the *Guidelines for the Definition of Managed Objects* [GDMO] defines the syntax for managed object definitions.

2.2 Managed Object Characteristics

Each implementor of a managed resource will have their own way to implement it, and it is not the purpose of standards to prescribe how this should be done. However in order to be able to manage similar resources in a common way, it is necessary to be able to view the different implementations of managed resources in a common way. This common view is provided by defining a common managed object model of that type of managed resource.

The way in which a managed object is managed is formally defined in the *managed object definition*. The managed object definition defines the *characteristics* of a managed object which consists of a list of the managed object's:-

- attributes
- operations
- notifications
- behaviour.

The *attributes* define the way that the managed resource is operating. The *operations* define the way in which the managed resource may be operated on by a managing process. These operations map onto the various CMIS services described above (except M-EVENT-REPORT). The *notifications* define how events are to be emitted by the managed object in order to inform a managing process of their occurrence. Notifications may be carried by the CMIS M-EVENT-REPORT service. The *behaviour* defines the way in which the managed resource will behave as a result of normal (or abnormal) operation or as a result of a management operation.

Having identified a managed object class, an attribute, an action or a notification, then it is necessary to register its *object identifier* so that the management applications and the managed objects can share the definition, and identify the objects in communications protocols. Also by registering definitions, existing definitions may be used as a basis for refinement when defining new managed object classes or for deriving other objects.

Not all aspects of a managed resource are of interest to management. For example unless there is (or is likely to be) a management application that requires information about a particular aspect of a managed resource, then it may be excluded. These aspects may be considered to be purely internal implementations, which are not of interest to the management system.

Therefore, a managed object provides a view of some aspects of one or more managed resources, and does not necessarily totally reflect the managed resource.

Similarly it may be useful to represent a managed resource in different ways to the management system, and this can be done by providing different managed objects that represent these views. The management system need not be aware that these managed objects are indeed modelling the same managed resource; however, the agent processes need to reconcile operations on the different views and resolve conflicts.

2.3 Object Oriented Concepts in Systems Management

The fact that the term used for the view presented to the management system is that of a managed object suggests that techniques and concepts from object-oriented design have been utilised in the definition of managed objects. This is indeed so, and so it is useful at this point to describe some of the Object-Oriented concepts that are used in the definition of managed objects. These are:-

- class
- instance
- inheritance
- containment.

For any managed object definition, it is probable that there will be many different managed objects that can use that same definition, for example there will normally be many message transfer agents in an electronic mail system. These managed objects are all called *instances* of the same managed object *class*. It can be seen, therefore, that managed object definitions define a class of managed object. Each managed object class is defined in terms of its characteristics.

Inheritance is concerned with defining new classes as a *refinement* of some existing class. A refinement of a class allows the definition of the class to be extended for example by the addition of some more attributes or by enhancing the definition of an operation. The main benefit of inheritance is the ability to reuse existing specifications when defining new managed objects.

Containment is a relationship between managed object instances (the "is-contained-in" relationship). For example a *job queue* is contained in a *job scheduler*, which in turn is contained in a *managed system*. An important aspect of containment is that a contained object cannot exist unless its containing object also exists. This implies that it is not possible to delete an object without first deleting all the objects contained within it.

Containment is important because it is used in the naming of managed objects. Managed objects all need to be identified in some way so that the agent process can identify which managed object an operation is directed at, and so that the managing process can tell from which managed object a notification has been emitted. Rather than ensuring that each managed object has a simple unique name in the distributed system, a hierarchic name is formed (analogous to that used to identify files within a UNIX^{TM} directory structure). This is done by using the containment relationship, and ensuring that each managed object has an attribute which uniquely identifies it within its containing object.

The rules as to which attribute of a managed object is to be used for naming it when it is contained within another object of a specific class are held in a *name binding*. A unique name for any managed object can thus be constructed by concatenating each of these relative names to get a full name starting at the root of the overall naming tree.

It is important to understand that inheritance is a hierarchy of *classes*, while containment is a hierarchy of *instances*.

3 Managed Object Architecture

This section expands upon the managed object part of the Systems Management Architecture introduced in Section 1.2 and clarifies the functions of the various components of the managed object architecture and the role of the three interfaces that are defined. The architecture is summarised in Figure 3.

3.1 Infrastructure

This is the standard management infrastructure described in [ACG]. It consists of three parts, a messaging infrastructure, a bulk data transfer infrastructure and a virtual terminal access infrastructure. Its provision is a fundamental part of any manageable or managing platform.

3.2 Agent

This functionality corresponds to that of the OSI agent process described in Section 2.1. It is responsible for:-

- handling the communications interface with the managing process on behalf of a number of managed objects
- access control
- scoping and filtering of operations
- discrimination of notifications
- logging.

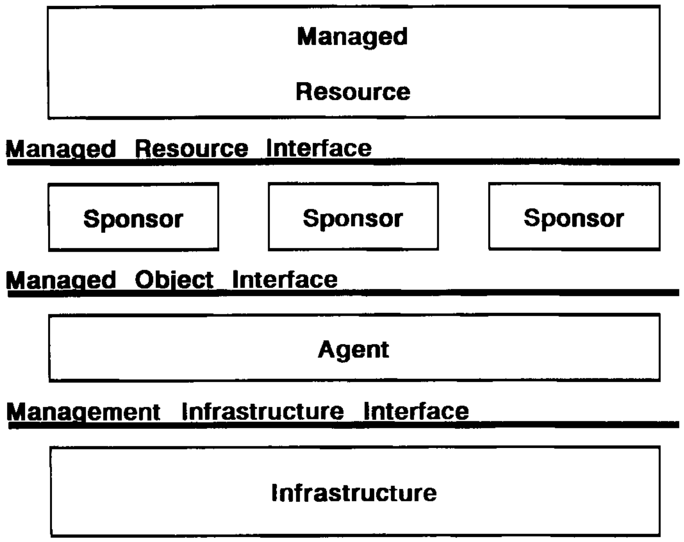


Fig. 3 Managed object architecture

3.3 Sponsor

This is a set of library functions that assist in the implementation of managed resources. The intention is that by utilising these routines, managed resources can have a very simple view of management. This also enables common functionality to be reused. All the sponsor libraries are available to any type of managed resource.

Sponsorship falls into two categories:-

- Simplifying the management interface
- Adding value to the basic managed object model

3.3.1 Simplifying the management interface: this enables a simple interface to be provided. For example for raising an alarm, the sponsor code would be responsible for formatting a proper notification to pass on to the agent. Similarly, when an action sponsor receives an action, it can convert it into an appropriate function call on the managed resource.

3.3.2 Adding value to the basic MO model: this enables added value to be made to the functionality provided by the managed resource. For example, if a managed resource regularly updates the value of a meter held by the sponsor, then the sponsor could provide thresholding§ or tidemarks on that meter.

§Thresholding and tidemarks are defined in [DMI].

3.4 Managed Resources

Managed resources can be viewed in a number of different ways. This is discussed further in Section 4.

3.5 Managed Resource Interface

This is the key interface as far as a managed resource developer is concerned. It is the interface that is provided by the various sponsor libraries, and enables them to be incorporated into the overall managed resource in order to provide a managed object view to management applications. This interface is likely to be proprietary to ICL, though it should also be common functionally (though not necessarily syntactically) across all ICL platforms.

3.6 Managed Object Interface

This interface represents the managed object boundary. It is likely to become a standard interface, and work is going on within X/Open** to define such an interface. It will be the main portability interface across all open platforms.

3.7 Management Infrastructure Interface

This is the interface used to access the infrastructure. It is used both by the management applications and the agent. This is an applications programming interface (API) implementing CMIS. However to maintain compatibility with our existing infrastructure (such as CAS††), it will also need to continue to support that.

3.8 Simple Platforms

Most of the above discussion has assumed a fairly sophisticated platform. For simple platforms (such as an MS-DOS PC), a simpler approach is required. It is important that the managed resource can be implemented in the same way, and so the managed resource interface will need to be maintained. However much of the functionality of the sponsor and agent could be provided on a separate server on behalf of the simple platform. Communication between the server and the managed resource interface needs to be defined, and conceptually exists within the sponsor functionality.

Another aspect of a simple platform or a simple managed object is that much more work will need to be done by the management application. For example, instead of having the sophisticated sponsor raising event reports only for serious problems, there will be a deluge of event reports for all

**X/Open is an independent company which defines a set of standard interfaces for application development, thus easing the portability of applications.
††the Community Alerting Subsystem.

sorts of trivia. This will in turn put a load on the communications channels and infrastructure. However it does mean that the agent and sponsor code can be much simpler. This is a trade off that needs to be made in the implementation of a platform. This is the way of working that has been adopted by other management architectures such as SNMP‡‡.

4 Managed Resource Framework

In order to try and get some sort of consistency across the definition of managed object classes a Framework is useful into which new classes can be fitted. This framework is described in more detail in [GENMO]. Once a number of classes have been defined, then the library of registered classes can provide this framework, but to start off with it is useful to model what an abstract distributed system that is to be managed looks like.

Managed resources can be viewed in a number of different ways as shown in Figure 4, and so these different views can be used to construct the abstract

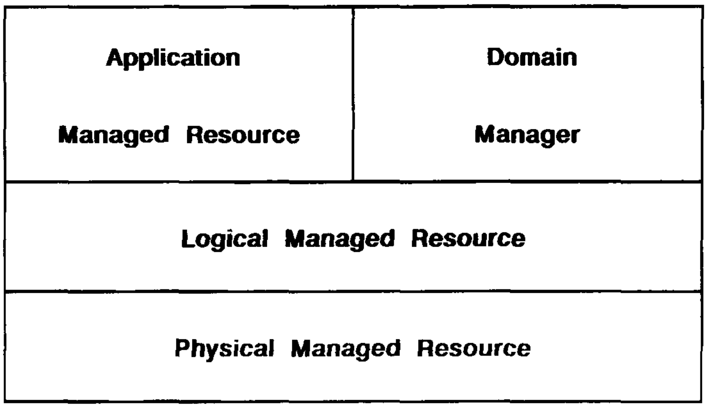


Fig. 4 Views of managed resourses

framework. Firstly they can be viewed at a very physical level, ie in terms of the bits of hardware that they use. Secondly they can be viewed at a generic logical level, ie as capsules. This view provides a similar model for all applications as seen from the underlying platform. Thirdly they can be viewed in terms of the functionality they provide, ie in terms of a business application. A special case of this is that of a domain manager, which manages a specific management domain, but in turn presents a management view of that domain to another domain. These are described in more detail below.

The benefit of having these varying views of a managed resource is that some of the views provide a basic level of management of the resource, without

‡‡The Simple Network Management Protocol, part of the TCP/IP suit of protocols for simple management of the Internet.

requiring the implementors of the components to be aware of management. An example of this can be shown by the fact that part of the physical view of a resource is the system processes (or VMs) that are used when that application is running. By ensuring that the platform upon which the application is running monitors these processes, some indication as to whether the application is running or not can be found by a management application without the managed application being aware of it.

4.1 Physical Managed Resource

Physical managed resources correspond to the view provided by the platform. It has such managed objects as Hardware, Processes, Filestore, Software containers, Comms connections etc. It is at this level that Teleservice is primarily aimed.

4.2 Capsule Managed Resources

Capsule managed resources correspond to the view provided through a capsule. It has such managed objects as Work Managers, Data Managers, Process Requirements. All applications can be viewed in a fairly generic way as a capsule, and any management applications will not need to understand the specific structure of the managed resource. Currently there is no capsule infrastructure to present this view, but as the concept of capsules is developed, this view of the managed resource will be presented to management applications without action by the application developer.

4.3 Application Managed Resources

Application managed resources correspond to the application specific view of the managed resource. The structure of this is very much business application specific, and so management applications will need to be tailored to the specific business applications that are being managed.

4.4 Domain Managers

The concept of a domain manager is not new. It represents the fact that there are a number of management applications that concentrate on the management of some particular aspect of the total distributed system, such as the management of a specific business application (eg electronic mail), or some particular area of technology (eg a LAN manager). The reasons for domaining are many, and the way that a community may be split into domains is discussed in [SMA]. A domain manager will look after the management aspects of a domain, and will allow a management application in a different domain to manage some aspects of its managed resources, by making them visible as managed objects.

The domain manager may manage its domain of managed resources as a normal management application following SMA, thus making the whole architecture recursive. For example a Message Handling management application that manages the resources used within a message handling system. On the other hand the domain manager may use a completely different mechanism for management. An example here would be an SNMP gateway management system.

5 Manageability in the Design Process

5.1 The Definition Process

Any design process goes through a number of phases and it is necessary that aspects of manageability are considered during each of these phases. The key phases can be described as:-

- feasibility and outline design
- functional specification (high level design)
- detailed design and implementation

Within ICL these stages are formally reviewed by a *phase review process.*

Phase 0 of the phase review process is about feasibility and outline design. Therefore managed object classes need to be identified at this stage, and also statements as to whether these classes already exist, or need to be defined, must be made. If new managed object classes need to be defined then statements are also required as to whether there are any existing classes from which they can be refined.

Phase 1 defines the functional specification of the development. At this stage the managed object classes must be fully defined and registered, and any dependencies on management applications must be declared.

Phase 3 is about implementation, and so this is the time at which any conformance testing will be done to ensure that the new product conforms to its managed object definition, and can be managed by the appropriate management applications.

Let us look at the total process for defining the manageability of a new (or existing) "thing". The main stages are as follows:

1 identify the scope of what is to be managed.
2 break it up into component managed objects.
3 relate these to existing managed object classes.
4 consider the relevant management functions.
5 consider naming.
6 fill in the managed object templates.
7 submit the templates for validation, harmonisation & registration.
8 implement.

Stages 1 & 2 are carried out during Phase 0, Stages 3 to 7 are part of Phase 1, while stage 8 is Phase 3. These are described in more detail in the sections below. Throughout the description use will be made of an example to illustrate the principles involved, the example being the management of filestore.

5.2 Identify the scope

The first thing that is required is to understand the precise scope of what the management problem is. This is as much an exercise of deciding what is *not* to be managed as that of deciding what *is* to be managed.

In our filestore example, we could try to manage the entire filestore of an organisation, or a department within an organisation. Alternatively we could be concerned with the management of the filestore of a particular platform. For our example we will consider the management of all the filestore belonging to a single MS-DOS PC.

5.3 Identify Component Managed Objects

The next step is to break the problem down into an understanding of the component managed objects that may exist within the problem space. This is probably the most difficult step of all. Looking at the generic framework for manageability [GENMO] should help here since that defines some general breakdowns of managed objects.

In the filestore example, we see that filestore is part of the physical view. In terms of our MS-DOS filestore, we can consider each logical drive (or partition) as corresponding to filestore. This managed object will be sufficient for this example, but others that could have been considered are directories within the drives, or magnetic media used for backups.

5.4 Relate to Existing Managed Object Classes

The next step is to relate the managed objects identified above to existing managed object classes defined both within ICL and externally. This is to ensure that the resulting managed objects can reuse as many characteristics as possible from existing managed objects, thus ensuring that they can be managed by existing management applications.

Within ICL there are the generic managed objects defined in [GENMO] and the register of managed object class definitions. External registers of managed object class definitions are described in Section 6.

In the case of filestore, in addition to the generic managed object class of filestore, there are also the specific managed object classes of UNIX filestore and VME filestore.

5.5 Consider Relevant Management Functions

The next stage is to consider in what way it is required to manage these managed objects. This may well be a phased approach, since it is probably desirable to manage all aspects of the problem, but resource constraints require that only a subset of the aspects can be addressed initially. This is very much a business driven rather than a technical decision, though availability of appropriate management applications may also be an influence.

There are many aspects of Filestore that could be managed. Examples are:

- the ability to monitor its usage in terms of how full it is
- the ability to monitor its usage in terms of statistics as to frequency of access and access times
- the monitoring of error rates
- the location of backups (or archives)
- the commands required to carry out backups and restores
- the deletion of "deadwood" (files that have been backed up and not accessed for a long while)

5.6 Consider Naming

All managed objects need to be named, and this naming needs to be done relative to some other managed objects so that they may be uniquely identified by any management applications. This requires the identification of naming attributes on each managed object and the name bindings to be used relative to their superior objects in the naming tree. The key thing here is to identify the scope across which a name needs to be unique, since that will identify the superior object.

In the filestore example the naming can be taken directly from the generic filestore managed object that has already been identified. This shows that filestore is named by the managed system upon which it resides. In the case of a MS-DOS Filestore (ie a drive), its name is constrained to be a single character, and so this needs to be documented as the permitted values of the file-system-id attribute.

5.7 Fill in the Managed Object Templates

The next stage is to define formally the managed object template for each managed object. In this all the attributes for the managed object are identified, together with the actions and notifications. It should be possible to inherit the majority of the characteristics from other like-managed objects which have been identified as part of step 3.

These formal definitions are too detailed for inclusion in this paper, but are available from the author on request.

5.8 Validation, Harmonisation and Registration

Having formally defined the required managed objects and all the required characteristics, then it is necessary to submit the definitions to the harmonisation & registration authority. The purpose of this is to ensure that different groups within ICL don't invent separate managed objects for what is basically the same thing, and to ensure that the maximum reuse of characteristics is achieved through inheritance. This may involve identifying some characteristics of the new managed objects that would more appropriately be defined on some higher level managed object (within the inheritance tree). This will have no effect on the manageability of that particular managed object, but may allow other similar managed objects to include those characteristics.

In the Filestore example it is likely that the various characteristics identified are likely to be appropriate for other filestores. Therefore the harmonisation process may modify or generalise these definitions and add them to the generic filestore managed object definition. There may then be a need to have some MS-DOS specific variants of these, and also other filestore managed objects (such as on VME and UNIX) will need to consider the implementation of these characteristics.

5.9 Implement

Good luck!

6 Relationship to External Standards

Currently a number of different standards bodies are busy defining a variety of managed objects. At this time there is no formal coordination of their work other than the fact that many of the participants are common to the various bodies. The principle definers of standards are:

- OSI Network Management Forum
- POSIX 1003·7
- NMSIG
- ISO/CCITT – Generic Management Standards
  – Network Layer Standards
  – Transport Layer Standards

As this area is rapidly evolving at this time it is not sensible to try and catalogue those managed object definitions that currently exist. However such a list has been produced and is available from the author if required.

7 Conclusions

As can be seen the subject of manageability of distributed systems is large. However it is possible to define a fairly simple process which the developers of components of a distributed system can follow to ensure that the components may be managed in a well defined and well understood manner.

At this time we are in the early stages of using and refining this process model, but in the future we expect to see a growing number of manageable applications being developed. What we have learnt in the development of the Open Systems Management Centre (OSMC) shows that this is the correct way forward, and also it is the way that will lead to interoperable distributed systems.

Acknowledgement

UNIX is a registered trademark of UNIX Systems Laboratories, Inc. in the USA and other countries.

8 Document Cross-reference

[ACG]		Gale, A.C.: The evolution within ICL of an Architecture for Systems Management. ICL Tech. J. Vol. 7 No. 4, 673–684, 1991
[CMIS]	ISO 9595	Common Management Information Service
[DMI]	ISO 10065–2	Definition of Management Information
[GDMO]	ISO 10165–4	Guidelines for the Definition of Managed Objects
[GENMO]		ICL Generic Managed Objects
[MIM]	ISO 10165–1	Management Information Model
[SMA]		ICL Systems Management Architecture
[SMF]	ISO 10164	Systems Management Functions
[SMO]	ISO 10040	Systems Management Overview

Biography

Gareth I Jenkins

Gareth Jenkins graduated from Downing College, Cambridge with a degree in mathematics in 1973. He then joined ICL in where he has worked in a number of different roles. Initially he was involved in sales support, primarily for central government, before secondment to ICL New Zealand in 1977 in order to set up a Systems Maintenance Centre in Wellington. In 1979 he moved to product development involved with IDMSX and TPMS, becoming the manager of the design unit. In 1985, when development of these products was moved to Manchester, he transferred to Systems Management, where he continues to work as a systems designer.

Distribution Management – ICL's OPEN approach

Phil Barthram and Tim Howling
Mid Range Systems Division, ICL, Basingstoke, UK

Abstract

ICL's Distribution Management product set is responsible for managing the distribution of all types of 'system management' objects; software, documents, configuration data etc. around a community of networked computers. Transmission may be either by network or exchangeable media.

Central to this product set is a Management Application, conformant to existing and emerging standards and capable of managing the distribution of 'system management' objects across a multi-vendor network of UNIX^{TM} and non-UNIX platforms. The product set is based upon an architecture that promotes scalability thus meeting the needs of small to very large networks.

This article describes the market requirement, the approach taken in terms of requirements analysis and architecture, the functionality of the product produced and, finally, the way the solution was engineered and the products on which it was built.

1 Introduction

The rapid increase in the number of distributed systems running common software can cause heavy cost increases for customers in maintaining these systems. ICL's *Distribution Manager* product set is ICL's response to keeping control of such costs.

A distribution management product is designed to manage (in the sense of both administration and control) the distribution of all types of '*system management*' objects; software, documents, configuration data etc. around a community of networked computers. The transmission may take place via the network or on some exchangeable medium (eg magnetic tape). The types of uses at which a distribution system is aimed are very diverse, ranging from distributing new software versions and patches to pricing information for retail outlets.

ICL has produced a product that is aimed at meeting the above need. It is now an integral part of a suite of ICL systems management products.

Central to this product set is a *distribution management application*, that conforms to existing and emerging standards in the area and is capable of managing the distribution of software, configuration information or other business data across a multi-vendor network of machines running both Unix and non-Unix operating systems.

The product is based upon an architecture that promotes both technical and financial scalability, meeting in other words the needs of small to very large networks.

This article describes the market requirement for *Distribution Manager*, the approach taken in terms of requirements analysis and architecture, the functionality of the product produced and, finally, the way the solution was engineered and the products on which it was built.

2 The market requirement

2.1 The management of software distribution

Ten years ago the typical network consisted of dumb terminals surrounding mainframes. While these types of network still exist, contemporary networks more often contain many remote intelligent workstations and servers, each with their own software. Consequently the distributed nature and large number of machines in these networks present a significant management problem.

A possible solution is one where teams of specialists travel the country in a fleet of vans carrying out complex upgrades of new versions of software on each workstation. Whether this option would be cheaper than an alternative where local skills were available to carry out software management after the postal delivery of media is academic. In either case the cost of the operation would be huge and several practical problems would arise. One such example is represented by upgrading and bringing into operational use simultaneously on all end-systems a new version of some software. This would not apply to items such as word-processors or spreadsheets where the use is essentially local. However communications software or distributed applications often rely on all components being at the same version. This could not be achieved reliably where local staff were individually made responsible for the change.

There can be other problems (for example scheduling) associated with managing software on networks of intelligent terminals. Thus these other problems together with the costs involved are major factors for any installation to consider when planning a network. In other words the ability to manage software remotely, that is distribute it and then subsequently install and activate it, becomes mission-critical for both ICL and its customers.

As well as providing customers with a solution to the sheer logistics and size problems associated with the management of software across large intelligent networks, automated software distribution also provides other benefits:

- reduced cost associated with the overall approach
- increased system integrity when compared with manual, ad-hoc methods
- scarce skilled resource freed to do other work
- rapid response to problems, error correction made possible
- cost savings associated with central purchase of software
- reduced chance of software viruses from unofficial software
- inventory of software maintained automatically.

If ICL (or any other manufacturer) were to distribute its software automatically to customers in this way, it would realise the following benefits:

- knowledge of who has what version of software
- reduction in manufacturing cost
- reduction in distribution costs
- reduction in administration cost
- improved control of all distribution activities (delivery, installation etc)
- improved 'ease of buying' and thus software revenues and cash flow.

2.2 ICL's response to this opportunity – The key attributes of the solution

Traditionally distribution management products have been "platform specific" with both management centre and end systems all being of a common type resulting in a platform-specific solution. Our product has a single, generic open application 'Distribution Manager' that manages distribution to an entire network of multi-vendor, UNIX and non-UNIX platforms.

2.2.1 The components supporting this generic application are:-

- *Distribution Database* This database contains a comprehensive inventory of the network and the machines to which software has been distributed, as well as identifying whether or not the software is installed and if so when.
- *Transfer Engine* This component contains a combination of computing power (filestore, processing and communications capability) with the addition of the infrastructure needed to run a distribution schedule: Community File Transfer. This component provides scalability, being replicated across as many machines as is required to support the network.
- *Remote Distribution Application* Resident on each end system, this provides a degree of control and feedback to the 'Management Application' improving that which can be achieved by basic infrastructure alone and reporting the success or failure of changes in software state, for example during a software upgrade.

2.2.2 The flexibility inherent in this architecture provides a number of unique benefits:-

- *Single "look and feel"* The development of a single application for all types of platform has a number of advantages. Not only does it minimise support and development costs which are ultimately born by users; it also offers the ability to distribute to a number of platform types in a multi-vendor network, concurrently, from an application that offers users a single and consistent look and feel.
- *Scalability* As the size of a network increases so must the size of the management system supporting that network if it is to maintain the same performance. By increasing the number of nodes in a network that serve as transfer engines the size of the distribution management system can keep pace with the increase in network size. The overall bandwidth of distribution that can take place increases.

The separate datastore component can also be replicated or placed on a hardware platform that provides performance and filestore capacity consistent with the scale of the users' network. These techniques ensure that a single product offers a solution that is technically and financially scalable, in other words a solution that can grow with its users.

- *Multi-vendor at low cost* The Remote Distribution Application itself splits into two parts. A generic and therefore portable Remote Distribution Agent and an operating system specific Product Set Agent. The flexibility to interwork with different end-system processes in this design coupled with the fact that the platform specific code is concentrated in one small area offers a solution that can be ported to different platforms at low cost.
- *Reuse* The Managing Application has its own application programming interface for programmers wishing to integrate Distribution Manager with other products. This allows future versions of Distribution Manager to be integrated into and re-used by other more customer specific applications.

3 Distribution Architecture

If any complex set of software products is to be successful it must be built upon an architecture that promotes openness, scalability and reuse of existing proven *infrastructure.* The Distribution Management architecture is a component part of the Systems Management Architecture and shares some of its infrastructure components such as CAM and CFT with other management components in the architecture.

As can be seen from Figure 1, the Distribution Management Architecture is split into two parts, a system-dependent part and a system-independent (or generic) part. The system dependent part comprises the Generator, which

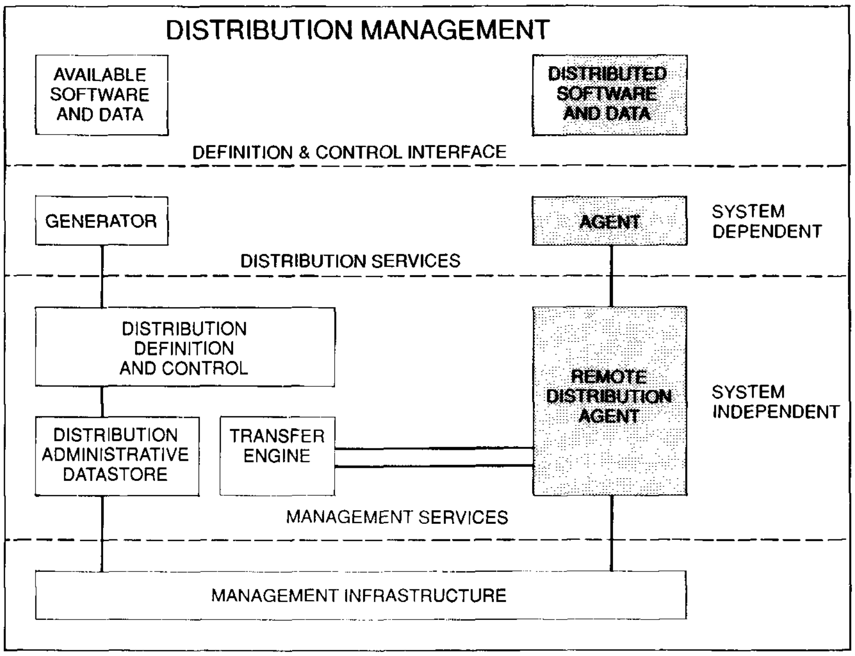

DISTRIBUTION MANAGEMENT
AVAILABLE SOFTWARE AND DATA
DISTRIBUTED SOFTWARE AND DATA
DEFINITION & CONTROL INTERFACE
GENERATOR
AGENT
SYSTEM DEPENDENT
DISTRIBUTION SERVICES
DISTRIBUTION DEFINITION AND CONTROL
REMOTE DISTRIBUTION AGENT
SYSTEM INDEPENDENT
DISTRIBUTION ADMINISTRATIVE DATASTORE
TRANSFER ENGINE
MANAGEMENT SERVICES
MANAGEMENT INFRASTRUCTURE

Fig. 1 Distribution Management Architecture

encapsulates the Management User's software and data in system independent envelopes, called 'filesets' through the 'Distribution Services' interface and the Agent which opens the envelopes and uses the software and data. The system-independent level is composed of the Remote Distribution Agent, Transfer Engine, Distribution Administrative Datastore and the Distribution Definition and Control components. These use the Management Infrastructure to move, replicate and invoke operations on the system independent envelopes in response to schedules provided by the Management User.

3.1 Generators and Agents

Generators take software and data files from the user, generate additional *control files* and use the 'Distribution Services' interface to define these software, data and control files to the 'Distribution Definition' component. The control files contain information about non-file aspects of a product as well as sequencing information for operations. A set of files defined together in this way is called a 'fileset'. A fileset can also support operations which are also defined to 'Distribution Definition' through the 'Distribution Services' interface.

Generators and Agents operate in pairs, the format of the control files and how operations are carried out being specific to a generator/agent pair.

A fileset may contain two types of file, ordinary files and logical files. If a fileset contains only ordinary files then the contents of the files found in one copy of the fileset will be the same as in another copy. This would typically be the one used when software products are delivered to a group of machines.

Logical files allow different files to be delivered to different machines. Typically, a logical file is a configuration file which must be different on each end system but where its logical purpose, say to configure its network, is the same. The distribution system manages and hides the underlying complexity inherent in having object-specific instances of what is logically a single file.

3.2 Distribution Definition and Control

'Distribution Definition' receives descriptions of filesets and operations from generators and places them in the 'Administrative Datastore'. 'Distribution Control' allows the user to specify a schedule which defines relocation and movement of filesets and operations to be performed on the filesets. This schedule, like many forms of job control, allows operations to be sequenced and for one operation to be dependent upon the success of a previous one. Schedules are automatically checked by the management system thus preventing avoidable human error. As a schedule may cause millions or even hundreds of millions of operations on thousands of end-systems it is important that avoidable errors be eliminated. These schedules are also held in the 'Administrative Datastore'. Schedules are run and monitored by 'Distribution Control' and the effects of relocation, movement and operations are recorded in the 'Administrative Datastore'. 'Distribution Definition and Control' has an Application Programming Interface (API) allowing itself to become a component in the larger Systems Management Architecture, for example, acting as the distribution mechanism under a 'Configuration Manager'.

'Distribution Definition and Control' replicates and distributes filesets throughout a managed network. It then controls the installation and activation of these filesets in order that changes are made to the software and application services running in that network.

'Distribution Definition and Control' administers and controls the state of these filesets.

'Distribution Definition and Control' supports four kinds of user:-
- *Fileset Controller* Responsible for defining filesets in terms of real objects such as files and executable tasks.
- *Network Configuration* Responsible for defining logical end-systems, routes between them and groups of them for example a group called All_in_London defining all the machines in the London area.

- *Distribution Administrator* Responsible for defining which filesets are required at which end-systems or end-systems groups and when. This results in the production of a schedule.
- *Distribution Controller* Responsible for taking the schedule which has a start time and an advisory completion time, initiating the running of the schedule and then monitoring the schedule of movement, replication and software installation operations so giving the user confidence in the operations taking place.

3.3 Transfer Engines

The 'Transfer Engine' effects the replication and movement of and invokes operations on filesets by controlling the Management Infrastructure. A 'Transfer Engine' may be any machine in the network with the right combination of computing power (filestore, processing power and communications capability) to act as the distribution centre for a large number of other machines. In addition it requires the infrastructure to run a distribution schedule, that is CFT and CAM (see para. 3.6 below).

The combination of the 'Transfer Engine' and 'Distribution Control' allow distribution to take place. Earlier architectures did not separate these two concepts and traditionally placed both these components on a single machine. By increasing the number of nodes in a network that serve as transfer engines the overall bandwidth of distribution that can take place increases. By this technique the architecture offers a solution that is scalable.

3.4 Remote Distribution Agents

Transfer Engines can replicate and move filesets by controlling the Management Infrastructure. To invoke remote operations there is two-way communication between the Transfer Engine and the Remote Distribution Agent through the Management Infrastructure.

3.5 Administrative Datastore

The Administrative Datastore is active and receives communication from the Transfer Engines via the Management Infrastructure. As well as providing a passive store to the Distribution Definition and Control Information.

3.6 Distribution's Management Infrastructure

This consists of the services of the 'Managed Bulk Data Transfer Service' (Community File Transfer – CFT) and the 'Management Messaging Service' (Community Alert Management – CAM).

4 Flexible incremental development

A number of separate organisations/projects both internal and external to the company were identified in 1989 as having need for a 'Distribution Management System'. The list is not exhaustive but at least includes:

- major ICL corporate programmes
- large external customers
- vertical business units
- internal service providers
- internal integration units

Given product requirements as they were understood and a team of the size that was planned for the production of the Distribution Management product the development life cycle time of a 'High functionality' product was likely to be 18+ months. A more traditional approach to developing a product would follow the route of defining these requirements and, following a single sequential development, culminating in a product 18+ months later.

Within this timeframe three external forces cause functional requirements to change. These are:

- technological change; allowing new development methods and techniques to be employed, so allowing requirements previously thought to be infeasible to be met.
- existing customers evolving their views of the functionality needed.
- new customers bringing new views of the functionality needed.

However by this time a committed product plan would be underway with planned end dates. It would be difficult to change direction. A possible outcome in 18+ months would be a product built to time and to a specification, that nobody would want to use. To cope with this situation and be more responsive to a changing set of requirements a development approach based on prototyping was adopted. In general terms a prototype may be created in order to answer any one or more of the following questions:

- marketing – What is the customer reaction to an idea for a new product?
- function – How can the required functionality be implemented? (The answer to this question allows implementors to evaluate one or more methods of achieving the functional requirements).
- technical – Will all the required technologies interwork?
- planning – What resources are required to achieve tasks with possibly unfamiliar technologies? The answer to this question improves estimates of resources and timescales.

The quality process for answering each type of question is different. Product development in the distribution theme was based on this method of incremental prototyping. The technique employed was to plan only the next step in detail, but to have analyzed systematically the functional options for the next three or four steps. This approach allows cross-company investment decisions to prioritise the functionality added at each increment and a development route to be agreed that avoids:

- replanning and discarding detail planning done 'n' months ago that is now out of date, and
- execution of a plan, the end-result of which is an out-of-date product.

The prototyping technique has the advantages of:

- a dynamic development route responsive to change.
- early deliverables that engender confidence, gain user "buy-in" and answer marketing, technical and functionality questions as well as providing users with interim solutions.

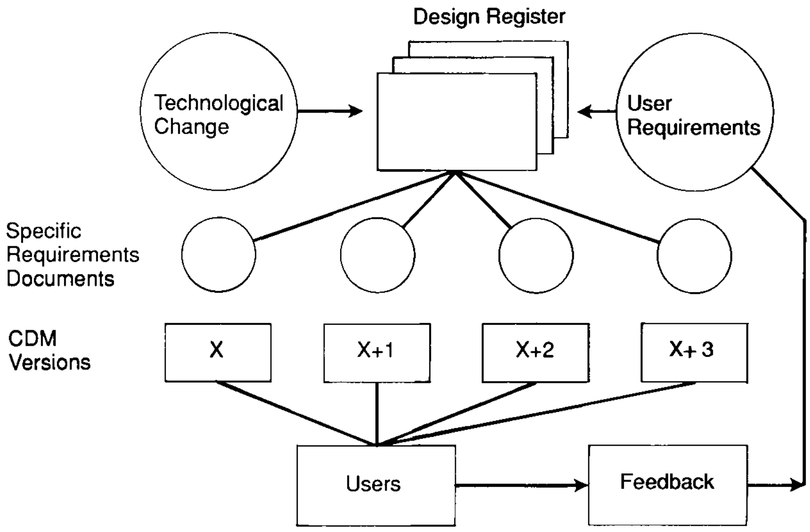


Fig. 2 The Incremental Development Process

Figure 2 summarises our approach to incremental evolution and capture of requirements. The 'Distribution Design Register' is fundamental to this approach.

4.1 The Distribution Design Register

The purpose of the Distribution Design Register (DDR) is to capture in a single document all the known functionality aspects of a distribution management system. It provides a basis for discussion between the development unit and other interested parties such as system integrators and requirements analysts. It presents the known functionality, not as an amorphous list but decomposes system characteristics into functional areas which are then further broken down into sets of options or levels of sophistication. This systematic approach provides a basis for the planning of an incremental series of developments. Readers of the DDR are encouraged to send additional requirements or proposed solutions to requirements to the author. There are currently over 200 entries.

The DDR decomposes the overall functionality of a distribution system into a set of functional areas:-

- Fileset Model.
- Machine Model.
- Routing Model.
- MMI.
- User Concurrency.
- State Model.
- Scheduling.
- Delivery Mechanisms
- Security
- Policy Options
- Backup and Restore (Archiving).
- RDA Functionality.
- Platform Porting Options.
- Software Copyright
- Operational Statistics
- Documentation
- Performance
- Conformance

Each register entry has a unique identifier. This identifier will remain fixed across different issues of the document and should be used as the primary index. Following the entry identifier is a short title for the entry. At the end of the entry there is a series of more complex source references. Several organizations may request the same function so there may be more than one source reference.

These references identify the organisation (typically by its acronym) that requested the function (the cross-reference provides lists by organisation), what was the source of the request, e.g. a workshop, a meeting or document, the date by which the function was requested to be available, the current target date (or version) by or in which the function will be provided (target means best guess) or the date (or version) the function is planned to be included and the functional area of this function.

The Cross-reference provides the ability to lookup requested functions by:-

- the organization requesting the function.
- the target or planned release the function is to be in.
- functional area.

Additional to the Design Register are a set of optional specific requirements documents. In some cases where individual organisations are dependent on an increment delivered by a certain date, a specific document is created to define this organisation's requirements. These will cross-relate to the requirements in the design register.

It is the strategy of the distribution theme to produce a product that evolves through successive incremental levels of functionality. The design register and these documents are used to prioritise the functionality for these incremental releases.

This approach is iterative in that customer satisfaction is achieved by the continual evolution of the next increment and then exposure to customers to gain feedback. Individual collaborations have served as a catalyst to this process.

5. The chosen design

The chosen design of ICL's "Distribution Manager" product follows the architecture described above and makes use of components from other architectural areas as well as re-using the Management Infrastructure. It can be split into three areas each with distinct platform characteristics; the Management Centre (MC), the Transfer Engine (TE) and the End System (ES).

5.1 Design objectives

The chosen design had a number of overall aims listed below, with comments:-

1 To *maximise investment* in development. Adherence to the ICL Quality Process minimised re-work.
2 To *reuse existing components* such as Ingres, CFT and CAM. 'Distribution Manager' creates standard CFT batches and actions CFT (via a CFT driver running on the transfer engine) to run them.
3 To ensure the Management centre is *X/OPEN conformant*. This will cut the cost of porting to new/or other equipment. Initially the management centre is UNIX based:
   - UNIX is becoming an industry standard
   - UNIX supports X/Open
   - UNIX runs on hardware covering a range of sizes and costs, providing flexibility in choice of machine.
4 To ensure that the other design components are *easily portable*. This reduces the cost of providing a multi-vendor solution and that of providing across the ICL range of platforms. As a consequence the customer can make use of his existing hardware.
5 To *minimise ongoing support costs*, the Management Centre will be generic, that is it will not make assumptions about object construction and relationships which are specific to the regime.

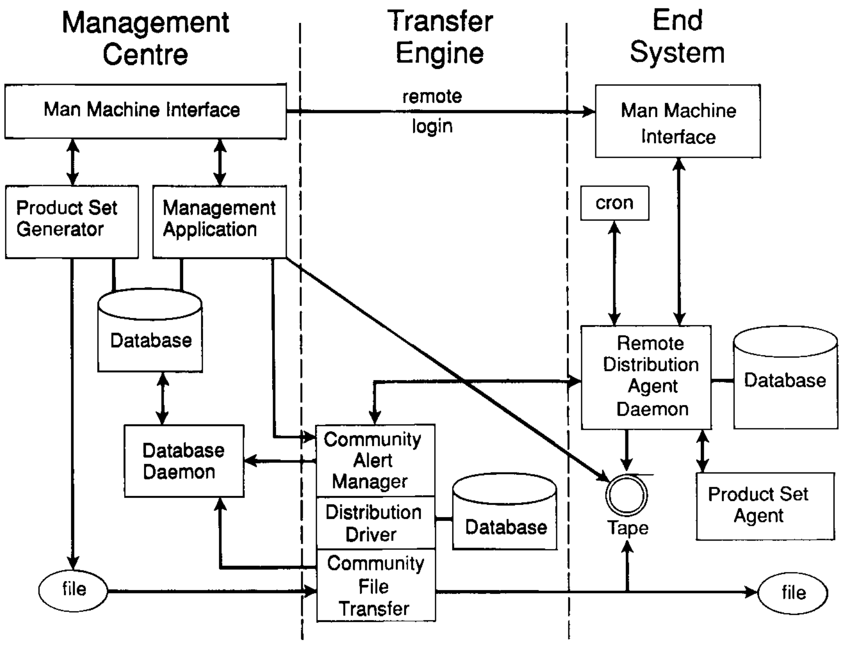


Fig. 3 Design of Distribution Manager

5.2 The design of Distribution Manager is illustrated in Figure 3.

5.2.1 Management Centre The Management Centre implements the architectural areas of Generator, Distribution Definition and Control and Distribution Administrative Datastore.

The simplest form of generator capability is provided directly by the Management Application (MA) component which allows a user to define the names of files making up a fileset. Product Set Generator (PSG) is a more complex generator which the user can use to define a UNIX software product along with control files and operations to allow the product to be 'installed'. In the case of UNIX products installation involves copying the files into PSA controlled filestore and activation which involves putting them into an end-system users filestore. Both of these generators use a common Man Machine Interface (MMI) which uses the Ingres/Forms product.

The MA component shares the Ingres/Forms MMI with the generators. Through the MMI the user can define and run schedules, print reports as well as make enquiries on the database and monitor the progress of running schedules.

The database used is Ingres and, along with the Database Daemon component, provides the architectural 'Distribution Administrative Datastore' capability. The Ingres database is updated directly by both MA and PSG and indirectly by the Transfer Engine through the Database daemon which is a persistent process, normally suspended, that receives messages from the Transfer Engine through the Management Infrastructure. Using the facilities of Ingres Net and Ingres Star in the future the database can become both distributed and therefore scalable and, if desired may be located remotely from the Distribution Definition and Control Application.

5.2.2 Transfer Engine 'Transfer Engine' capability is provided by the Distribution Driver component which has its own persistent database based on flat file technology. The Distribution Driver interfaces the Management Infrastructure components of Community File Transfer (CFT) and Community Alert Management (CAM) through their respective service interfaces. The Distribution Driver is written in 'C' and has been ported from VME to UNIX. Both of these support CFT and CAM and indeed Distribution Manager can manage a community of transfer engines from multiple vendors running UNIX and non-UNIX operating systems. Using more than one Transfer Engine allows Distribution Manager to be scaled to meet the customers requirement. Portability allows re-use of existing customer hardware investment.

Community File Transfer can control many different kinds of bulk file transfer mechanisms including FTAM, FTF, NFS and RFS. Thus replication and movement of filesets can be achieved to any end system supporting one of these mechanisms.

Community Alert Manager provides the control flow from the management Application to the Distribution Driver and the feedback from Distribution Driver to the Database Driver. MA's ESQL access to the database completes the control loop. CAM is also used to invoke operations on the Remote Distribution Agent daemon (RDA) and to receive responses from the RDA back to the Distribution Driver. If the Transfer Engine were to act just as a conduit for the messages from the RDA daemon it would be redundant. The Transfer Engine is more than this, in that it acts as a multiplexer-demultiplexer taking a single request from the Management Centre and fanning it out to many RDA daemons, much as X400 fans out a single mail item to all the addresses. It also carries out the reverse process collating together the many replies from the RDA daemons and sending only a few messages to the Database daemon. These two effects greatly reduce the bandwidth required between Management Centre and Transfer Engine and distribute the job of collation – a cpu-intensive task.

5.2.3 End-System An *end-system* is any machine in the network managed by the Management Centre. An end-system contains two components, namely the Remote Distribution Agent architectural component which maps directly to the RDA daemon and the Agent architectural process which maps to the Product Set Agent (PSA). Both the architecture and design prevent these components from existing on an end-system in which case only delivery of files is possible. For some customers, managed delivery of business files without the added benefit of software installation has sufficient benefit to justify the use of a managed distribution system.

The RDA daemon receives CAM messages requesting the invocation of operations from the Transfer Engine and invokes the PSA to carry out the operation. The result of the PSA operation is placed in a message and sent back to the Transfer Engine. The RDA daemon can also be invoked from a FMLI (Forms and Menu Language Interpreter) MMI on the end-system. Changes carried out through this MMI also update the Distribution Manager database. It is also possible to request the RDA daemon to carry out an operation at a pre-determined time in the future.

The interface between the RDA and the PSA is via "command line call"; that is to say it is as if the user had typed in the command at a terminal. This arrangement allows for the simple addition of more Agents in the future.

It is recognised that sometimes it is not sensible or reasonable to transfer many megabytes of files over a network. There is therefore a facility whereby a fileset may be copied to exchangeable media and delivered manually to the end-system.

5.3 Summary of Product Functionality

In summary, OSMC DM, (the Distribution Manager component of the ICL Open Systems Management Centre suite of products) includes functions to provide for the following features:

- Files are grouped into sets called filesets. Groups of files are treated as a single management unit.
- End-systems can be grouped together in groups called Machine Groups. Groups of end-systems are treated as a single management unit.
- A fileset may contain either ordinary or logical files. This allows, for example, products and their configuration files to be managed.
- Schedule entries that allow a fileset to be replicated, moved or that invoke an operation (eg. software installation) on all machines in a machine group, providing a powerful set of operations on high level management units.
- A schedule is made up of many schedule entries, each done in sequence. This allows the success of one operation to trigger the next.
- A Unix product can be put in a fileset and it can be moved, replicated and have end-system operations such as installation and activation performed on it. A Unix product may be managed on many end-systems from one management system reducing manpower requirements.

- All products, filesets, machines, machine groups and schedules are recorded in an Ingres database. A database of the hardware and software of an organisation is produced, automatically maintained and may be used to provide management information within that organisation.
- All fileset movements, replications and operations are recorded in an Ingres database. Thus an organisation can find out what software it is using and where. This information can be summarised in printed management reports and used for forward planning.
- All fileset movements, replications and operations can be monitored in real time, giving the user confidence of the operations taking place.
- Reports can be tailored to report on specific products or on specific groups of machines and used to print out information from the database. The customer can tailor reports to provide senior management with up-to-date information on installed hardware and software.
- More than one Transfer Engine may be used and the database may be distributed and is re-locatable. To meet the requirements of organisations of many different sizes, the product is scalable in more than one way, namely in respect of the size of filestore, the network bandwidth and database.

6 Conclusions

The team that developed Distribution Manager has an established and systematic method by which its product solutions may be incrementally developed. In the coming years technologies will change causing:-

- what is not possible now to become feasible.
- new requirements to arise.
- distributed systems to grow in size and complexity.

Distribution Management is one of the areas in which ICL has chosen to invest. It will continue to develop its distribution products incrementally to meet the needs of the 90's.

ICL is confident that the Distribution Management products it produces will be exploited in many different ways, that their functionality will be extended and that they will act as a catalyst to product integrators working on solutions to a wider set of problems.

Acknowledgements

INGRES and INGRES/FORMS are trademarks of Relational Technology Inc. UNIX is a registered trademark of UNIX Systems Laboratories, Inc. in the USA and other countries.

References

WHITE, J.T., Generation of configuration a collaborative venture, ICL Tech. J. Vol. 7 No. 4, pp. 732–740, 1991.
GALE, A.C., The Evolution within ICL of an architecture for Systems Management, ICL Tech. J. Vol. 7 No. 4, pp. 673–684, 1991.
PICKWORTH, I., Practical experience in the management of Retail business data flows in a distributed computing environment, ICL Tech. J. Vol. 7 No. 4, pp. 718–731, 1991.

Biographies

Phil Barthram

Phil Barthram is the Development manager at ICL's Systems Management Product Centre in Basingstoke. He graduated in Mechanical Engineering at the University of Southampton in 1975. Prior to his involvement with the IT industry he worked as a Research and Development engineer and as a mathematics teacher. He subsequently joined MoD at Bureau West as a software engineer eventually managing the development of their security enhancements to the VME operating system. Before joining ICL in 1989 he worked for a software house where he managed the development of a range of VME systems management products whose primary focus was in the areas of performance analysis, capacity planning and job scheduling. At ICL, prior to taking this current position, he was responsible for developing and progressing the strategy for the Distribution theme of Systems Management Product Centre.

Tim Howling

Tim Howling joined ICL in 1982 after taking a Joint Honours degree in Applied Physics and Chemistry from Durham University. He worked on various projects using ME29, DRS 20, System 25, VME and Pera rising to team leader of a collaboration project with Rediffusion Robots to produce a "Flexible Manufacturing Cell". In 1985 he moved to Concurrent Computer Corporation where he developed X.25 and OSI transport software and a new generation of network configuration tools. He then took responsibility for a team porting file transfer and remote login software to MSDOS PCs, SUN and APOLLO workstations, VAX VMS, MASSCOMP parallel Unix computers and Concurrent's own parallel realtime operating system. He returned to ICL in 1989, joining the Design team of SMPC, where he led the design of the ICL product – OSMC Distribution Manager. He is now the Principal Designer responsible for all SMPC Development.

Experience of Managing Data Flows in Distributed Computing in Retail Businesses

I. Pickworth
ICL Retail Systems, Bracknell, UK

Abstract

Considerable experience with some of the largest retailers worldwide has led to a specialised ICL product line to manage the flow of data within a retail organisation. The products are based on ICL Systems Management products, but specifically aligned to the retail market. This paper first gives a brief background on retail data processing, and then shows how the products have evolved over the past decade. It ends with a view of the future direction the products might take.

1 Introduction

Starting in the early 1980's, retailers throughout the world invested substantial capital in electronic point-of-sale (EPoS) equipment in their stores. Initially, the driving force was the reduction in operating cost that could be brought about by automating, and thus de-skilling, activity at the point of sale. The reduction in operating costs continued to be a very strong driving force in retail organisations throughout the 1980's.

However, as most retailers began to benefit from EPoS equipment, the commercial advantage of installing such equipment began to be eroded. This caused a search for further advantages from the EPoS equipment by using the data generated in the point of sale device to run the retail operation more efficiently.

Retailers were quick to see that an EPoS terminal can gather very accurate data about one of their most valuable assets, ie their customers. This realisation led to ever increasing and more focused use of data gathered about transactions made on EPoS devices.

In 1984 ICL formed the Retail Business Centre (RBC), specifically to develop business in the retail vertical market. Recognising from the outset the value of data collected at point of sale, the RBC launched a development programme to allow retailers to make use of it. Since then, the development programme has evolved and widened to meet the needs not only of large supermarket chains, but also those of businesses across all retail segments.

This paper gives background to the data processing environments in which most retailers operate, and describes the evolution of a product line (known as Retail Systems Management) that has grown out of practical experience of working with ICL Retail Systems customers.

2 The Retail Model

A quick walk around any town centre will show that retailing is a very rich and multi-faceted business segment. However, behind the apparent variety of goods on display to the public, retailers by and large follow a very similar formula for running their business. This formula has been well tested, and is constantly changed by retailers operating in an extremely competitive environment.

The basis of any retail operation is the "merchandising cycle". The basic idea is that retailers follow an endless loop of interconnected processes whereby goods are bought, stored, transported and sold as efficiently as possible. The cycle embraces the whole retail business, from original marketing decisions to the precise placement of merchandise on the shelves of a given shop.

Figure 1 shows the most common steps in a merchandising cycle. Although the process is continuous, most of its actions are directly affected by decisions made at head office, so a description of the process should start from there:

In the head office marketing departments, decisions are made about what product mixes to offer, in which stores and at which prices. These decisions require detailed analysis of a large quantity of data, particularly the past performance of the retailer's stores.

Head Office decisions are quickly converted into specific buying actions, in the form of instructions to suppliers, distribution centres and stores. These instructions result in the movement of goods from supplier to stores, either via the retailer's own distribution system, or in the form of direct delivery to store.

Because the store is the place at which the retailer and the customers meet, it *is* the retailer from the customers point of view. Thus, along with information about goods and prices, the store also needs information to plan its staffing levels, adjust its layout, accept a multitude of payment methods etc. Since most retailers look for growth, it is seldom cost-effective to have highly trained staff in every store to manage all aspects of the business. As a result

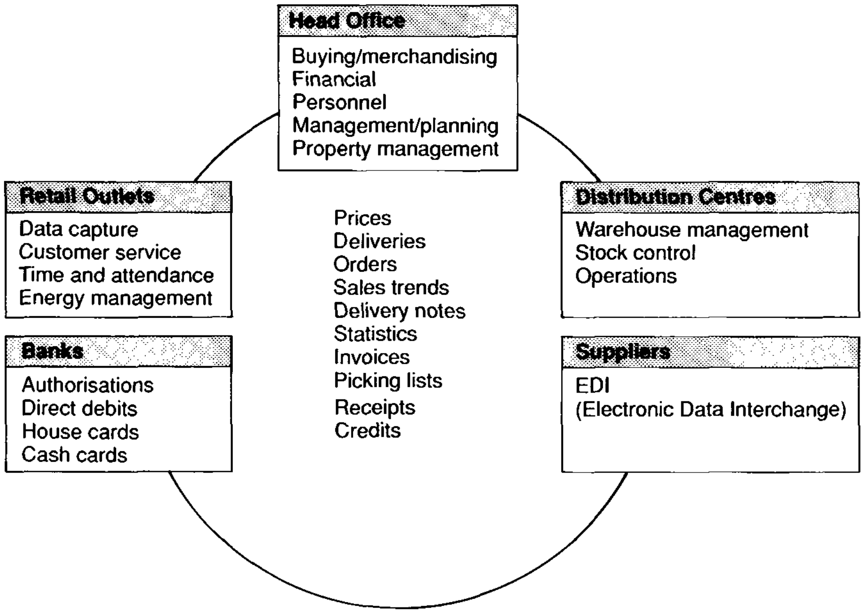


Fig. 1 The Merchandising cycle

stores tend to be managed by "remote control" through very rigorous definition of policy and operating procedures.

The activity in the stores pushes the merchandising cycle, by providing the data to drive the business. Close track is kept of cash takings and stock levels in each store, and a constant process of fine tuning is undertaken to attempt to maximise profitability, eg by analysing sales volume per square foot of floorspace, or by tracking the contribution of product lines etc.

The movement of data in support of the merchandising cycle has to be both timely and accurate. Timeliness relates to the correct time at which a data item has the most value eg. the retailer must know how much cash is in the stores at the deadline for depositing money on the overnight financial markets. Accuracy is also time-related eg. data on a store's takings is not accurate until a store accounting period has been balanced and reconciled.

Two points about the merchandising cycle should be stressed:

- Management of the cycle will directly determine the profitability of the retailer.
- The merchandising cycle operates almost in real time, which is very different from other businesses. Events in a store can affect orders on suppliers in as short a time as a few hours, and seldom in more than a few days.

3 The Data Processing Environment

Data Processing can support the management of the merchandising cycle in very many ways which vary a great deal with the management style of the retailer. That in turn is closely related to the segment in which the retailer operates, though there are no hard and fast rules.

Where stores are very large, not very numerous, and stock a large variety of goods, in-store management tends to have a higher degree of control over day to day operation. Good examples are the French hypermarkets (Pisigot, 1989) where store managers effectively operate as managers of a local business, and have a great deal of say in product mix, pricing and presentation. This leads to a requirement for much in-store data processing, which will need to cope with all the facets of an independent business.

At the other extreme, where stores are small but very numerous, the retailer will want to keep tight central control on the image of the chain as a whole, and will seek to minimise the cost of the operation per store. Good examples here are fashion chains in the UK, USA and Canada. In-store data processing is typically limited to capture of EPoS data.

In between these two extremes, one can find almost any split of data processing between store and head office. Since, to reduce staff training and administration cost, and to present a consistent image, all stores in a given retail chain will be more or less exact replicas of one another, it follows that every data processing decision involves cost that is multiplied by the number of stores. This puts extreme pressure on the price most retailers would be willing to pay for any in-store equipment (or services) unless there is a proven link to higher sales, or reduced cost per sale. Most retailers prefer to spend money on the display of goods to entice people into the store.

Therefore, the following are vital characteristics of any products seeking to manage data flows in a retail business:

- Retailers are expert at dealing with suppliers, and understand that several suppliers are always better than one. All retailers therefore seek a multi-vendor capability in their data processing departments, to allow them to influence the price of various parts of the system.
- Although this appears to match exactly the Open Systems strategy adopted by ICL, retailers apply openness in a much broader sense than OSI or X/OPEN. In particular, with networks they will be seeking openness at the lowest possible cost. Very often, this will mean using asynchronous protocols over dial-up telephone connections.
- Following a strict policy of reducing in-store processing costs leads to the need for increased processing power at head office. This area of the market is dominated by small to medium-sized mainframes from IBM. This implies that IBM interworking and SNA network conformance are very common requirements amongst retailers.

4 Typical Data Flow Management

ICL Retail Systems has found a common trend amongst retailers in the way the flow of business data around their organisations is managed. There are typically four types of data flow, and retailers attach a very different value to each of the types:

4.1 Bulk Data transfer

Typically at end of trading day by moving sales data and price updates to and from a store over a telephone call. This is usually seen as a necessity when EPoS equipment is in use, but is not perceived as a very high value activity.

4.2 File management

This is an extension of bulk data transfer, but gives the retailer more freedom to be flexible during a trading day (eg by altering prices in response to competition). It will usually lead the retailer to establish some form of on-line connections to the stores, and is perceived as having several benefits in improving the responsiveness of the operation.

4.3 Operations management of distributed data processing equipment

This can be shown to reduce the need for trained staff in the stores, and is thus seen as having some benefit in reducing operational costs. However, a significant number of retailers could view this as a vendor charging them to solve a problem the vendor has introduced, and therefore very careful thought has to be applied to positioning products in this arena.

4.4 Message management

This is the processing of information as the events occur, and is universally perceived as having high value. Applications such as payment authorisation have been in the lead, with the management of the logistical supply chain (movement towards "just in time" methods) now taking more prominence.

These four types of data flow are not mutually exclusive, and they may all be found within a single retail organisation.

It must be remembered, however, that management of data flows and systems management is a means towards the end of a more efficient merchandising cycle. The four areas above must be linked with applications that show benefit to the business.

5 Evolution of Retail Systems Management product

The development of products that managed the movement of files between EPoS equipment and Head Office started in the early 1980's by following pragmatically the requirements of the Retail Business Centre's customers. These customers fell mainly into two categories:

- Specialist retailers (mainly apparel and home improvement).
- Large food retailers.

Specialist retailers had a need for basic dial-up data transfer, and products were developed on both the DRS20 (Retail 67) and the System 25 (Retail 47), that interworked with the ICL EPoS products on offer (9516, 9518, DRS20 EPoS controllers and System 25 EPoS controllers). These products were completely self-contained, in that they made no use of other ICL management products, largely because they preceded the developments in the community management programme outlined below.

Food retailers by and large tend to use IBM mainframes at head office, and in the 1980's a large proportion were actively installing System Network Architecture (SNA) products to manage their networks. Those that had decided on SNA networks demanded conformance to the IBM products from all their computer suppliers. Working with some large food retailers ICL therefore developed two software products that ran on IBM mainframes:

- *File Transfer Facility* (FTF) which is an implementation of the Network Independent File Transfer Protocol (NIFTP).
- *Community File Transfer* (CFT) which is a file transfer and task scheduling product designed to coordinate many different data transfers to and from many locations.

At the time the alternative to developing these products was to develop IBM compatible file transfer protocols in the EPoS systems. ICL decided against this course because:

- The IBM standards are not open, since all details of the interworking protocols are not made generally available. This would imply a significant risk for any implementation that attempted to copy them.
- There were several file transfer methods in common use amongst IBM users, making difficult a choice of one to copy.
- Most significantly the ICL EPoS system would be limited to the functionality on offer from the IBM host product, thus eliminating the opportunity to better IBM and other EPoS suppliers in this highly important area.

The development of FTF and CFT on IBM coincided with the start of the community management programme in ICL. FTF was widely available on

ICL hardware and the design of CFT was made to be portable, so that it could be reimplemented on other host hardware. The Retail Business Centre also enhanced the user interfaces to CFT by developing *Retail File Manager* (RFM), which allowed users to specify activities to be applied to logical branch groups (ie. numbers of stores related in some way), in a way that naturally met the retail requirement for file management.

Both Retail Systems and the community management programme continued to develop products during the late 1980's until by the end of the decade a large assortment of product was on offer from ICL. The sheer number of products made the benefits provided by each product difficult to describe and sell, even though the products themselves performed very well.

To remedy this situation the community management programme reorganised its products and activities around a series of management themes (Gale, 1991), such as statistics, status, configuration etc. This served to align the programme (now known as the *Open Systems Management Centre* (OSMC)), with the emerging standards for systems management. Although this was clearly a positive step, it did not go far enough to solve the problem of selling systems management specifically to retailers.

Retail Systems therefore closely examined its experience in this area and created a product marketing framework to meet the following requirements:

- Product benefits must be described in terms that retailers can easily relate to.
- Numerous and confusing components must be replaced with a clear structure.
- The product range must be summarisable on a single presentation slide (to give an easy-to-remember message).
- The product range must clearly support retail systems major business, ie. selling EPoS equipment.
- Retailers must be offered options at several cost levels, with a clear vision of potential growth from their entry level.

The approach taken was to group product options around the four types of data flows described in section 4. What emerged was a product portfolio consisting of 4 levels, with each level building on the previous level by adding features and functions. The four RSM levels are:

5.1 RSM Level 1: Dial-up data transfer

A grouping of products that provide the functions needed to manage a dial-up data transfer cycle to a large number of stores. Main functions provided are:

- two-way data transfer, provided over either asynchronous or bisynchronous protocols, and a variety of modem types.

- the ability to preset a schedule of dial-up activity to allow unattended overnight operation.
- the ability to recover failed transfers easily, either in a follow-on session or in the following day's cycle. Also, the specification of the number of retries for any given connection attempt.
- the ability to group stores logically for different types of data transfer, with each store being allowed to belong to any number of groups.
- the ability to create and edit a list of file transfers, which can subsequently be applied to a logical group of stores (like a macro).
- reports and logs to show exactly what has happened, and to show exceptions.

5.2 RSM Level 2: File management

This level supports the more sophisticated requirements of retailers that have moved beyond the simple cycle of overnight data retrieval. The main functions required at this level include all those needed at Level 1, plus:

- an extended diary manager, with very flexible scheduling capabilities.
- the ability to start ad-hoc groups of transfers in response to business need, with on-line creation and editing facilities to support ad-hoc activity.
- management of local and remote job execution, thus allowing distributed application functions to be coordinated with the movement of data.
- on-line recovery, giving options for the system to take immediate action on failure.
- an Applications Programming Interface, to allow retail applications to exploit RSM products.
- support for mixed networks, by allowing the definition of work to be separated from the file transfer protocols. This allows mixtures of network types and protocols, and makes it easier to change network characteristics in line with business needs (eg. in a transitions from a dial-up to a permanent network).

5.3 RSM Level 3: Operations management

This level is very closely aligned to the management themes of the OSMC (status, statistics etc). It is the least retail specific of the four levels, because it addresses common problems faced by users of distributed computers. Main functions needed include all those at level 2, plus:

- support for Remote operation, to allow central staff to operate store equipment.
- software distribution and remote installation of software.
- management of "alerts", by routing to the correct point in the customer's organisation.
- problem management (ie. the logging, tracking and resolution of problems), both for alerts received automatically, and for problems reported verbally.

- statistics gathering for analysis of performance.
- monitoring of systems status to allow pro-active action in the event of failure.

5.4 RSM Level 4: Message management

This level deals with the management of messages that originate from retail transactions, such as credit authorisation, stock in transit enquiries, employee sales transactions (both for employee discount and sales commission applications) and many more. The management required is of one or more of following types:

- routing based on message content (eg. by credit card type, transaction amount etc).
- routing based on time of day (to minimise cost of using public networks).
- routing based on urgency (eg. debit authorisation takes priority over sales history data).
- buffering, to get the best out of network charges (eg. by buffering sales transactions to fully use the packet size available).

There are also several functions that are retail specific and are provided for in RSM Level 4 products.

- Local authorisation of credit based on a combination of the following checks:
  - authorise all credit under a given amount (known as the floor limit).
  - authorise up to the nth use of a credit card, referring all subsequent uses to the card issuer for authorisation (known as a velocity check). Typically used together with floor limit checks.
  - check each card against a known list of invalid cards provided by the card issuers (known as negative card checking against a hot card file).
  - refer all authorisation requests to the card issuer (known as positive credit checking).
- In all cases a response is routed back to the EPoS terminal within a specific time. Systems have to provide tools to allow response times to meet very strict performance criteria.
- Logging of the sales audit trail, providing for recovery after loss of continuity for whatever reason. A vital requirement of this function is that no sales audit data may ever be lost.
- Distribution of non urgent data files on a record by record basis during off peak times, or after all important store activity is complete. This will allow, for example, a new programme update to be transmitted at off-peak times over a period of several days.

The four-level approach allows the benefits of the generic capabilities of the various products to be realised, and aligned with business benefit. It clearly demonstrates the openness of the approach, in that the same functions can be provided on a whole range of central and in-store systems. Since each level adds function to the previous level, the requirement for showing potential growth is met, as well as the need to offer several options at different costs to the user. The increase in function can easily be related to an increase in price.

Initially, the existing products (already numbering more than 50 on head office systems alone!) were packaged only in a marketing context. However, in 1990 another significant event occurred, the launch of DRS6000. For the first time, Retail Systems could offer an acceptable alternative to IBM mainframe processing, which allowed a change of development priorities. The first RSM option on UNIX V.4, RSM Level 2, was generally released in December 1990. During 1991, RSM UNIX options will be completed, with levels 1 to 4 available on the UNIX V.4 platform.

Further development of the approach has lead to the introduction of the concept of an RSM *in-store gateway*. The RSM in-store gateway is a tested grouping of all those functions needed to support a given level of RSM functionality in the store. This was found to be required since the integration and testing of the correct in-store communications and systems management components was being left to the customer (or the ICL customer support team), and thus adding significantly to the cost of sale. Thus:

- A level 1 gateway supports asynchronous, and bisynchronous file transfer protocols.
- A level 2 gateway supports the common "permanent" network protocols (ie. SNA and OSI), and a file transfer protocol such as FTF.
- A level 3 gateway supports the systems management functions of Alert generation, Software distribution and remote installation, and remote operation.
- A level 4 gateway will support various access protocols to on-line networks, and buffering mechanisms to allow message based data flows between distributed applications.

As far as is possible, the RSM product is developed on generally available ICL products, eg CFT, Community Alert Manager etc. However, the emphasis on low cost networking has resulted in additional developments, such as a Retail Connection Manager (RCM) for RSM Level 1. This facility has added the vitally important asynchronous and bisynchronous dial-up capabilities to the standard CFT product.

6 Experience of Retail Systems Management in use

The benefits that a Retailer can get from using RSM are very strongly determined by the segment in which that Retailer operates. The RSM product line recognises this by grouping the options into levels as described above. Each level is particularly relevant in a certain retail segment, as follows:

6.1 RSM Level 1: Dial-up data transfer

This is directly applicable to specialist retailers such as eg. jewellers, who have many small stores, and stock a specialised range of goods. Sales volumes are usually relatively low, and stores do not carry extensive ranges of stock. In-store equipment would typically be one EPoS terminal.

In the USA, where telephone communication has traditionally been both cheap and easily available, this method of managing data transfer is also very common in the small to medium sized supermarkets, with store controllers managing typically up to 30 EPoS terminals.

6.2 RSM Level 2: File Management

The amount of data generated in a medium sized supermarket or departmental store usually requires the ability to manage file transfers on-line. The in-store equipment would typically still be limited to the system running the EPoS equipment, although this would now be a server of some form, controlling a number of EPoS terminals.

6.3 RSM Level 3: Operations Management

Any large stores (Hypermarkets, large supermarkets and large departmental stores) are now likely to install increasing amounts of IT equipment, and to run systems ranging from stock control to customer service applications on them. Retailers need automatic alerting, software distribution and version control, remote operation and problem tracking systems to keep down the operational costs of such systems.

6.4 RSM Level 4: Message Management

The application of RSM style message management techniques in retail has tended to follow a very pragmatic approach, being strongly influenced by the data communications culture in each country. Thus, where public networks have not been widely available, or have not been cost effective (taking the cost per store into account), message processing is limited to credit authorisation routing from store to card issuer. In some countries (most notably in Canada) very cost effective connections to public networks have been provided. These have encouraged the implementation of systems using many of the RSM level four functions described in section 5.4. Retailers that have taken this lead have received significant pay back in the reduction of operating costs, resulting from both reduced charges for credit transactions and more efficiency in the management of store restocking, by holding fewer goods in each store without running out of stock.

7 Examples of RSM products in use

Retail Systems Management products are currently used by more than 100 retail customers worldwide. The following selection of customers shows RSM products in operation:

- The UK toy specialist, *Early Learning Centre*, have been users of Retail 47 on System 25 for several years. They have ICL 9518 point of sale terminals in every store, which have been programmed in COBOL to meet their specification. When the stores close, an overnight collection/distribution cycle collects the day's item movements and data on the store takings, and distributes the price/item changes for the next day's trading. Communication is achieved over dial-up bisynchronous 2780 bulk transfer protocols.
- *ASDA*, well known UK food retailers, use RSM Level 3 on an IBM mainframe to control the population of DRS20 in-store systems running the General Merchandising System (GMS) developed by Retail Systems. GMS controls all the EPoS terminals in a supermarket, and provides a comprehensive set of retail options which ASDA exploit. As well as transferring data to and from stores (store orders, price changes etc), the DRS20 and GMS software is distributed and remotely installed without the need for intervention by staff at the stores, and "alerts" are routed to the help desk facilities supported by the central IBM mainframe.
- *Dylex*, the largest fashion retailer in Canada, use RSM level 4 facilities on a number of System 25's to manage the on line credit authorisation and sales data capture from about 1400 stores across Canada, each containing between 1 and 4 ICL 9518 point of sale terminals. The Canadian Datapac network supplies a very cheap form of on-line connection (known as 3201, which is an asynchronous connection to an X.25 network at 1200 bits per second); this makes running such a facility very cost effective for Dylex.

8 Strategic Directions

The RSM programme faces three main challenges:

- providing close links with the all important retail applications that must process the data being generated within retail organisations.
- exploiting new network technologies in such a way as to protect the retailers investment in skills and applications, but quickly enough to provide competitive advantage.
- exploiting international standards (such as those for systems management and electronic data transfer) as these become established in the retail market.

An example of a retail application that will need to closely exploit RSM facilities is a Strategic Marketing Information System of the type described by Dobbyn and Cheesman (1990). Such a system brings together data from many diverse sources and uses knowledge based techniques to synthesize market conditions. More mundane, but no less important, are applications that control the supply chain, ie the movement of goods from suppliers, through a retailer's warehouses to the stores.

The most prominent new network technology on the horizon is ISDN (Orange, 1991). This has the potential to revolutionize retailers approach to on-line message management, if it can be integrated into the data processing environment without needing wholesale changes in applications and procedures. RSM provides a good framework for achieving this in the way that it seperates the network details from the definition of data movement.

International standards that relate to Electronic Data Interchange (EDI) are now taking shape and will allow more generic products to be developed for the larger market that will emerge once the standards are accepted. Standards for Electronic Funds Transfer (EFT) are very different from country to country, making the development of generic facilities very difficult. However, this is an important topic in retail and RSM products will have to follow the key trends to keep up with the market.

9 Conclusions

Over the last decade retail businesses have won enormous benefits from the automation of the check out procedure at point of sale.

Processing the information collected here and passing it rapidly back to head office or to the source of supply has given retailers an ability to adjust prices and stocks to meet changes in demand with unprecedented rapidity.

To be competitive or even to stay in business at all, retailers, now commonly exercising control from mainframes at head office, must have an utterly dependable network of communications channels to both depots and outside suppliers, as well as links to national banks to verify the credit status of customers.

All this has created a demand for integrated schemes that will manage the entire system including both communications links and the installed terminals and computers – regardless of supplier.

ICL's portfolio of Retail Systems Management products enables retailers to manage their flows of data so as to strike the optimum balance, for a given concern, between speed of transmission and response on the one hand and cost on the other.

By conforming to industry standards, ICL RSM products can manage networks linking hardware and software from a combination of sources.

Acknowledgements

- IBM and SNA are registered trademarks of the IBM Corporation Inc.
- UNIX is a registered trademark of UNIX Systems Laboratories, Inc. in the USA and other countries.
- X/OPEN is a trademark of the X/OPEN Company Limited.

References

DOBBYN, C. and CHEESMAN, J. SMIS-A Knowledge-Based Interface to Marketing Data, ICL Tech. J. Vol. 7 No. 1, pp. 66–81, 1990.
GALE, A.C. The Evolution within ICL of an architecture for System Management, ICL Tech. J. Vol. 7 No. 4, pp. 673–684, 1991.
ORANGE, M. Introduction to the Technical Characteristics of ISDN, ICL Tech. J. Vol. 7 No. 3, p. 451, 1991.
PISIGOT, Y. La Solution ICL chez Carrefour à Orleans, ICL Tech. J. Vol. 6 No. 3, pp. 451–467, 1989.

Biography

Ian Pickworth

Ian Pickworth graduated from the University of the Witwatersrand in Johannesburg in 1976, with a BSc in Computer Science. He joined ICL Dataskil in 1977, and worked on Transaction Processing systems development, becoming a consultant in TP systems design. In 1987 he joined ICL Retail Business Centre as a retail systems consultant. He currently manages the Retail Systems Management Business Unit within Retail Product Operations in Bracknell, UK.

Generation of Configurations – A Collaborative Venture

Jim White
ICL Systems Management Product Centre, Basingstoke, UK

Abstract

Configuring networked, distributed systems and the applications that run on them can be an error-prone task requiring significant levels of skill. The problems associated with this can deter the growth of Information System usage in an enterprise and/or reduce their capacity to change.

The Generation aspects of ICL's Systems Management Architecture are specifically aimed at these problems. This paper describes how ICL developed prototypes of its approach in collaboration with one of its large customers. An overview of the approach is given with some of the justifications for the particular choices made.

1 Introduction

The extent of the Systems Management problem posed by distributed systems has been appreciated by ICL and the other suppliers for many years. There have been many estimates of the amount of work, the complexity and the diversity of skills required to realise management of a distributed set of system resources. One of the most meaningful ways of quantifying the task is to compare the programme to the development of a sophisticated operating system like VME. The estimates, even with such a metric, vary significantly, but all are daunting.

It was the appreciation of the size of the programme of work that prompted Systems Management Product Centre to investigate collaborative ventures. Potentially, collaborations reduce cost, supplement and complement our skills and allow the ICL portfolio of products to be enhanced more rapidly. Obviously, in the internationally competitive climate in which ICL operates, rate of market penetration is a critical factor in establishing a presence in this potentially lucrative new area.

The SM collaboration over configuration generation clearly illustrates the benefits of such ventures. Collaboration with Inland Revenue has been going on since 1988, and has produced a valuable prototype, code named V250, which is regularly used to configure certain components in the Inland Revenue network; the experience gained has advanced ICL's developments in the whole area of configuration generation.

2 Background

The generation of configuration parameters for the components of a network has been a central aspect of ICL's Systems Management since the beginning of the Programme in 1983. By 1986 that aspect of the work had progressed from the configuration of a DRS20 based electronic mail service to an ESPRIT funded research project. Many valuable lessons learnt from the mail work have been incorporated in the architecture and in all subsequent work. The work done under the ESPRIT project attracted the interest of the Inland Revenue whose operational experience had begun to reveal the considerable costs associated with generating configuration parameters for large networks.

Corporate networks are large, dynamic and complex entities intimately associated with the commercial effectiveness and efficiency of their users' businesses. Technological change and business growth mean that generation is dynamic calling for frequent reconfiguration, while the size and complexity of corporate networks increase the skill required to perform reconfigurations.

Because the network is so important to business efficiency, any loss in operational effectiveness resulting from erroneous configuration can be expensive. Yet, existing configuration processes predominantly require the manual generation of obscure parameters; this is highly prone to errors.

Thus network configuration is expensive because:-

- corporate networks are dynamic and require frequent re-configuration,
- configuration requires particular expertise involving skills in great demand which are therefore expensive,
- existing configuration systems are essentially manual and so prone to error, and
- because errors can result in the loss of operational effectiveness they can be very expensive (for example the loss of the IR network for 8 hours results in lost time equivalent to a cost of £8 million for salaries alone).

The recognition of these costs led IR to believe that network configuration – not technology or business need – would ultimately constrain the size of network that could be operated economically.

So there were two collaborators – IR and ICL. Successful collaboration demands not only that both parties need to participate fully, but also that both are willing and capable of doing so. In the case of Inland Revenue, the need was to remove a constraint on "business" growth. ICL's interest in a successful outcome was to develop a unique product – one that solved a problem encountered not only by many other customers but whose solution can also increase market share by attracting new customers.

Inland Revenue's contribution was their expert understanding of the problem; this in turn enabled requirements to be clarified and solutions validated. ICL's contribution has been the architectural solution. Both organisations contributed substantial development resources and specialist skills.

3 Architecture

Within the quality process model of systems management (Gale, 1991), generation appears in the *introduction and deployment section* between *change* and *distribution*. The rationale behind this placement is that network components are re-configured as a result of some change – new requirements to be satisfied or problem removal, for example. The resulting files of configuration parameters are delivered to, and installed by, distribution on the end-systems affected. (Barthram and Howling, 1991).

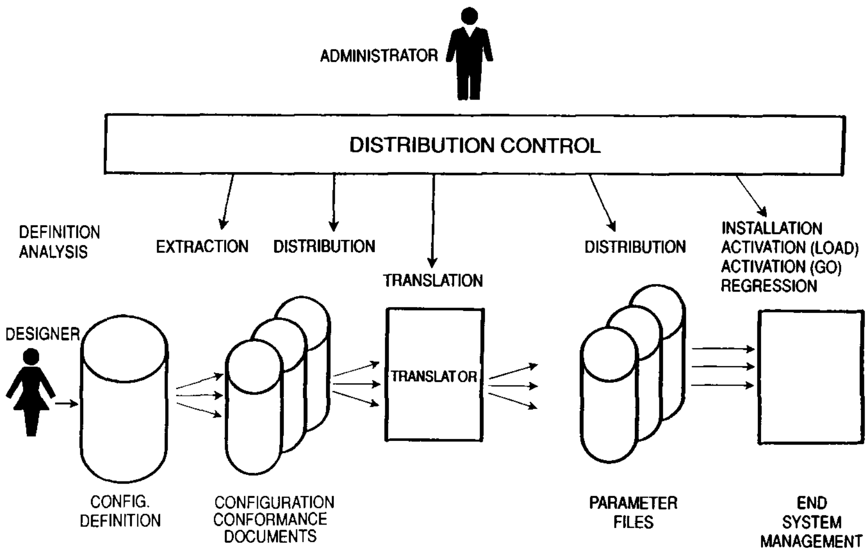


Fig. 1 Community Generation Architecture

The architecture of generation itself, is illustrated in Figure 1. The generation architecture contains 7 processes; each is considered in turn.

3.1 Data Capture

Data is captured by the configuration designer who translates the service requirements of the network users into the design of the network. A significant influence on the design of data capture is the sheer quantity of data to be collected.

Consider an organisation of 20 remote office buildings interconnected by an X25 service. Each office has 10 floors each containing a LAN. Each LAN contains 3 UNIX-based servers and 20 PC workstations providing access to any one of 6 corporate mainframes. This fairly typical organisation contains 6 corporate mainframes, 600 UNIX servers and 4000 PC's. If, for the purpose of illustration, we assume each component requires 20 items of configuration information, we see a total requirement for this network of over 92,000 items of information, a daunting challenge for data capture.

Use of high resolution graphics allows the designer to lay out the design of a network in the way that he/she visualises it; this capability is enhanced by using graphical techniques to display the various levels of hierarchical decomposition in the network.

A significant reduction in the amount of data that has to be entered can be achieved by recognising that many features of a network are replicas of some other part; by providing data replication facilities pieces of design can be copied easily thereby reducing detailed input.

Automatic deduction of information from that already provided can also reduce the amount entered during data capture (see para 3.3 on Analysis)

The ability provided for users to indicate sub-systems that have similar configurations so that the majority of data can be inherited automatically and then edited to give local variants, reduces the amount to be captured.

3.2 Data Storage

Once captured, the data needs storing for subsequent processing and retrieval. Due to the wide diversity of components in a network coupled with the similarity of the functions a management system must apply to those components, it is advantageous to adopt an *object-oriented approach* and an *object-oriented database* (OODB). The OODB is the subject of an internal ICL collaboration (called Raleigh) between ICL's Data Dictionary group and Systems Management Product Centre to produce a Corporate OODB. Raleigh is described elsewhere in this journal (Kay and Rivett, 1991) and is another example of the benefits of collaboration.

3.3 Analysis and Validation

Validation significantly improves the value of the information stored in the database, remember the error-prone manual system and the cost of disabling parts of the network. So if the integrity of the data can be established and error-free parameters created, a valuable facility will have been produced for network users.

By analysing the stored definition of the network it is possible to detect inconsistencies in that definition and to fill in many of the detailed linkages which make the network work in practice.

Essentially, analysis and validation are achieved by the application of rule-based logic to the relations portrayed in the database.

3.4 Extraction

Once a configuration has been validated configuration design is complete. It is then the task of the configuration administrator to implement the approved configuration. This delicate task requires that updated parameter files be delivered to each end-system affected and that each of those systems adopts the new configuration simultaneously. This is achieved through the control of the distribution system. Thus the remaining phases of configuration generation are controlled by the configuration administrator using the distribution facilities of system management.

As the name implies, *extraction* is the process of deriving configuration data for each end-system from the database. The database contains a global or 'birds eye view' of the network – it sees every component in relation to every other component. In order to configure each end-system individually, it is necessary to extract the singular or 'worms eye' view of the network as perceived from that end-system. (This view contains only that information required by that specific system to perform its role in the overall system design)

Unfortunately, there is no established standard for configuration parameters. Over the years each component designer has selected parameter formats for the convenience of the specific component in operation. Yet from the earliest endeavours in generation ICL learnt the importance of isolating the extraction process from the variation between parameter files. Extraction is a complex process in itself and, would become unwieldy if it were not isolated from the numerous, diverse and varying formats of parameter files.

To achieve the necessary isolation ICL introduced the concept of a file of *generic configuration parameters*. This file is based on the principle of providing an end-system and all its associated components with all information necessary for it to be configured. It provides the missing standard for a parameter file. The file is called a Configuration Conformance Document, or CCD for short. ICL is encouraging the appropriate standards organisations to introduce such a concept for the benefit of the entire networking community.

Extraction then, is the process of deriving from the validated database a CCD for each end-system affected by a reconfiguration.

3.5 *Translation*

If there were an established standard for a CCD, then end systems would be designed to be configured directly from CCDs. However to become meaningful to an end system, CCDs need to be converted to the file format of that system. This conversion from a CCD to a parameter file is known as *translation*.

To become configurable from CCDs, each end system requires a *translator*. This is the cost of not having an established standard for configuration parameter files and of achieving isolation at the extraction phase. At least this approach limits any change in the requirement for the configuration of an end-system to its own translator and prevents destabilisation of the extraction process.

In general, a change to the network, arising from either the need to resolve an operational problem or meet a new requirement, will affect the configuration of a number of end-systems. Therefore, there could be many CCDs and translation into parameter files could represent a significant computational load. In the interests of processor efficiency it may be prudent to decentralise the translation to a number of processors rather than restrict the availability of a central processor during translation. The Distribution System offers the option of centralised or distributed translation. There is an intermediate phase between extraction and translation, during which CCDs are distributed to the remote processors for translation.

The progress of all the configuration phases from extraction onwards is monitored by the Distribution System. When the system is satisfied that a translation has been completed successfully, parameter files can be delivered to the appropriate end-system and installed.

3.6 *Activation*

Only when all the parameter files have been successfully installed on their end-systems can *activation* be considered. Activation is the process by which an end-system and its components "adopt" a new parameter file. To ensure continued communication it is essential that all end-systems adopt the new parameter files at the same moment.

If the newly configured system is found not to provide the desired result, then the Configuration Administrator has the option of *regressing* to the previous parameter files. Regression is basically the reverse of activation and, of course, it is just as important that it occurs simultaneously on all end-systems.

3.7 Tidying

Once the Configuration Administrator is satisfied that the new configuration is working correctly and that there is no longer a need to retain the previous version of the parameter files, then they can be deleted by issuing a *tidy* instruction to the Distribution System.

4 The Achievements of Collaboration

The configuration generation collaboration has had two phases:

- the development of a working, bespoke prototype known as V250 and
- the development of a generic solution

An experimental version of the generation software known as V250 was first tried out in 1990 at Inland Revenue. The first four stages of the process were "bespoke" in V250 inasmuch as (for speed and convenience) they were based on existing IR data files. V250 was designed to configure DRS300, MCU1s(Network Gateways), VME-hosted TP services and the VME catalogue. Choice of these network components reflected the interests of the two collaborators:

- they constituted a significant portion of the IR network
- being quite different components a terminal, a gateway, a service or a mainframe – they represented a rigorous test for the configuration architecture.

Work started on V250 in June 1988 and eventually involved over 60 staff from Inland Revenue, the ICL (UK) IR Account Team, the sub-contractor Northern Computing Services, and from the former Mainframes, Office System and Network Systems Divisions of the ICL Product Operations Group working on 5 sites.

The product was delivered on schedule in January 1989 to an independent validation, conducted by Inland Revenue and ICL staff, and was put into service in October of that year.

The objectives of the collaboration were achieved in that:

- the viability of the architecture was proven,
- network components of different types were configured, and
- the time and the cost of configuration were reduced significantly.

V250 currently configures 17 mainframes and over 400 terminals representing over 1000 objects requiring a database capacity of 300 M Bytes; it will soon be extended to cover 1100 terminals. Revenue regard the system sufficiently highly to have extended its scope to configure their IRON terminals from Bull.

The second phase of the collaboration, known as Community Generation Manager (CGM), is a generic product removing dependence on bespoke Inland Revenue datastores.

The CGM project is supported by three converging streams of development:-

- an Object Oriented database (OODB)
- a suitable graphical HCI and
- a refinement of the configuration application.

Each development stream is producing a series of increasingly complex prototypes that are being integrated into the application stream.

This prototyping approach was adopted to ensure that the viability of each level of complexity is established before proceeding to the next. Essentially the approach is a form of risk management for the development of complex products using unproven technology. The approach has proved successful to date with the first set of prototypes available for demonstration; work is well advanced on the second set of prototypes.

5 Conclusions

There is a serious need for tools to make network configuration generation easier and less error prone; without these the manageable size of network is serverely limited and there is a risk of significant financial losses from human error.

Inland Revenue and ICL have collaborated:

- to give clear expression of this need which would have been far harder for either to do independently.
- to bring into operational use an approach, which has been working satisfactory for 18 months, to meeting the need to develop the experience and knowledge which will now be made more widely available in future releases of ICL products (known as Open Systems Management Centre).

Acknowledgements

Many people have been involved in this work but special mention must go to Mark Bestwick (ICL), Clive Fenton (Inland Revenue) and Mike Headd (Northern Computing Services).

UNIX is a registered trademark of UNIX System Laboratories, Inc. in the USA and other countries.

References

GALE, A.C.: The Evolution within ICL of an Architecture for Systems Management. ICL Tech. J. Vol. 7 No. 4, pp. 673–685, 1991.
BARTHRAM, P.K. and HOWLING T.D.: Distribution Management – ICL's Open Approach. ICL Tech. J. Vol. 7 No. 4, pp. 702–717, 1991.
KAY, M.H. and RIVETT, P.J.: An Overview of an Raleigh Object - Oriented Database System. ICL Tech. J. Vol. 7 No. 4, pp. 780–798, 1991.
WILES, P.R.: Government IT infrastructure for the Nineties (GIN) – an Introduction to the programme. ICL Tech. J. Vol. 7 No. 2, pp. 412–431, 1990.

Biography

Jim White

Jim White took a PhD degree in physics at Birmingham University in 1975 with a thesis on the slowing-down time of neutrons in iron. He joined ICL in 1984 and from then to 1988 was project manager for CFT (Community File Transfer) and for CAM (Community Alert Manager). From 1988 to 1990 he was collaboration manager for the Inland Revenue project and from 1990 to 1991 was the development manager for SMPC. He is currently responsible for SM major projects and the SM Partnership Programme.

Operations Management

David Hacker
ICL Systems Management Product Centre, Basingstoke, UK

Abstract

Earlier in 1991 ICL launched a set of software products called the Open Systems Management Centre (OSMC). This paper describes the Operations Management component of the OSMC and explains what ICL customers need to do in order to achieve an improved level of manageability using the current release of OSMC.

The OSMC programme recognises that *manageability* needs to be an integral part of the development process of all applications in the future. Therefore ICL is working on the establishment of a number of relevant standards for Open Systems. In the interim, a number of facilities for operations management are provided that do not require changes to the applications involved. Such ease of implementation greatly enhances both the flexibility and speed of implementation of systems management across a network conforming to standards for Open Systems.

Where the end systems are not open, i.e. they conform to proprietary protocols, and where the OSMC infrastructure exists, manageability can still be provided through Operations Management, though this requires a small amount of bespoke application code to be written.

The paper shows how OSMC Operations Manager allows a service provider to be forewarned of problems before they impact the level of service provided to the user. Thus the service provider can be proactive rather than reactive. This behaviour is fundamental to improving both the actual and the perceived level of service computing offers to business.

1 Introduction

Operations Management is the stage at which all the detailed planning and development of a business system becomes a reality. It is the stage at which that system goes live. Nevertheless facilities for management of operations have been the least developed. Applications development has been, and continues to be, the focus of interest and activity with CASE tools. In too many cases when the application arrives at the Data Centre, far too little thought has been given to the way it runs and how it is to be managed in routine operation.

The shift to distributed systems and the associated client-server architecture compounds the problem. Only too often the Data Centre Management are firefighting; the first they know of a problem is when the user (phones up and) complains, yet again, of an unacceptable level of service.

Applications developed for a distributed environment are often just repetitions of some centralised application, there may have been no recognition of the fact that different issues have to be addressed in a distributed environment. Perhaps the most significant of these is a distributed office application which runs "standalone", but linked to the network with no skilled IT staff in attendance.

With the launch of its Open Systems Management Centre (OSMC) product, ICL offers a range of software that starts to address the operational issues of a (distributed) Data Centre. The particular purpose of the Operations Manager in the OSMC product set is to allow staff at a Data Centre to monitor and control existing applications remotely. Thus operational problems can be tackled more effectively at the centre in the light of a much more comprehensive picture of the situation than could be available locally. As a consequence action can often be taken before service becomes disrupted.

In this paper the fundamental principles of the OSMC Operations Manager are described, together with an indication of what each application owner must contribute to achieve a managed community of applications throughout an enterprise. It is the responsibility of each application owner to define how his application can be managed.

2 Operations Management within OSMC

The OSMC architectural framework (Gale, 1991) distinguishes three areas of Management:

- operational control
- introduction and deployment
- planning and reporting.

Operational Control is further sub-divided into:

- operations management
- problem management
- performance management aspects of capacity management
- activation management aspects of distribution management.

Operations Management, which lies at the centre of the OSMC, is concerned with the real-time flow and control of the work through the Data Centre and with the administration of the Data Centres' resources themselves.

Areas typically considered to be part of Operations Management include:

- job scheduling
- console message management
- tape library management
- file store management
- system archive and backup
- print production and distribution
- environmental control
- support and call-out.

Traditionally, most operating systems have provided only rudimentary facilities in these areas, thus leaving the way open for third parties to offer packages to the Data Centre's Service Provider.

The design of OSMC recognises the need to provide an integrated range of applications for use by the Service Provider in day-to-day operation at the Data Centre. Because computing is increasingly adopting client-server architectures, tools for operational management must themselves be distributed and operate over a common infrastructure picking up Open as well as de facto and proprietary standards.

Furthermore, most products available in today's market are passive monitors of status, that is to say, they only report status information. To realise the full benefit, the Service Provider needs to take action, via remote login etc., to make the necessary correction. It is even better if a common user interface can be provided for the disparate end-systems and applications now encountered in many networks.

3 Issues for the Service Provider

From an Operational Management perspective, a Service Provider is faced with a number of issues both technical and social:

1. Ignorance, on the part of operating staff, of what is happening, when and where across the distributed IT environment.
2. The need to be constantly fighting fires or reacting to pressures or threats, from users.
3. Pressures for longer hours of business, involving lengthier processing times and shorter windows for background tasks such as system administration and housekeeping.
4. Growing constraints on costs and on-going shortages of skilled IT staff.
5. Social and financial pressures to reduce shift working, especially at night and at weekends.

6 Insufficient attention given to operational aspects of managing applications, more especially distributed applications.
7 A lack of tools to help run a distributed environment.

As a result many Service Providers feel themselves to be running fast merely to stand still. Their users on the other hand may see the service provided as expensive, unresponsive and late.

4 OSMC Operations Manager

OSMC Operations Manager caters for remote management of networked systems. Its facilities enable a Service Provider to take a comprehensive view of all the components that together deliver the services for which he is responsible. He can decide which areas most need corrective action, taking the business view rather than a technological view of what should most urgently be fixed.

Central to OSMC Operations Manager is an ability to display information from and manage a number of different types of system from a single location. Systems are displayed graphically as colour coded glyphs (Small et al, 1991). Each glyph can represent a number of managed objects (servers or applications) which in turn can be broken down to finer levels of granularity. This removes the need for first-level operations staff to be familiar with the individual command sequences of the different operating environments.

OSMC Operations Manager is available on both Series 39 and DRS 6000 central management systems. A number of OSMC Operations Manager centres can be distributed across an organisation, hosted on any combinations of Series 39 and DRS 6000 central management systems. These centres can, if required, be networked together to form a regional management capability.

A schematic of OSMC Operations Manager is shown in Figure 1.

The software components of OSMC Operations Manager addressed in the rest of this paper are:

1 Operations Control Manager
2 Status Monitoring
3 Operations agents for VME and for UNIX

4.1 Operations Control Manager

The Operations Control Manager (OCM) is a graphics based system using a high resolution SUN 4 SPARC workstation (Small, 1991). The OCM provides facilities to display graphically, either logically or geographically, significant status information about the managed IT objects (systems and

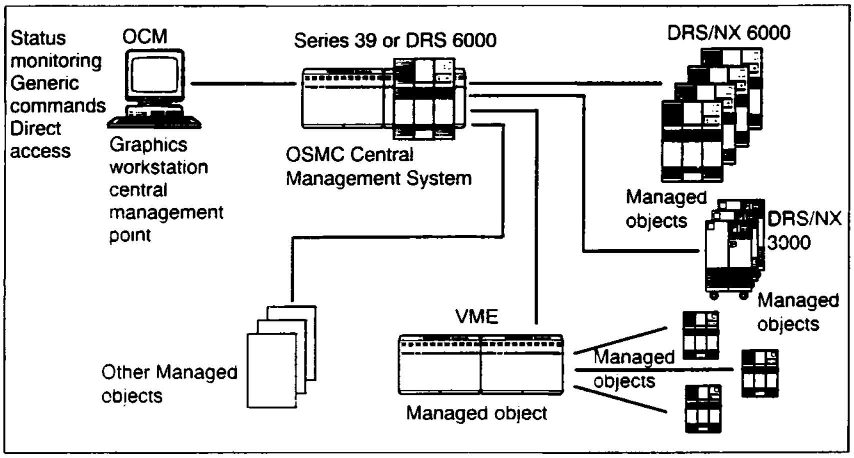


Fig. 1

services). The information displayed can be exploded to show greater detail of the network of managed objects and may be accessed from any level in this hierarchy. The OCM includes a knowledge base of the objects being managed.

The high resolution graphical display of the OCM is based on the 'Open Look^{TM}' User interface standards and provides:

- standard glyphs to represent managed objects
- colour to indicate status of managed systems
- display layout editable by the user
- display expansion facilities
- automatic propagation of status change
- menus of actions available for each managed object
- access to DRS/NX and VME systems by terminal emulation.

Through the OCM, Operations Manager provides facilities for remote management of the IT infrastructure:

- monitoring of status, prompts and alerts in real time
- operational control
- monitoring of performance and file threshold in real time
- archiving and housekeeping remote systems
- automatic operation of VME systems.

To reduce the need for the central OCM operator to have specific operating system knowledge, the command structure of managed objects is represented as a set of generic commands to initiate those functions common to all environments, such as loading services and showing print queues. These commands are initiated from and the results displayed by, the OCM (see Figure 2).

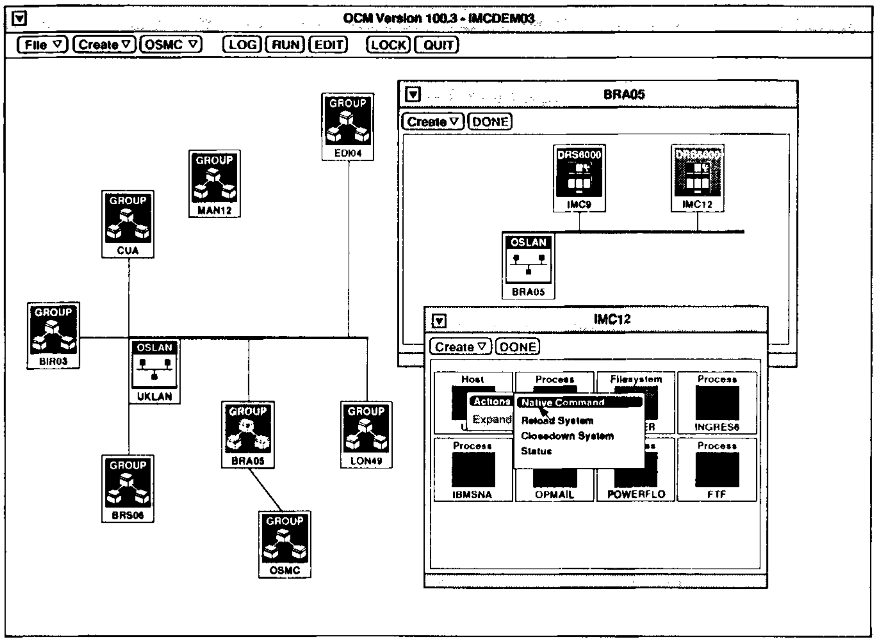


Fig. 2 Diagram (to improve clarity) from actual photo. Icons show connected systems and represent status of groups of machines at each location. Location BRA05 has been expanded showing two DRS6000's there; one DRS6000 has been further expanded (IMC12 window) to show status of processes running on it. The operator can now select further information or issue commands from the menu.

4.2 Status Monitoring

Status Monitoring provides facilities to monitor the current status of objects and to record the history of their change of state via a *sponsor* at the managed object. A management system may have several Status Monitors. These may be linked in a hierarchy or operated independently. Each may address an individual view of objects being monitored even on the same host platform. For instance one view might be of the databases distributed across the network while another view could be of the Office Power applications running in a particular geographical region.

Operations Manager provides the required flexibility, since every customer installation will be different when organisational, resourcing, political and other factors are taken into account.

OSMC Status Monitor follows the ISO/CCITT standards for Operations Management and status conditions. Status is defined in accordance with

OSI standards (ISO 10164: Part 2). Status information on managed objects can be provided in three ways:

4.2.1 Poll and Response The managing status monitor polls its managed objects to enquire on their current status plus the status of objects that it in turn knows about. The managed element responds if it is able to do so.

4.2.2 Heartbeat The managed object provides alerts at regular intervals as defined in the status database, thus notifying the managing status monitor that it is still working.

4.2.3 Status Change Notification The managed object notifies its status manager whenever its status changes, or when one of the objects it is monitoring does so.

The Status Monitor uses this status information to determine the operational status as defined by ISO (active, busy, unknown, disabled or enabled) and the ISO administrative status (unknown, locked, unlocked or shutting down).

Combinations of operational and administrative status are used to determine the overall management status of the object being monitored. This is then fed into and displayed by the Operational Control Manager.

4.3 Operations Agents

Operations Agent is the name given to software that runs on a remote managed system, and provides the status information requested for display upon the OCM.

An Operations Agent is a very powerful tool and one that is easy to implement. All software, whether system or application, needs to exploit the Operations Agent to realise the first phase in Systems Manageability.

Using the Operations Agents (either VME or UNIX) the interaction of the application with its environment can be monitored *without necessarily making any change to the applications.* These interactions can relate to specific events or to thresholds being exceeded.

Use of Operations Agents is very simple to implement. It is called the Remote Management System (RMS) and there are versions on both VME (where it is also known as VSS) and UNIX (Ref. 3).

Configuration of RMS for both VME and UNIX requires a number of clearly defined parameters to be submitted, including:

1 name of the *object* to be monitored (ie its file name, process name etc)

2 the *threshold* above which a status change will occur (ie percentage of file store utilisation or percentage of cpu utilisation)
3 the *interval* for reporting status information.

Any number of objects can be monitored on an individual remote system. For example each of five business applications, their major files, communications and swap space could be monitored individually.

All product authorities and application developers need to establish how best to implement RMS on their environments. The information needs to be published as part of their supporting documentation.

4.4 Current Implementation

The list of objects currently monitored by the VME and UNIX RMS is given in Table 1 with a brief summary of what is available at the current release of OSMC (OSV230).

Table 1 Objects that may be managed by RMS under VME or UNIX.

	UNIX objects via RMS	VME objects via RMS/VSS
Host	Status of UNIX server	Status of VME server
Process	Status of any named UNIX processes	Specific VME processes are CAM, TPMS 510 and 520, IDMSX, VCMS and ASO
Processor use	Percentage threshold monitoring of UNIX swapspace	Percentage threshold monitoring of VME secondary store
File space	Percentage threshold monitoring of named UNIX files	Percentage threshold monitoring of named VME filestore objects
Printers	Status of named UNIX Printqueues	Status of VME spooler
Communications	Status of UUCP and status and percentage threshold on OSLAN and X.25	Status of file transfer
Peripherals	Using basic status sponsor and user written application	Status of named VME peripheral hardware units

5 Conclusions

The OSMC Operations Manager is expected to provide a simple but powerful tool to enable the status of remote servers and their services to be monitored and displayed centrally.

The current release of OSMC also provides tools for monitoring an existing application without affecting that application. This enables the service providers to begin very quickly to gain the benefits of using OSMC.

Operational Control Manager not only allows corrective action to be taken (quickly) but allows that action to be done in a preventive fashion ie before too much damage has resulted and the levels of service impacted.

The use of icons to initiate remote logon and subsequent command sequences using a simple human interface, irrespective of the environment of the remote object, simplifies and speeds up the responsiveness of those working at the management centre.

What is essential is that the staff of a Service Provider are seen to be "in control of the situation" at all times, or, in other words, that they have the facilities and are both trained and organised in advance to respond appropriately to any changes in operating conditions or to any crisis, and are not merely obliged to cope unprepared as best they can.

The task of providing implementation rules for the operations agents for ICL applications is believed to be simple and straight forward.

Acknowledgements

UNIX is a registered trademark of UNIX Systems Laboratories, Inc. in the USA and other countries. SPARC is a trademark of Sun Microsystems Inc.

References

1 GALE, A.C., An Evolution within ICL of an Architecture for Systems Management, ICL Tech. J. Vol. 7 No. 4, pp. 673–685, 1991.
2 Information Processing Systems. Open Systems Interconnection. Systems Management Overview. ISO/IEC DIS 10040.
Information Processing Systems. Open Systems Interconnection. Systems Management – Part n. ISO/IEC DIS 10164 n (n = 1–7).
Information Processing Systems. Open Systems Interconnection. Systems Management – Part n. ISO/IEC CD 10164 n (n = 8–11).
3 VME RMS Manual. CISG/CCC/CDS/449.1 (Issue 1).
UNIX RMS Manual. 11092/001.
4 SMALL, M. et al., OSMC Operations Control Manager, ICL Tech. J. Vol. 7 No. 4, pp. 751–762, 1991.

Biography

David Hacker

David Hacker is the Marketing and Services Support Manager for the Open Systems Management Centre. He has been with ICL for four years, working on Systems Management in both Development and Marketing capacities. Before joining ICL he worked for a number of years as a consultant, advising clients on running their data centres and taking personal responsibility for data centre operation as a consultant and as a line manager previously. In his current role, He has responsibility for the marketing of OSMC, the consultancy support required in installation and commissioning on customer sites and any subsequent 3rd and 4th line support issues that may arise.

OSMC Operations Control Manager

M. Small, J. D. Mitcalf, K. J. Johnstone, J. W. Doores
ICL Strategic Systems, Cardinal House, Manchester, UK

Abstract

A company which operates a large IT system often has a number of mainframes and office systems. These may be in different locations and each location may be supported by a team of IT personnel. The Operations Control Manager (OCM), which is part of the OSMC system allows the management to control these systems from a central location.

The OCM uses a graphics based workstation to display a view of the entire network. It displays the status of specified objects, so allowing the central operator, to investigate and issue commands to the managed systems. When necessary it is possible to expand and view parts of the network in greater detail.

The OCM was developed using ICL's Marketing-to-design methodology. This methodology incorporates human factors into product specification and design resulting in products which are more acceptable to their users.

1 Introduction

A company which operates a large IT system often has a number of mainframes and office systems. These may be in different locations, and each location may be supported by a team of IT personnel. The Operations Control Manager (OCM), which is part of the OSMC system, allows the Management to control these systems from a central location.

The OCM uses a graphics based workstation to display a view of the entire network. It displays the status of specified objects, so allowing the central operator, to investigate and issue commands to the managed systems. When necessary it is possible to expand and view parts of the network in greater detail.

The OCM allows the management of a large number of objects. These may be complete mainframe systems, or items related to these systems, such as services or terminals. When the OSMC system is installed, a database known as the OCM knowledge base is set up which mirrors the configuration of the IT system. This may subsequently be amended to reflect any changes in organisation. The managed objects may be remote systems running under VME or UNIX.

Using the OCM is intuitive. Menus and commands are defined to make the system easy to use, and there are prompts and pre-defined forms which make selections and data input easy to understand.

The OCM system is designed to fail safe. If any faults occur on the OCM or the central system they will not have any effect on any of the managed objects. If a fault does occur, any incoming status changes from the managed objects are stored until the system is back on-line.

Clearly the facilities provided by the OCM are very powerful and must be secure against unauthorised use. To log in to the OCM you must supply a user name and a password. Once you have gained entry to the system, you can explicitly lock the OCM terminal without disconnecting it from the OSMC system. When the system is locked, the screen goes blank and the system can only be unlocked by supplying the password of the original user.

2 The OCM within the OSMC

The OCM is based on a SUN workstation with a high resolution colour graphics screen. The workstation runs on the SunOS 4 UNIX operating system. Figure 1 shows the position of the OCM within the OSMC system.

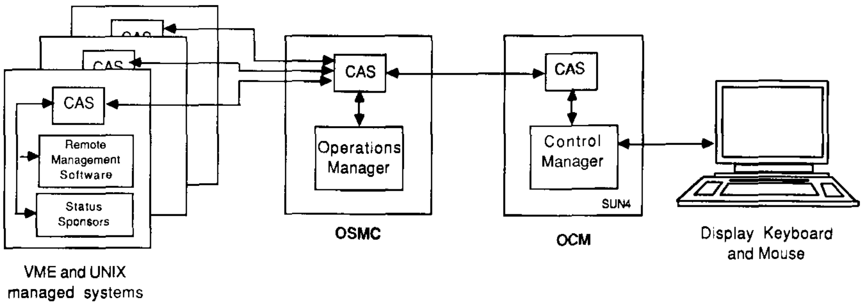

CAS
CAS
CAS
Remote Management Software
Status Sponsors
VME and UNIX managed systems
CAS
Operations Manager
OSMC
CAS
Control Manager
SUN4
OCM
Display Keyboard and Mouse

Fig. 1 The OCM within OSMC

The principal source of information for the OCM is the Community Status Monitor (CSM), which runs within OSMC Operations Manager. The CSM receives requests for status, which it either responds to directly (from information which it has already collected), or passes the request on to a managed subsystem. Information is either aggregated or passed on to CSMs at a higher level, therefore the CSM must be configured so that missed heartbeats or changes of state detected in systems being managed from the OCM are notified to the OCM.

For the transmission of commands from the OCM to the managed objects, remote management software (RMS) and status sponsors must be installed and running on the remote system. This combination of OCM and RMS allows the control of a managed object from the OCM using the menus of actions held in the OCM. When an action is selected at the OCM this is transmitted to the RMS via a CMIS/P interface. The RMS then executes the command on the managed object and returns any response to the OCM.

3 OCM Design

To ensure that the OCM was designed to match the requirements for centralised operational control, a detailed study was undertaken. This study followed the ICL Marketing to Design Methodology described elsewhere (Hutt, 1990). This considered in detail the user roles, their tasks, the processes followed and the objects involved.

3.1 Organisation

While IT departments may have a common purpose they do not have a typical organisational structure. This is further complicated by the trend toward centralisation and the presence of equipment from a number of vendors.

An organisation may have started to move towards centralised operation with some sites centralised while still retaining some distributed sites. Managing this transition is in itself a significant challenge.

An organisation may have equipment on some sites from different manufacturers. Where there are a number of vendors, it is often the case that the IT organisation is split at a high level according to vendor. This poses further problems when trying to centralise operations.

Thus it is important that the OCM is flexible enough to match a range of organisational structures. This is achieved by considering the roles that will make use of the OCM rather than the job titles.

3.2 Roles

At the highest level the roles involved in providing computer services may be divided into 2 categories:

Planning roles: concerned with ensuring that the computer services provided continue to be satisfactory and efficient in the future. For example capacity planning, service level agreement and change control.

Operational roles: concerned with the real time activities of providing a satisfactory computer service today. The OCM is intended to support these roles which are:

- Operating Systems Operator
- Event Monitor
- Job Scheduler

The role of the Operating System Operator is undertaken by staff in operations performing tasks to progress the workload of jobs.

The role of the Event Monitor is normally undertaken by operations staff responsible for the day-to-day running of the machines, i.e. the shift supervisor or senior operator.

The role of Job Scheduler is undertaken by operations staff responsible for ensuring that the jobs in a workload run as scheduled with the correct priority given to the appropriate jobs.

3.3 Tasks

Providing computer services involves two basic tasks:-

- following a predetermined schedule to run jobs, do housekeeping, archiving, etc.
- monitoring services – when events occur, prioritise them and respond to items in this event agenda.

3.4 Objects

The principal classes of *objects* are the following:-

- *operable objects:* hardware, services, queues, jobs and processes.
- *resources:* filestore, memory, processor time.
- *events:* alarms, messages, operating system prompts etc.

While similar in concept, the details of these objects vary considerably between operating systems and hardware types.

3.5 Processes

The processes involved in the central operations task are represented in Figure 2. The boxes are user roles and the arrows indicate the passing of control of an action.

3.6 Attributes Required

From the study described above, a number of conclusions were reached about the necessary attributes of the OCM.

a) overall graphical presentation

In centralised operation the operations staff need to have a representation of the whole IT infrastructure to work with. They need this to identify

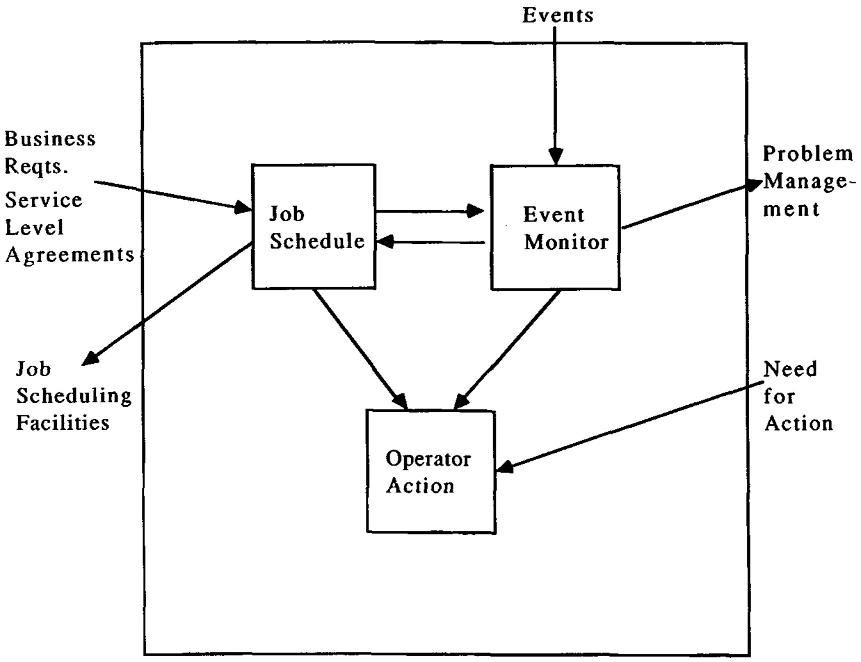


Fig. 2 Process Model for Central Operations

problems quickly, as they occur, within a large and complex environment. Therefore a multi-level display is required. This should provide both an overall summary from which it is easy to see whether or not there are problems together with detailed expansion to allow specific problem areas to be investigated.

b) unified control interface

Different hardware and operating systems provide different ways of performing similar tasks. In a multi-vendor environment with central control this will pose a skills problem. To reduce the range of skills necessary a unified control interface is required at the OCM, isolating the operational staff from the intricacies of the systems being managed.

4 OCM Interface

4.1 Management Interface

The OCM is used to display and manage the whole, or sections, of a company's IT network. Figure 3 shows an example of the type of network which may be managed by the OCM.

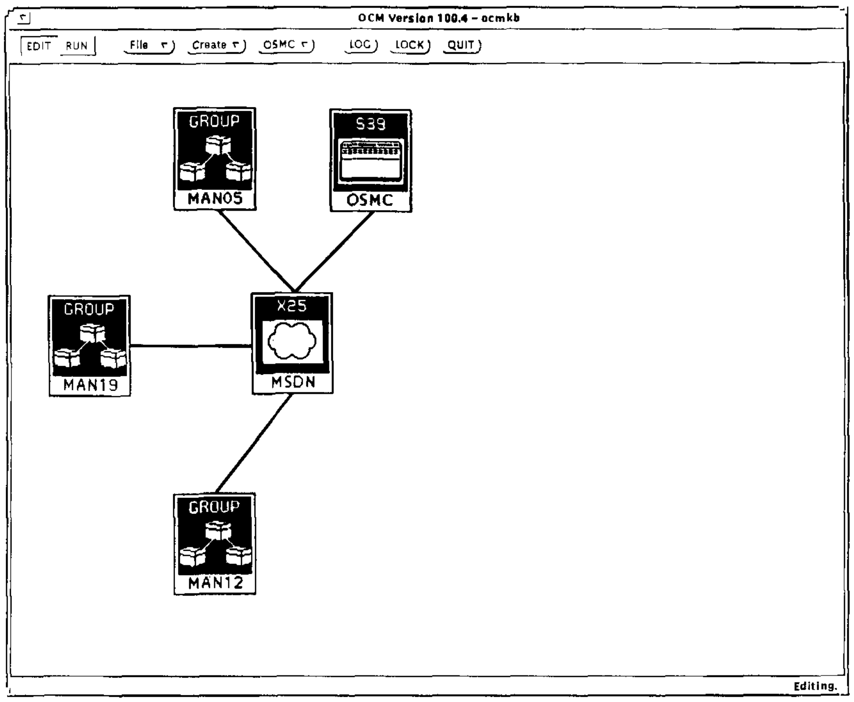


Fig. 3 Overall Network Display

The representation of the network is created, and may be amended, using editing facilities which are available with the OCM. The configuration of the network and the properties which identify each object which is managed, are held on the *managed object knowledge base*.

The user interface of the OCM is entirely isolated from the operational inconsistencies of the objects it manages. This is of benefit to the user, who need not learn the intricacies of the systems under his supervision.

The OCM interface is based on the OPENLOOK^{TM} user interface. For further information on its facilities, see (SUN, 1990).

4.2 Managed Objects

The individual systems or system components which are managed, are known as *managed objects*. They may be:

- managed computer systems, such as VME or UNIX systems
- services, such as TP
- important network components, such as schedulers or printers

The managed objects are represented on the OCM screen by glyphs*, each of which is coloured to represent the current condition, or status, of its corresponding managed object.

The OCM uses different glyphs to represent different systems.

Glyphs may represent:

- managed systems, such as VME or UNIX
- components of managed systems
- networks, such as OSLAN or X25
- groups of managed objects. These are complete systems which are grouped together for geographic or other reasons

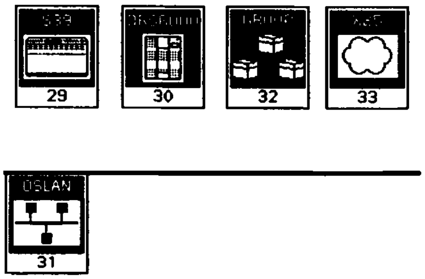


Fig. 4 Examples of Glyphs

Examples of glyphs are shown in Figure 4.

When the system is running, the glyphs are coloured in to represent the status of each managed object, and they change colour to reflect a change of status. For example, a glyph may be green to represent that the object is on-line and running. A change to red may indicate a problem which needs investigating. A black glyph indicates that the managed object is not supplying performance data. For more information on the use of colour see Van Larr (1990).

On the diagram glyphs may be linked to show connections in the network. They can also be expanded. For example, a *group glyph* may be expanded, when necessary, to show all the separate managed objects within the group. An expansion can also be used to show all the underlying objects of a system (such as printer queues or file store).

The glyphs and the colours used to indicate a change of status are set up at installation. The colours used are standard over all the managed objects. The default colours may be tailored on installation.

*"glyph" is the OPENLOOK term for a picture used as the title for an object. The term "icon", which is more commonly known, is reserved for the closed representation of the base window to an active application.

4.3 Managed object containment tree

The OCM uses the concept of a *containment tree* to define a hierarchy of managed objects.

An object, which is defined on the OCM, may itself contain other managed objects, each of which may transmit status information to, and receive commands from, the OCM.

For example, a UNIX managed object may be defined which may itself contain managed objects. These will be the components of the system, such as printers or filestore, or services running on the system, such as TCP/IP. If they are defined to do so, these contained, or underlying objects may each transmit status information to the OCM and receive actions in return.

Therefore, any underlying object may be managed in its own right, although it is contained within another managed object, hence the term containment tree.

The managed objects contained within another may be viewed by opening an expansion window, as in Figure 5:

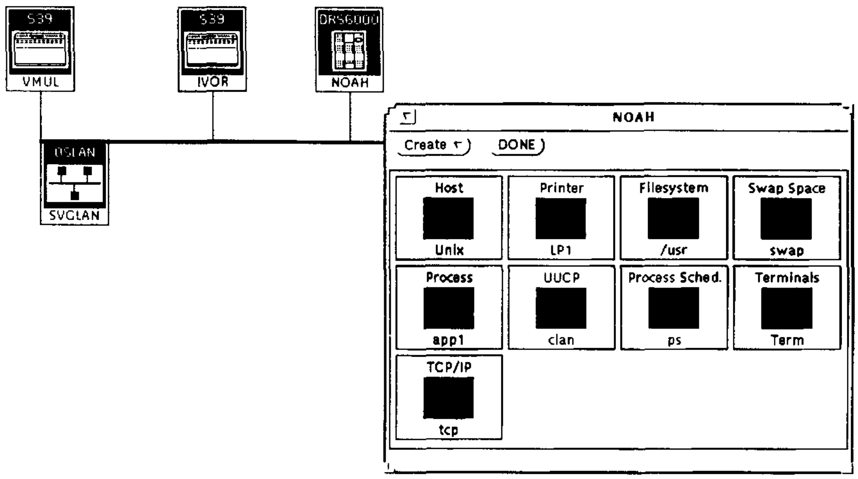


Fig. 5 Managed Object Expansion

If there is a change of status of an object within a containment tree, the associated colour change is propagated upwards. Therefore, if the top level glyph on the display changes colour, an expansion window can be used to identify the object which has caused the change.

4.4 Action Menus

A menu of actions is associated with each glyph. The particular actions depend upon the type of managed object. These come from the class definition hierarchy associated with the managed object in the knowledge base.

Thus, if a glyph indicates a change of status of an end-system, it is possible to determine from the OCM which sub-component has changed status, and issue a command from the menu associated with the sub-component. As an example, if the sub-component is a printer, one can decide to disable the device. Figure 6 shows the actual menu for printers on UNIX.

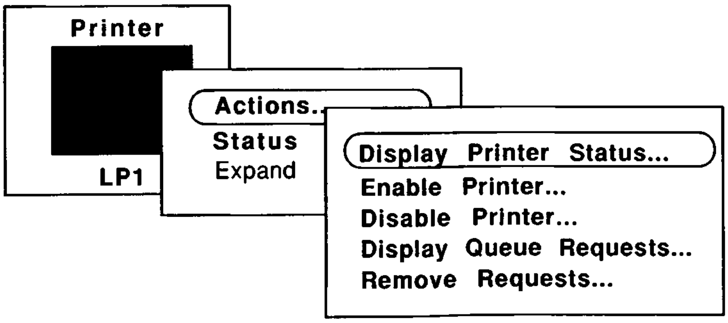
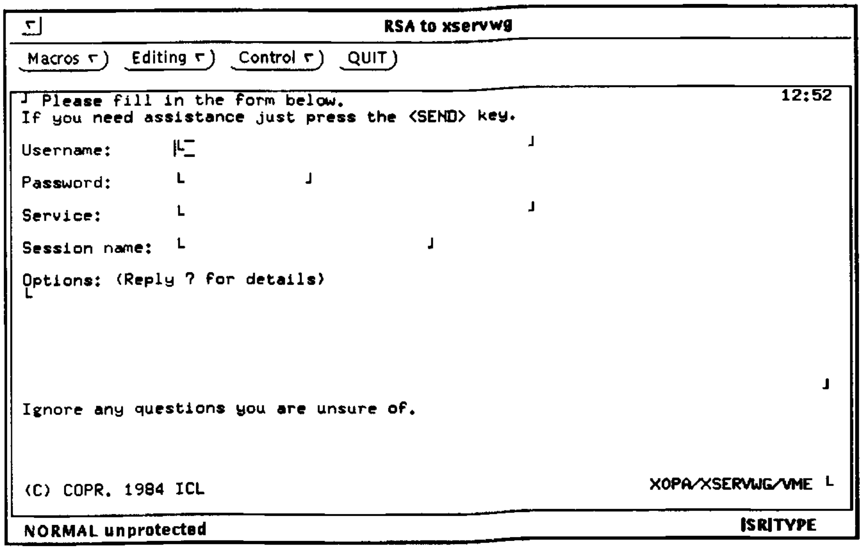


Fig. 6 Action Menu Example

4.5 Terminal Emulation

One can carry out most operations by menu selection. However, one may also use the OCM as a remote terminal to enter commands or make enquiries of a managed system. Figure 7 shows a terminal emulation window to VME.



```
Please fill in the form below.                                12:52
If you need assistance just press the <SEND> key.

Username:

Password:

Service:

Session name:

Options: (Reply ? for details)



Ignore any questions you are unsure of.


(C) COPR. 1984 ICL                               XOPA/XSERVWG/VME
NORMAL unprotected                                     ISRITYPE
```

Fig. 7 VME Terminal Emulation

5 Summary and Conclusions

This paper has described the OCM, a graphics based system presenting a unified management interface to VME and UNIX systems. The OCM was designed using the ICL Marketing-to-Design methodology which identified the key useability requirements. The OCM has been implemented on a SUN4 workstation using OPENLOOK^{TM} graphical standards. The OCM works in conjunction with the other standard OSMC products using standard open interfaces.

The experience at the time of writing is that the ICL marketing-to-design methodology proved to be effective, that the OCM matches the requirements of users for the systems operations task. Also it enabled a system to be designed, that is flexible enough for a variety of organisational structures and which can be extended to cover the management of hardware and operating systems provided by other suppliers.

Acknowledgements

UNIX is a registered trademark of UNIX Systems Laboratories, Inc. in the USA and other countries.

OPENLOOK is a trademark of SUN Microsystems Inc.

References

HUTT, A.T.F. et al., The Development of Marketing to Design. The Incorporation of Human Factors into Product Specification and Design. ICL Tech. J. Vol. 7 No. 2, pp. 253–269, 1990.
SUN MICROSYSTEMS INC., OPENLOOK Graphical User Interface Application Style Guidelines, ISBN 0-201-52364-7
VAN LARR, D. et al. How to use colour in Display II. Coding, Cognition and Comprehension. ICL Tech. J. Vol. 7 No. 2, pp. 362–383, 1990.

Biographies

Mike Small

Mike Small is responsible in Strategic Systems for the design of knowledge-based systems management applications and, currently, for the Open Systems Management Centre Operations Manager. This is a real-time expert system for monitoring and controlling a set of services running on multi-vendor equipment to be controlled over an OSI network, and incorporating manageability definitions of services based on standards emerging from the OSI Network Management Forum.

He started in ICL at West Gorton nearly 25 years ago, with design of mainframe computers. He was next responsible for software tools for developing 2900 test software and, later, for test software for design, manufacture and maintenance of the ICL ME29 series.

Since creation of the Knowledge Engineering Group he has been concerned with commercial exploitation of software technologies and tools derived from research into AI. He was responsible for system design of VCMS (VME Capacity Management System), a knowledge-based system for planning and monitoring performance and capacity of VME systems, which won the IPC "million in one" award in 1987. He also received the ICL GOLD quality award for this work. He was responsible for OMAC Expert, an expert system forming part of OMAC 2000, a system allowing customers to model decision-making by their material control staff in the form of rules. These rules are then used by OMAC MRP to confirm selectively raising of orders and for supression of MRP actions.

He has published many papers in the ICL Technical Journal and the proceedings of international conferences.

David Mitcalf

David Mitcalf took a degree in management science in UMIST in 1979 and joined ICL in the same year. He worked on the STAGE II project for the DHSS developing front end processor software. In 1983 he joined the newly formed Knowledge Engineering Business Centre working on ADVISER, an ICL expert system shell. He has been in Strategic Systems for some 8 years as a technical team leader. In that capacity he has worked on OCM, for the past 18 months, as technical design authority. During that time he was responsible for supervising the implementation of OCM.

Kevin Johnstone

Kevin Johnstone graduated from Queen's University Belfast in 1973 with a degree in Mathematics and Computer Science and joined ICL where he worked as a programmer in Engineering Diagnostic Systems and later Management Information Systems. He joined the Knowledge Engineering Business Centre in 1982 and was a mmeber of Alvey DHSS Demonstrator project for 5 years.

Jim Doores

Jim Doores joined English Electric Computers in 1967 working as a h/w engineer on the System 4/70's and later with ICL on 1906A systems. In 1973 he moved to a Systems Integration Unit validating early versions of VME on 2970/80's. After working on VME S/W support and Test S/W specification for Series 39, he joined the Knowledge Engineering Group in 1983, where he designed PROLOG environments for VME and UNIX users.

In 1986 he designed a Knowledge Based System for the Statistics Office of the European Community in Luxembourg. This used a rule set encapsulating expert knowledge of a statistician to provide Import/Export trade forecasts for the EC in Brussels.

He then spent two years on the EUROHELP project, designing tools using LISP to facilitate the production of Intelligent Help Application Models. Since then he has worked on the OCM project responsible for type design of the Managed Object System and Message Handling Functions implemented in PROLOG using the DECISIONPOWER toolkit.

The Network Management Domain

Tony Maynard-Smith
Network Systems, ICL Secure Systems, Stevenage, UK

Abstract

Network Management and Systems Management have many similarities and overlaps, but an understanding of the differences between them is important for a proper appreciation of the directions being taken in the two areas. The background to the development of network management is described, with the current position on standards and the various styles of integration appropriate to different circumstances. ICL's approach to delivering network management is described.

1 Introduction

This paper describes the current position of the market, and the technology appropriate to the management of networks, and shows how the subject of Network Management relates to that of Systems Management. The differences in product development between the two subject areas is traced to the different requirements placed by the market conditions in which they originated. Despite these differences however, there is a very considerable overlap between them which is leading to increased requirements for integration, both horizontally and vertically.

This paper first describes the background to the development of Network Management, its scope in relation to Systems Management, and the various styles of integration which are applicable within the network and between the network and system levels. The current situation regarding standards in the field of Network Management is described, and then the manner in which actual products are being realised at the different levels.

1.1 Scope of Network Management

Part of the confusion surrounding this subject is that there are no universally accepted meanings for the terms "Systems Management" and "Network Management". In some cases "Network Control" is also used with a different emphasis from "Network Management". The relationship between these terms, as used here, is shown in diagrammatic form in Figure 1. This indicates that Systems Management has a very broad scope, in a "horizontal" dimension, encompassing common functions across a large number of Domains. Network Management is one such Domain, with a relatively narrow scope in the "horizontal" dimension, but with a full "vertical" range of functionality. In other words it is considered to include all the functions necessary to manage a network effectively and efficiently. Network Control is a term used by some authorities to cover the operational aspects of network management, and specifically to exclude many of those administrative functions addressed by Systems Management. Alongside the Domain of Network Management are analogous Domains responsible for the management of say UNIX end systems, X.400 Message Handling Systems, Business Applications and so on.

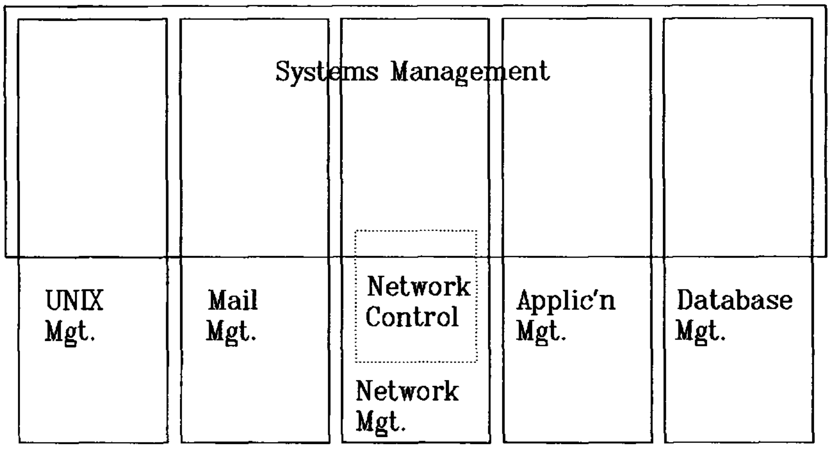


Fig. 1 Scope of Systems and Network Management

If a broad view is taken, the process of management is the same whether considered at the systems or network level. This process is shown in Figure 2, reproduced from [ICL SMA]. This shows the Quality Process of managing the total lifecycle of an IT system, broken down into ten major functional areas. These functions are generally applicable to any of the Domains which could be shown in Figure 1, though of course the implementation details differ in each case. It is the concern of Systems Management to provide the commonality and integration required across these Domains, and the concern of Domain Management, such as that for networks, to address the full range of functions in the detail needed within its particular scope.

1.2 Background to Network and Systems Management

If the practical development of Systems Management and Network Management products is examined, it can be seen that they have come from rather different backgrounds, which has led to rather different outcomes in terms of current product and capability.

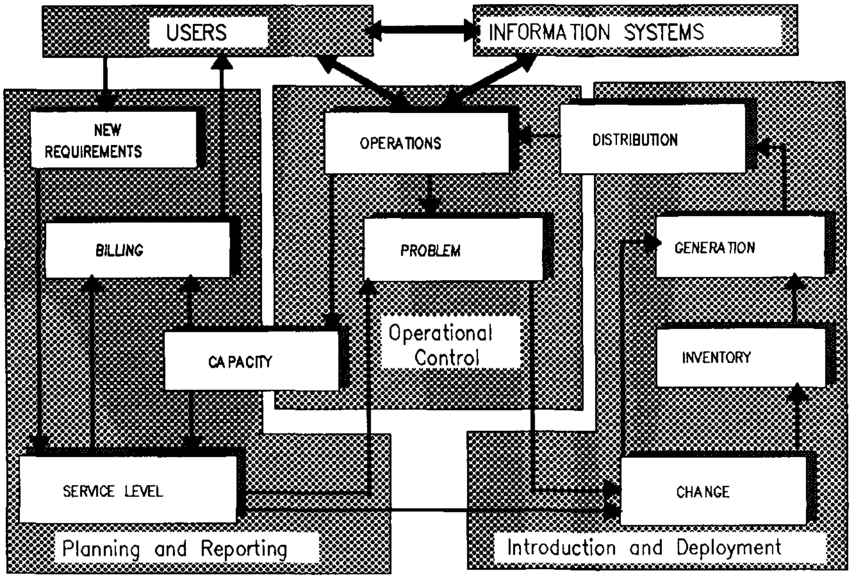


Fig. 2 The Systems Management Quality Process

Systems Management has in general been developed by major players in the data processing industry who have recognised the broad scope of the subject and put in place comprehensive architectures to cover the range of functionality required. Examples of this are IBM's Netview, DEC's Enterprise Management Architecture (EMA), and ICL's Systems Management Architecture (SMA). There are relatively few such architectures in the industry, and several of them are converging on the OSI standards for Systems Management (ie. [CMIP], [CMIS] and the many other supporting standards associated with this protocol).

Network Management by contrast has been developed in a much more tactical manner by the manufacturers of networking products. This has been in response to the imperative to provide remote control over the systems they deliver, which are by nature geographically distributed. Given that there are a large number of such manufacturers, and a wide variety of types of networking equipment to manage, there is now a very considerable variety of product available in the area of network management. This variety extends to the communications protocols used (which are often proprietary), to the level of functionality provided, and to the platforms and development tools used to build them. All of these factors make for difficulties in integrating the results.

While by no means a rigid distinction, it is generally true that the two disciplines have also concentrated on differing functionality. Network Management has concentrated on Operational Control, providing the ability to operate a network from a central control centre, while Systems Management has put more emphasis on the administrative functions which are common across a number of Domains and which are increasingly needed as the scope of the management problem grows.

The issues facing Systems Management can therefore be seen to be breadth of coverage, conformance to standards, and populating the functionality space with generic but useful applications. The issues for Network Management are integration, reduction of variety, selection of appropriate standards, and access to a range of administrative applications.

2 The Integration of Network Management

The key issue in Network Management today is seen to be integration. There are a number of drivers in this direction, including:

- the need to manage a diverse and distributed multi-vendor network from a single control centre, because of the need to make best use of staff and skills.
- the need to simplify the tasks of network management staff, for cost effectiveness and to improve Quality of Service. This means reducing or hiding the variety presented to the users of management systems.
- the need to handle the variability in networking products, in the technology involved, in the topology of network designs, and in the users' own organisational structures and geographies.
- the need to add administrative applications which directly address the issues of efficiency and cost control, such as asset management, inventory, change management, and Service Level Agreements.

While integration is a key element in the solution to these issues, it must be noted that since all of these are also changing with increasing rapidity, flexibility of any solution is also a key requirement.

2.1 The Hierarchical Model

The simplest approach to integration, and the one most widely adopted so far, is a hierarchical model with a "Manager of Managers" at the top, a set of individual Element Managers in the middle, and the actual devices being managed at the bottom. This class of model is shown in Figure 3. Typically the top level manager in such a hierarchy provides a consolidated view of the state of the network, a log of events or alarms, and some level of control capability. The latter is frequently achieved by providing a window which emulates the console on the Element Manager. This model is behind the design of architectures such as IBM's Netview, AT&T's Unified Network Management Architecture, the Allink product from Nynex, and a number of others.

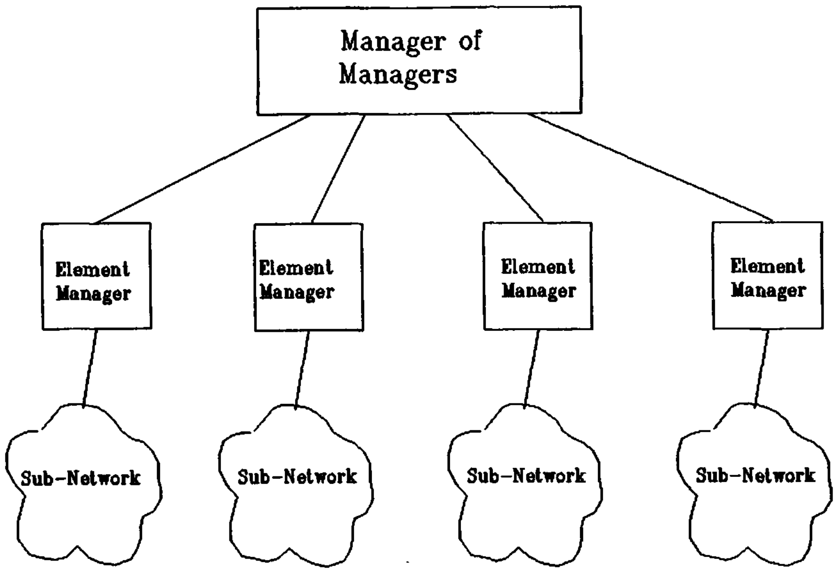


Fig. 3 The Hierarchical Management Model

This style of integration is quite satisfactory for some purposes, and in particular does help to provide a consolidated view of the state of the network which is not available from the multiplicity of individual Element Manager screens. However, looking at the full set of requirements outlined above this model is not altogether adequate, particularly with regard to flexibility.

Firstly it is implicit in the hierarchical model that information flows up the pyramid to some higher intelligence at the top, and decisions or control flow back down, to be executed by the relatively unintelligent systems at the bottom. As the number and variety of the lower level Element Managers increases, and the number of management functions to be performed also increases, the complexity of the data processing required at the centre increases geometrically. This rapidly becomes impossible to maintain, and some more comprehensive model of the system being managed, and a more appropriate structuring of the management information is needed to cope. This can be addressed partly by placing more intelligence in the network itself, and partly by adopting the alternative management models described below.

Secondly the hierarchical model assumes that there is a centralised system which has responsibility for managing all the lower level systems under its control. This is quite appropriate for collecting a number of sub-networks together into a network control centre, but less so when one is considering networks alongside mainframes, distributed UNIX systems, networked PCs, database and application services, and so on. A management system exists to support the people doing the job, and the structure of the system must fit the structure of the human organisation involved. Human organisations are no longer usually simple hierarchies.

2.2 *The Service Domain Model*

It will therefore be necessary to provide a richer model of system structure and management data flows than is available in a simple hierarchy of control. An important model which can be used to capture this structural complexity is that of Service Domains, adopted by the ICL Systems Management Architecture. A Service Domain is some collection of resources which supplies a defined service to an identifiable set of users. Service Users in this sense may be human end-users or other Service Domains, themselves supplying a higher level Service.

A network, or part of a complex network, can very often be considered as a Service Domain providing services such as interconnection, access to applications, transport of data, etc., upwards to other Service Domains. An example of such a collection of Service Domains is shown in Figure 4 which is adapted from a case study on how IT Services should be organised within ICL.

In the Service Domain model there is a management function present in each Domain, whose purpose is to ensure that the Domain continues to deliver the service it is responsible for, in accordance with a Service Level Agreement. This view of Service Users and Providers has a number of consequences for the structure of management systems:

- Each Domain must provide, or have access to, the full range of management functions to support the system lifecycle as shown in Figure 2.
- Each Domain will report to its Users, and receive requests from its Users, in terms of the Service supplied, not in terms of the resources within the Domain. Thus management interaction between Domains will relate to, say, a connection of a certain bandwidth between two end systems, not to the switches and communications lines used to achieve this. The management function within a Domain is responsible for maintaining this abstracted view.
- Since a networking Domain will typically have many Users, it will be necessary to report to, and receive instructions from, many places. There is no single intelligence at the top of the pyramid which understands everything.

It would be possible to maintain such a system in a totally distributed manner, with each Service Domain being complete and self-contained, interfacing to its neighbours only at the boundaries. Indeed if one looks at networks of legally independent entities this is the way they operate. Taking the international telephone network as an example, each country manages

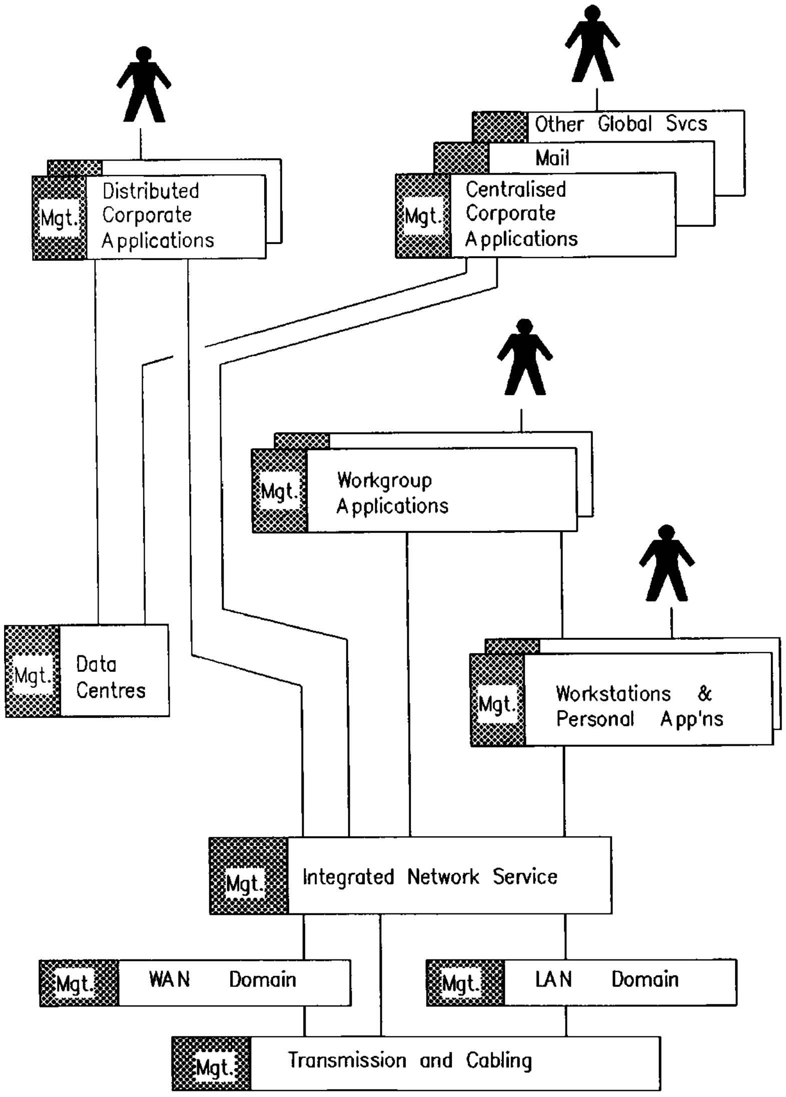


Fig. 4 The Service Domain Model

its internal affairs, and negotiates with its neighbours at the boundary in accordance with rules laid down by the international standards bodies. There is no single authority actively managing the system as a whole.

Networks run by corporate enterprises vary a great deal, but do not often show this degree of distributed management. There is usually some, and often a considerable, level of centralised control. However central management can well be a matter of setting policies and standards, with operational control being devolved to divisions or departments. As interworking between legally separate organisations increases, with the spread of EDI for example, the need for fully distributed management will increase since there will be no central point which can exercise the control.

2.3 *The Architecture of a Domain Manager*

To briefly review the material presented elsewhere in this issue, and in particular in Jenkins (1991), a Domain Manager has a dual existence within

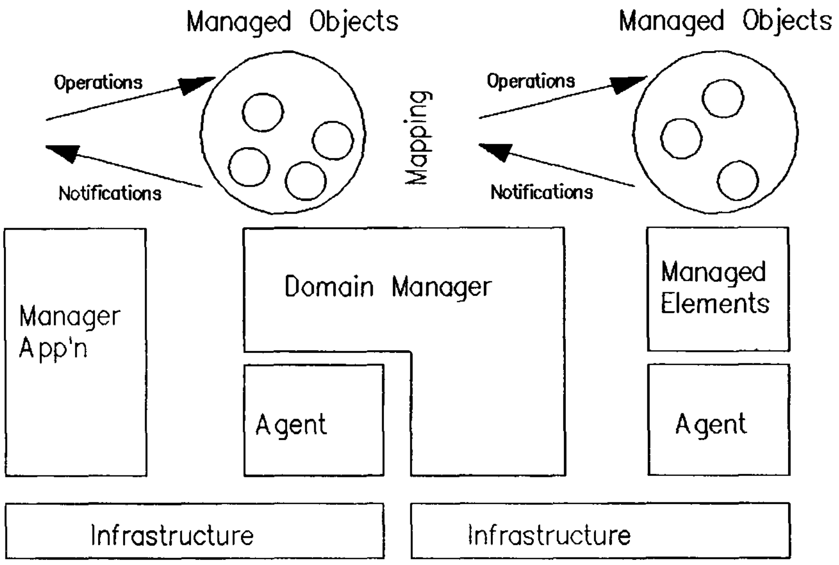


Fig. 5 The Dual Nature of a Domain Manager

the OSI Systems Management architecture. As shown in Figure 5 a Domain Manager is on the one hand a Manager (takes the Manager Rôle), talking to a set of Managed Objects and manipulating them according to the Operations and Notifications they provide. On the other hand it is a managed system (takes the Agent Rôle) providing a view of its own set of Managed Objects to one or more further Management Applications. The mapping between the two sets of Managed Objects can be simple or complex, and reflects the functionality of the Domain Manager itself.

It is quite possible for a single Domain Manager to provide simultaneous views on two or three different sets of Managed Objects, appropriate to different Management Applications, all mapped onto the one Domain of real managed elements. One view may map every real resource into a standard Managed Object model which supports fault reporting to a common Problem Management application, another may give an abstracted view suitable for Status Display or Configuration Generation, and another may give the quite different view appropriate to software distribution.

3 Styles of Integration

It was stated above that integration is a key issue in Network Management, but that simple models based on hierarchical control are inadequate to meet the full requirement. There are in fact a number of styles of integration, which are each appropriate in different circumstances, and which are needed in combination in any real, complex system. These styles are dictated by the different information structures found in the various functional areas, and in particular by the degree to which common Managed Object Models can be mapped onto the real underlying resources.

3.1 Full "Flat" Integration

There are a number of application areas which do not depend on the detailed nature of the managed element, but operate on an abstract Managed Object Model which can be mapped onto any (or a large proportion of) real managed resources. In these cases it is appropriate to collect information on all objects, at a common level of detail, and handle it in a homogeneous manner using a common application.

The best examples of such an application are probably the Help Desk and Problem Management, where the same application can be used to deal with faults reported on the photocopier or the coffee machine just as easily as those on the IT system. Other applications in this class are financial asset management, probably billing, and possibly inventory control.

3.2 Integration by Abstraction

This class includes those application areas where the Service Domain model needs to be applied to give an abstracted or Service oriented view of a set of underlying resources. A good example of this is the case of Configuration Generation, where the overall configuration details of a large and complex system are held centrally and distributed out to the various Domains in a controlled fashion. A central application maintains a model of how all the Domains interact, and what Services they deliver, but cannot practically track every low level managed element with all of its parameters. Controlling these low level details is the responsibility of a Domain Manager, which sets them up so as to meet the requirements for Services defined centrally.

Other application areas which have some commonality with Configuration Generation are automated Fault Location by correlating reports of wide-spread symptoms, and Impact Analysis of faults or proposed changes. Both of these require an abstracted view of Services delivered in order to control the level of complexity they have to deal with. A case which combines features of this category and the previous one is the display of the state of a network, using a schematic graphical display. In this case the display itself needs to be hierarchically arranged with a high level abstracted view at the top, but with the ability to drill down to the detail required in any particular area.

3.3 Use of Common Tools

In a number of cases the information from different Domains is similar in form, but there is no great value to be added by collecting or aggregating it together. In these cases there are cost benefits in using common tools to process the information, but applied independently to each Domain. Common tools will also ensure that the information is available in a common form if and when a more sophisticated application does wish to take an overall view. The Managed Object Models reflecting this situation will be similar in form or syntax, but will have no semantic relationship to each other.

Some examples which fall in this area are tools for statistics analysis and display, managing audit logs, software distribution, Service Level Agreement monitoring, change management and so on. Note that most of these tools can be applied at the Domain level, or at the System level if similar data exist with a system-wide scope.

3.4 No Integration Required

At the bottom of the scale there are a number of cases where no integration is required between Domains at all, because the Managed Object Models are so different that it is impractical. Examples are those functions performed at the level of individual components, such as detailed fault diagnosis. The work on management standards is however moving in the direction of providing common models, and hence integrated management applications, even for these functions.

4 Standards in Network Management

As indicated above the development of Network Management has not historically produced a clear consensus on the issue of standards. The reasons for this include the lack of large players able to impose de facto standards, the diversity of suppliers and technologies involved, and the fact that many networking products are small boxes where tuning for efficiency has so far been more important than conformance to standards. This pattern is now changing however, with a strong probability that the Simple Network Management Protocol standard (defined in [SNMP]) will establish itself as the preferred management protocol in the networking area, at least for a time.

SNMP was developed by the TCP/IP Internet community in response to their need to manage heterogeneous, multi-vendor networks of LANs, routers and attached hosts. The protocol definition was developed in 1988 by a working group of the Internet Engineering Task Force (IETF) under the direction of the Internet Activities Board (IAB), and is published in a series of documents termed Request For Comment (RFC) which are in effect now standards definitions. The SNMP definitions comprise three main sections:

- The Simple Network Management Protocol. This defines a set of primitive operations which may be performed by a manager or a managed system. The main capabilities provided are for a manager to read the value of an attribute (GetRequest), to alter the value of an attribute (SetRequest), or for a managed system to transmit an unsolicited event message (Trap). A GetNext operation is provided to allow sequential access to the contents of tables.
- The Structure of Management Information. This defines the common structures and language used to define all the information which may be accessed by the SNMP protocol. The language used for definition is the ISO Abstract Syntax Notation (ASN.1), but confusingly SNMP uses the term "objects" to refer to what the OSI standards would call "attributes". There is no concept in SNMP of Managed Objects as used in the OSI standards.
- The Management Information Base, now at its second revision MIB-II. This defines the set of attributes (or "objects") recognised by the standard, arranged into a tree which is used to define their names. This tree includes branches such as Internet Protocol (IP), Transmission Control Protocol (TCP), Transmission (eg. Ethernet, FDDI, etc.), the System (eg. the name, location, etc. of the managed system), and several others. It also provides for implementors to add their own proprietary extensions where necessary.

SNMP was defined by the TCP/IP community, and is most commonly implemented on products supporting these protocol standards. However SNMP only requires a datagram service and could in principle be run over a variety of lower layer services, including OSI Transport (Connectionless or any Connection Oriented Class), and Link-layer or MAC-layer protocols. The simplicity of SNMP and the ability to run over low level protocols make it an attractive choice for networking products, which often do not contain the full 7-layer protocol stack required by OSI CMIP.

Set against this economy of implementation is the fact that SNMP does not contain the functional richness of the OSI standards. At the protocol level CMIP provides further operations such as Action, Create and Delete. More importantly OSI adds the concept of Managed Objects with fully fledged inheritance and containment mechanisms. This makes for a much more comprehensive representation of the structure of a managed system, and permits powerful scoping and filtering mechanisms to be defined by which multiple objects can be handled in a single operation.

It is an open question whether in the long run the functional richness of the OSI standards, and the fact that they are underwritten by ISO, will cause the demise of SNMP, or whether the relative simplicity and economy of SNMP will ensure a continued niche for its survival. In the author's opinion SNMP will continue to evolve and adapt itself, and will continue to be used in the networking area, because it fills a need. At the higher levels represented by Systems Management functionality, the platforms used can readily support CMIP, and the richness and openness of the OSI standards will make them the preferred solution.

What is now clear is that SNMP has established a presence in the networking market, and will continue to be a significant de facto standard in this area for some time to come. In fact it is the only standard which appears capable of bringing some commonality to the world of network management in the near future.

5 The Rôle of Network Management and Systems Management

The ideas presented so far, of Domains and Domain Management functions, the different styles of integration between them, and the growth of SNMP as an industry standard, can be combined into an overall picture containing distinct Network Management and Systems Management products. This is first described in terms of a number of idealised product rôles, followed by some comments on how actual products fit onto the idealised picture. Figure 6 shows the former, and Figure 7 the latter view.

5.1 The Element Manager

The Element Manager is a unit which manages one Domain of managed elements, of a single or closely related type. It provides the physical focal point needed to maintain management connections to the devices, and provides the specific management functionality, data storage, and HCI support needed for those elements. The Element Manager will also provide the interfaces out to other management systems, and will provide a level of abstract Managed Object views, but does not itself provide any of the integrated management functions.

There are as many products fulfilling this rôle as there are types of network element, and historically they have all tended to be different, produced to match the particular networking products they are managing. Often they have been designed as stand-alone systems without well defined interfaces to external management applications. For the reasons described earlier these systems tend to concentrate on Operational Control functions and to be weaker on administrative applications.

For all the reasons described in this paper products in this area are broadening their scope and increasing their functionality, with the aim of providing greater integration. Interfaces to higher level management applications are becoming more common. Systems which are designed as generic

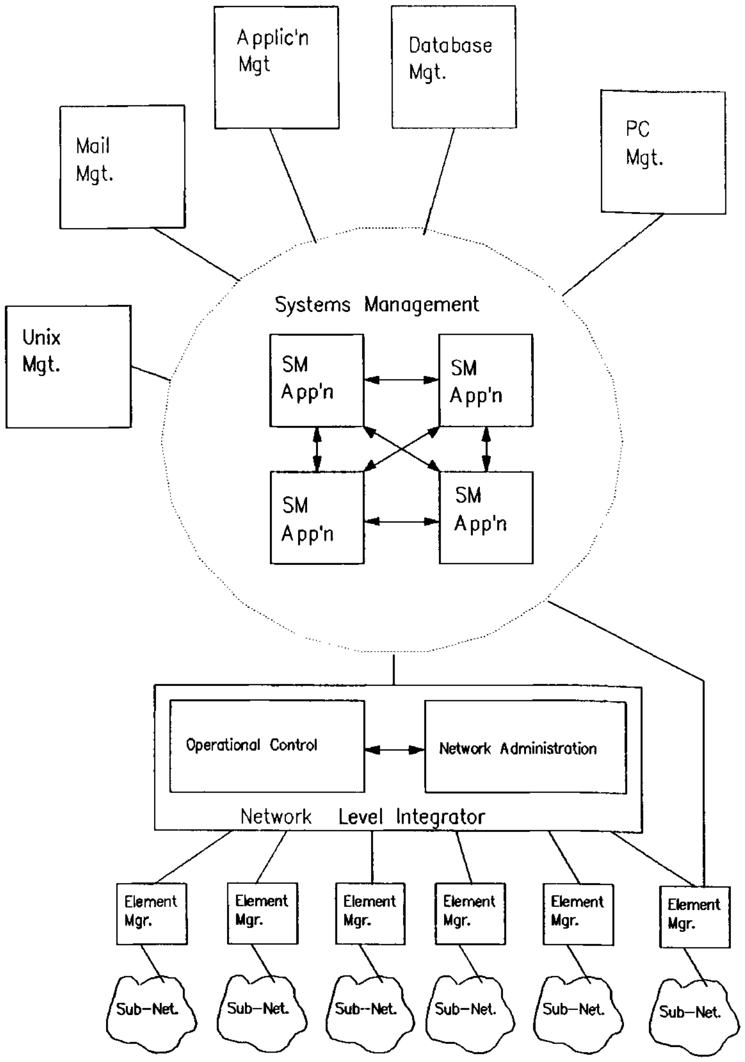


Fig. 6 Network Management and Systems Management Rôles

SNMP Managers, capable of being configured to manage any SNMP conformant device, are starting to appear.

5.2 The Network Level Integration System

Because of the degree of diversity in the network product market, and hence in the Element Managers available, there is a rôle for a system which provides integrated management of the network, but without going any further into the wider Systems Management field. This type of system is

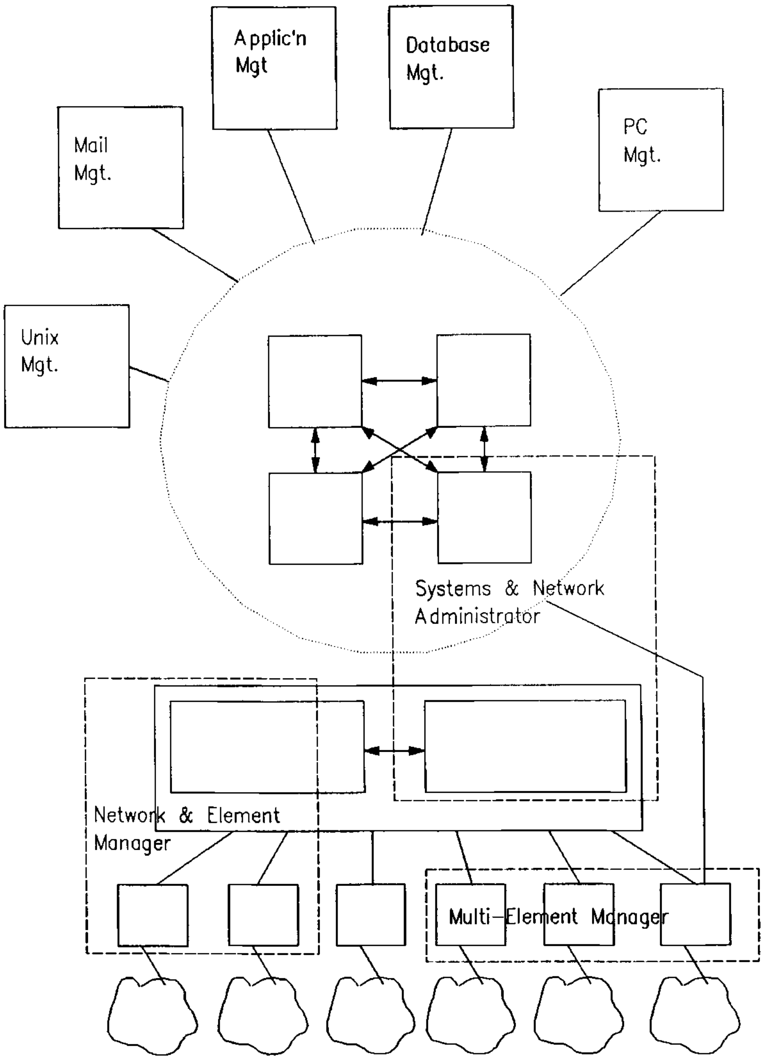


Fig. 7 Some Product Mappings

aimed primarily at the same functions as the Element Managers, ie. the area of Operational Control, but on a network wide basis. There is also increasing interest in extending its coverage into the areas such as Inventory, Asset Management, Change Management and other functions which are about managing the total cost base of the network and administering changes to it over the lifecycle.

Products in this area are coming from two different directions. Firstly there are products which exist simply to provide this level of integration, running on top of existing Element Managers. These are usually from independent suppliers, operating with a mixture of *de facto* standard and proprietary interfaces as found appropriate. Secondly there is a general trend for Element Managers to broaden their scope and encompass several Domains, typically those supplied by a single manufacturer, using proprietary protocols. However the advent of SNMP is changing this picture and is enabling both of these types of system to manage a wider variety of elements directly, and is increasingly blurring the distinction between them.

The Network Level Integration products provide a further point at which an abstract view can be derived for the benefit of Systems Management applications. Where a Network Level Integrator is present it will usually provide the route into Systems Management, but there will be cases where no separate product is provided at this level and Systems Management applications interface directly to the Element Manager.

5.3 Systems Management Integration

Historically there has been an organisational split between the management of data processing systems and the management of the network, though there is a trend towards removing this distinction. As a result the management products in the two areas have also tended to be distinct, even when their functionality is similar. It is however clear that there are benefits in integrating the two, provided the appropriate models are adopted as described above.

Following this line, Systems Management represents both a higher level integration of the management view and a fanning out or distribution of management responsibilities. The higher level integration applies across the multiple managed Domains, and the distribution because the different management applications support different Users with significantly different interests. Each of the management functions is likely to be implemented as a separate application or group of applications, possibly distributed geographically, though they will also interact with each other. Each application will be interested in a different part of the view provided by the Domain Managers (including the Network Management products), either by looking at different levels of abstraction, or by looking at different attributes or objects in the model presented.

The rôle of Systems Management is to provide those functions which are common across all management, and not specific to networks or other Domains. It thus includes the applications where "flat" integration is required such as Problem and Asset Management, those working on abstracted views such as Configuration Generation and Status Display, and the common tools needed for such things as statistics analysis.

It is clear that there is a considerable overlap between some of these applications at the Systems Management level and the same functions at the Network Integration level. While products at these two level tend to be distinct today, there will clearly be benefit in exploiting this similarity in common tools and integrated applications.

6 Conclusion

ICL's approach to Network Management is dictated by the state of the market, and our customers' requirements, and in particular by the highly diverse nature of networking and network management products. The networking requirements of each of our customers are different because each has a different history, a different installed base of equipment, and different business requirements. There are also many possible solutions which could be constructed from available technology and products. In these circumstances ICL's approach is solution-led, aiming to put the best and most cost effective solution in place to satisfy each requirement.

To do this ICL has organised itself to be able to provide a wide range of networking technology, sourced from wherever is appropriate, in order to provide quality solutions. Element Managers are generally sourced from the same vendors as the networking products, since they are by definition specific to the characteristics of that technology. Integration of management functions is achieved by a number of routes, working in concert.

- Taking advantage of vendors' offerings, and increasingly of the SNMP industry standard, to provide Element Managers capable of managing a number of Domains.
- Providing separate Network Level Integrators to provide coordinated management of all the network domains, particularly in the area of Operational Control. ICL intends to offer a graduated range of capability in this rôle, to meet the wide range of system sizes and functionality required.
- Providing links into ICL Systems Management applications from the lower level systems as appropriate. These provide the abstracted view of the network resources needed by the Systems Management applications which are working at the wider scope of the total IT system.
- Exploiting common applications wherever possible to provide administrative functions, where similar functions are required at both the Systems Management and Network Integration levels.
- Providing links into other vendors' management systems where this is required for particular solutions.

References

OSI – Common Management Information Service Definition (CMIS). ISO/IEC 9595: 1991.
OSI – Common Management Information Protocol Specification (CMIP). ISO/IEC 9596-1: 1991.

The Simple Network Management Protocol (SNMP) (authors CASE, J., FEDOR, M., SCHOFFSTALL, M. and DAVIN, J.) published electronically by Internet Activities Board, reference RFC 1157.
ICL Systems Management Architecture (SMA). ICL PSD 3442. 1990.
JENKINS, G.I. Manageability of a Distributed System, ICL Tech. J. Vol. 7 No. 4 pp. 686–701, 1991.

Biography

Tony Maynard-Smith

Tony Maynard-Smith obtained a B.A. in Physics from Cambridge University in 1966 and joined ICL to work on the development of data communications systems, being involved in developments associated with the System 4, 1900, and later the VME ranges of mainframes. From 1970 to 1985 he worked in the Company's Letchworth Development Centre which produced a wide variety of systems customised to individual clients' requirements, particularly in the fields of communications and networking. He currently works for Network Systems Technology, within ICL Secure Systems, with responsibilities for network management and involvement in the Company's systems management programme.

An Overview of the Raleigh Object-Oriented Database System

Michael H. Kay
ICL Fellow, Reading UK
Peter J. Rivett
CASE Product Centre, ICL Basingstoke, UK

Abstract

Raleigh is an object-oriented database system, incorporating a functional data model and its own computationally complete language, OODL. It is being developed initially for internal use within ICL's System Management and Application Development product programmes.

This paper provides an overview of Raleigh*, focusing on its data model and language, but also outlining the implementation architecture.

1 Introduction

It has been recognised for some years that conventional database systems are not well suited to storing the kind of complex objects that occur in design applications such as software engineering and network management. Object-oriented databases have been proposed as the solution.

The Raleigh project was set up to develop a database system for use by such applications. The first two applications are a new Open Dictionary (or repository) system (Kay, 1991) being developed as a successor to the well-established DDS product (Bourne, 1979), and a configuration generation system forming part of ICL's Open Systems Management suite of products (Gale, 1991).

Object-Oriented database technology is still young: there are no standards yet, and there have been some emotional debates about the directions the technology should take (Atkinson, 1989; Stonebraker, 1989). In designing

**Raleigh* is an internal development name, and comes from the resemblance of the original system diagram to a bicycle.

Raleigh our aim was not to take sides in this debate, nor to add to the body of research in this area, but merely to meet the requirements of the applications we were trying to develop. The result is an engineering synthesis of ideas from a variety of research sources.

This paper provides an introduction to Raleigh with emphasis on the data model and its associated language, OODL.

2 Requirements

The requirements on Raleigh were derived from the known demands of the first two application areas, but wherever possible these were generalised to ensure maximum reusability of the resulting product.

The requirements included the following:

- ability to support complex objects (such as screen layouts and network diagrams): these might include documents and graphics as well as the more conventional numbers and strings
- complete extensibility, that is, flexibility to define new types of object, preferably by refining existing types
- the ability to perform data transformations "behind the scenes": this need was particularly acute in the dictionary application, where we needed the capability to integrate tools from different suppliers that did not share a common data model
- support for retrieval using high-level (declarative) queries, not just navigation using pre-defined access paths
- support for a usage profile typified by designers working on different parts of the database, at their own workstations, independently but as part of a team with disciplined processes and quality controls. This leads to requirements for distribution, for flexible object naming, version control, and integrity control, for long transactions, process support, and access control.

Note that Raleigh is intended for use as a back-end server with the user interface being provided by the application clients.

3 The Object Model

3.1 Choice of Model

The object model we chose for Raleigh is a functional model, derived from Daplex (Shipman, 1981), with similarities to IRIS (Fishman, 1989) and EFDM (Kulkarni and Atkinson, 1986). In common with (Poulovassilis and King, 1990) we have extended the functional model to include computationally-complete methods.

We made this choice for a number of reasons.

- As with the relational model, the functional model has a solid basis in mathematics, which means that declarative queries become possible. Unlike the relational model, this includes recursive queries such as the notorious "parts-explosion", which arise very frequently in our chosen application areas. We felt that a return to navigational data manipulation languages would be a retrograde step.

- Unlike some developers, we had no strong requirement or predisposition to be compatible either with existing relational databases or with an existing programming language such as C+ + or Smalltalk. Rather than adding programming features to SQL, or database features to C++, we have attempted a harmonious synthesis of database and programming ideas. The functional model provided the right framework for this integration.

A good object model, we feel, should describe both the permissible states of the system and its permitted behaviour. Traditionally database models have been strong in describing state, and weak on behaviour; while the programming tradition has focussed on behaviour and has been weak in modelling state. The functional model, augmented with computationally-complete methods, models both with remarkable symmetry and economy of concepts.

One of the attractions of a functional model is that it is what the Object Management Group [OMG, 1990] call a *generalized* (as opposed to a classical) object model. In a classical object model, such as is found in Smalltalk or C++, every operation is "sent to" a particular object for execution. This leads to a source of arbitrariness: when a customer orders a product, is this an operation on the customer, the product, or the order? Such arbitrariness has always been anathema to database people, because once made, the decision is difficult to change. In a generalized object model, such as is found in Daplex and CLOS (Moon, 1989) it is recognised that operations can be applied symmetrically to several objects.

3.2 Overview of the Object Model

The core of the Raleigh object model can be summarised by the following definitions and axioms:

Objects
- Every distinct thing of interest is an Object.
- Each Object has distinct identity.

Entities and Literals
- There are two kinds of Object, called Literals and Entities.

- The identity of a literal is its value. Examples of literals are the number 93.7 and the string "London". Literals are neither created nor deleted, they simply exist.
- The identity of an Entity is established by its creation-event. That is, an Entity acquires a unique, non-reusable identifier when it is created, and retains this identifier until it is destroyed.
- Literals may refer to Entities. For example, if X and Y are Entities, the Set $\{X,Y\}$ is a Literal. This literal exists at all times X and Y both exist. The literal itself is not explicitly created and is not explicitly destroyed.

Classes
- Every object is an instance of exactly one Class. A Class is a description of behaviour shared by a collection of Objects.
- A Class is itself an Entity.
- In general a Class has one or more superclasses (the exception is the Class Object). Classes inherit the behaviour of their superclasses. Inheritance ambiguities are resolved by reporting an error if the situation arises.

Functions
- The behaviour and the state of Objects are expressed using *Functions.* Functions are used to represent the concepts colloquially called attributes, relationships, and operations.
- A Function is itself an Entity.
- A Function takes one or more arguments and returns a single result. The arguments and the result are all Objects.
- Functions may be implemented intensionally or extensionally, that is, either by providing an algorithmic method or by tabulating the results in storage.
- A Function may return an Aggregate as its result, in which case it may be described as a multi-valued Function. (An Aggregate is a Literal; the elements of the Aggregate may either be Literals or Entities).
- A Function may return the Literal *Null* (see section 5.4) as its result, in which case it may be described as a partial Function.
- A Function may be defined to be updatable. In the case of a Function implemented intensionally, an update method must then be supplied.
- Functions are inherited with respect to each of their arguments. For example, the Function $F(X,Y)$ is inherited so that it is applicable to any subclass of X and any subclass of Y.
- Multiple implementations of a Function may be defined for different classes of arguments. The choice of an actual implementation is made using an algorithm that takes into account the Classes of all the supplied arguments (see section 5.1 for details).
- Functions with a single entity-valued parameter may have a corresponding inverse function declared. This takes as parameter the result class of the original function and returns the associated entity or entities. (For example the function *Name(Customer)* returning a *String* may

have an inverse function *CustomerNamed(String)* returning a *Customer*).

Blocks

- When a Function is implemented intensionally, the algorithms for evaluating or updating the function are expressed as Blocks. A Block is a piece of executable code that takes parameters and returns a result. In practice, Blocks are established by compiling strings written in the OODL language.
- A Block is a Literal.
- Blocks are used not only to implement Functions, but in a variety of other contexts. For example, some Functions are defined that take a Block as an argument or that return a Block as their result.
- A Block that takes a single object as its parameter and that returns a Boolean result is known as a Predicate.

3.3 The Class Hierarchy

The core class hierarchy is shown in Figure 1: the lines represent the relationship between superclasses and subclasses.

Raleigh supports multiple inheritance, but this is little used in the core object model. The only case where it is used is for the class String, which inherits both from List and from Magnitude.

The diagram excludes those Classes which play only an incidental role in the core model, for example enumeration classes that are used solely to constrain the result of a Function in the core model.

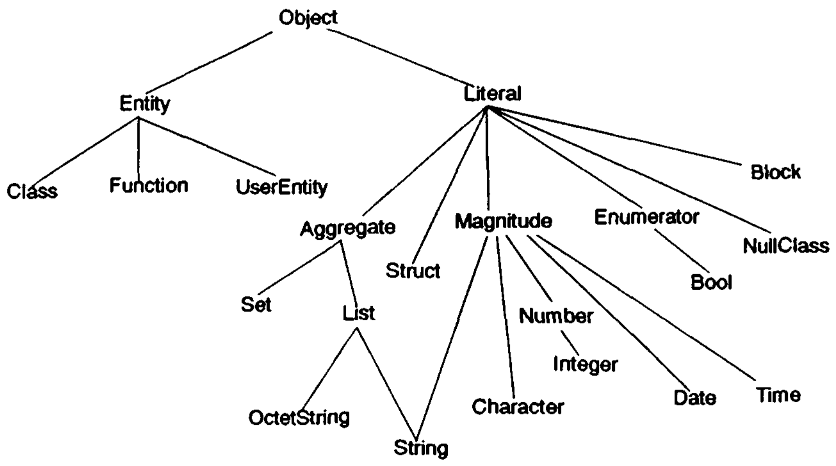


Fig. 1 The Raleigh Class Hierarchy

3.4 The Extended Model

The core Raleigh model described above includes no facilities for object naming, session management, version control, access control, distribution of data and processing, or transaction management. These features are provided by the extended model, which is beyond the scope of this paper. The extended model is based on the concept of an Activity, which is a unit of user work – similar to an extended (long, nested, multi-user) transaction. This is derived in turn from the process modelling ideas developed by the IPSE 2.5 project (Warboys, 1989).

4 The OODL language

4.1 Overview

This section gives a brief description of OODL. OODL acts as a database language, encompassing both data description and data manipulation facilities; it is also a computationally complete language in which arbitrary processing can be expressed. OODL is used to write the methods that implement intensional functions, and it also serves as the means of communication between an application and the database server, analogous to SQL in a relational system.

The database aspects of the language and the processing aspects are inseparable from each other. The same types and operators are used throughout.

The syntax of OODL is fairly sparse; most of the power of the language derives from the rich vocabulary of functions that may be called. However, some syntactic sugaring has been introduced to make commonly-written functions appear more familiar to programmers trained in languages such as C and SQL.

OODL is compiled. Names (e.g. of Functions and Classes) are translated to object identifiers at compile time. However binding to an appropriate Implementation of a Function is always done at run-time.

The unit of compilation and of execution is a Block. A Block consists of declarations of parameters, declarations of variables, and executable statements: for example the following block counts the number of spaces in a supplied string

```
{ param str; var i; i := 0;
  for each ch in str where ch=' ' do
    i := i + 1;
  endfor;
  return i
}
```

Variables and parameters in OODL act as handles for Objects; they are not themselves Objects in the sense of the Raleigh Object Model. Because new classes can be introduced dynamically, it seemed inappropriate for variables to be strongly typed at compile-time; all type-checking is therefore deferred until run-time. Variables are initialised to the value Null.

Although Raleigh uses a functional data model, we chose not to make OODL a pure functional programming language. The only reason for this was the unfamiliarity of our target users with this style of programming.

4.2 Statements

The following kinds of executable statement are defined:

Class definitions, e.g.

```
create class customer subclass of UserEntity;
```

Function definitions, e.g.

```
create function customer_name (customer) returning
String with (Updatable);
create function creditworthy (customer) returning
Bool;
```

Implementation definitions, e.g.

```
implement customer_name (customer) as storage;

implement creditworthy (customer) as
{param c; return (credit_rating(c)=="AAA")};
```

Constant declarations, e.g.

```
create name pi as 3.14159;
create name cust_name as customer_name;
```

Function calls:

```
destroy(x);
customer_name(New(customer)) := "Smith Enteprises";
```

Variable assignments:

```
a := ClassOf(b);
```

Function assignments:

```
customer_name(c):= "Smith";
```

Conditional statements

```
if creditworthy(c)
  then return "OK"
  else return "NO"
endif;
switch credit_rating(c)
  case "AAA", "AAB": return "OK";
  case "XXX": return "NO";
  default: returnnull;
endswitch;
```

Conditional iteration:

```
repeat while i<64
  i := i+1; total := total+f(i)
endrepeat
```

Iteration over an aggregate:

```
for each c in PopX(customer)
  where creditworthy(c)
  do discount(c) := Percent $ 10
endfor;
```

(PopX is a function that returns all the instances of a given class and its subclasses: see section 5.6; "$" is an operator representing the Cast function, which in this case converts the Integer 10 to an object of class *Percent*).

```
for each char in input_string where char="," do
comma_count := comma_count+1
endfor;
```

A return statement:

```
return x*x;
```

4.3 Expressions

In many contexts within statements, expressions can be used. This applies not only to the obvious contexts such as the right-hand-side of an assignment, but also to less obvious contexts: for example in a Function call the Function name may be replaced by an expression returning a Function; in an Implementation definition the Block may be replaced by an expression returning a Block.

Expressions can take the following forms:

A function call:

```
a := f(b);
```

A function call expressed in infix notation (infix operators may be defined for any dyadic Functions):

```
a := b+(c*d);
x := Percent $ 10; /* "$" invokes the Cast function */
```

The name of a variable or parameter:

```
a := b;
```

The name of an object:

```
a := pi;
b := null;
```

(The compiler first searches for a variable with a matching name, as in any block-structured language; if no suitable variable is found, it searches the database for an object with the relevant name).

A literal:

```
a := 3.14159;
b := "Jamaica";
c := [10, ["a","b"], X];
```

A call on a block:

```
a := {param x; return (x**x)};
b := a(2);
```

A select expression. A select expression returns an Aggregate, and may be used in any context where an Aggregate is required:

```
a := select name(c) from c in PopX(customer)
     where creditworthy(c)
     order by asc zipcode(c)
     endselect;
b := Count(select c in PopX(customer)
     where creditworthy(c)
     endselect);
```

4.4 Summary

The above discussion is intended to give a flavour of the language and omits a few features that cannot be concisely explained, for example invocation of superclass implementations and exception handling.

The key to the power of the language is the uniform way it handles processing of persistent data (Objects) and transient data (variables), with a uniform type system and set of functions available in both cases. In particular, Aggregates can be manipulated either by bulk operations or by element-at-a-time iteration, and this applies equally to Aggregates stored in the database (for example the population of a Class) as it does to local Aggregates such as Strings.

5 Observations on the model

5.1 Implementations and Binding

The same function may be implemented in different ways when applied to different objects. The class of an object determines which functions are applicable to it, and is also used to select the correct implementations for any function applied to the object.

The classic example of this is the function Print that can be invoked on any object, but with a different implementation for each class.

The process of determining the correct implementation to use for a specific function call (with parameters of specific classes) is called *binding*. For maximum flexibility Raleigh delays this until the function is actually invoked (called late binding). Late binding allows new classes to be introduced at any stage, with no need to recompile existing functions. For example, the function to calculate the tax payable by an employee will access the salary of the employee; it does not need to know that there are different subclasses of employee whose salaries are calculated in different ways. Late binding means that it is possible to introduce a new subclass of employee at any time, without any impact on the function that calculates tax.

If a function has a single parameter then the binding process first looks for an implementation appropriate to the class of the object supplied. If unsuccessful it looks for one for its superclass, and then so on up the class hierarchy. If none is found then the call fails.

For functions with more than one parameter the process is similar, but also uses the classes of the other parameters. In the case where there is more than one possible implementation then the earlier parameter has precedence.

For example, Integer is defined as a subclass of (real) Number. This makes it possible to define the following implementations for the Divide function:

```
implement Divide(Integer, Number) as Block1
implement Divide(Number, Integer) as Block2
```

The call *Divide(4,2)* will be evaluated using *Block1.*

Functions are not always implemented by executable code. They can also be implemented by stored data.

The data can be considered as a stored table with one column for each parameter and one for the result. For example the *And* function, which implements 3-valued Boolean logic, could be implemented as a stored table of the form:

Parameter_1	Parameter 2	Result
true	true	true
true	false	false
true	null	null
false	true	false
false	false	false
false	null	false
null	true	null
null	false	false
null	null	null

Alternatively the *And* function could be implemented using the Block:

```
{param x,y;
  if x=false then return false endif;
  if y=false then return false endif;
  if x=null then return null endif;
  if y=null then return null endif;
  return true;
}
```

Thus the details of the implementation are hidden from the caller, who need not even be aware whether the result is explicitly stored or calculated on the fly when the function is called. The details of the implementation, and in general the internals of an object, are said to be *encapsulated* by the interface to the functions on it.

5.2 Data Modelling

Raleigh's functional data model provides great semantic richness in modelling data and relationships.

Functions are used to represent the conventional notions of "attribute" and "relationship" in a uniform way: attributes are modelled as a function on an entity that returns a literal, while relationships are modelled as a function on an entity that returns another entity.

Modelling is enhanced by the fact that functions can be multivalue: that is to say that the function called with a specific parameter may return a set, or aggregate of values. An example of this is the function *Child* applied to a *Person* that returns that Person's children.

Facilities are provided in OODL for processing aggregates and also for adding or removing individual values from a multivalued result. Multivalue functions make for natural modelling of one-to-many and many-to-many relationships, and multivalued attributes.

Raleigh also provides the concept of inverse functions. The inverse of a function such as *Name(Person)* which returns a *String* is the function which when applied to a *String* returns the *Person* objects (if any) who have that *Name*. They are thus the equivalent of a traditional file inversion. Raleigh automatically provides a suitable implementation, based on any storage or code used for the primary function.

5.3 *Extensibility*

Class and function definitions are themselves objects stored in the database which can themselves be manipulated. This makes the schema very extensible and avoids the need for a separate Data Definition Language.

The power of the system is further increased by the ability to generate and manipulate code (in the form of OODL blocks) in the same way as any other data.

5.4 *Null values*

The special class *NullClass* has only a single instance, the object *null*. Null is defined so that functions can return it when no other result is applicable. A function in Raleigh always returns a result; but where no other result is applicable, this result may be the object *null*.

If *null* is supplied as an argument to a function the default action is for that function itself to return *null*. So, for example, adding *null* to an integer gives a result of *null*. There are some exceptions, however. Three-valued logic is used as in SQL, so for example *null or true* returns *true* although *null and true* returns *null*. Unlike SQL, *null* is defined to be identical to itself.

Systematic handling of null values is especially important in design applications, where data often becomes available incrementally as design proceeds. The uniform treatment of null values in both procedural code and database access is one of the benefits of using OODL rather than a conventional programming language with an embedded data sublanguage.

5.5 *Blocks as objects*

Blocks of executable OODL code are Objects with the same status as other values, and may be supplied to or returned from functions.

This provides great flexibility and power, similar to the higher-order function capabilities of functional languages such as LISP.

This allows generic Aggregate-processing functions to be defined that apply a supplied block to each element of an Aggregate object, for example:

```
A: = PopX(customer);
B: = {param c; return customer_name(c)};
C: = Map(A, B);
```

This applies B to each element of A in turn, returning a new aggregate C, in this case the collection of customer names. Another example is:

```
Restrict( orders(cust 1), {param ord; return (order_value(ord) > 500)})
```

This returns an aggregate containing all those elements in the aggregate for which the block returns the result *true*, in this case the orders for the customer valued at more than 500.

This corresponds closely to the relational *restrict* operator.

In practice, these higher-order functions will not usually be written directly, since the SQL-like *select* expression has been provided in OODL to provide a more familiar syntax.

Another use of Blocks as objects is for initialisation, for example:

```
Initialisation(customer) +: =
  {param newcust;
  credit_worthy(newcust) : = false;
  balance(newcust) : =  0}
```

This adds the supplied initialisation code to the list of such blocks to be automatically invoked when a new instance of customer is created.

5.6 Content Retrieval

One of the great advantages of the relational model is the ability to find data on the basis of any predicate, whether or not a predefined access path exists. Some object-oriented systems, in their desire to implement the "information hiding" aspects of encapsulation, have lost this ability, and can locate objects only by navigation. In Raleigh we allow objects to be retrieved on the basis of any predicate, but the predicate refers only to the (visible) functions on the object, not to its (hidden) internal content.

Frequently the user will want to search through all instances of a given class to find those that match specified criteria. To assist this, we make the extension (or "population") of a class accessible, using the function

```
PopX( Class ) returning Object with (MultiValued)
```

which returns all the objects that are instances of this class and its subclasses. This function means that every object in the database is potentially accessible, and enables a search for objects that satisfy given conditions, by applying the Restrict function.

For example, the query to find the names of all suppliers located in London would be expressed in purely functional terms as:

```
x := Map(
            Restrict(PopX(Supplier),
                        {param s; return Location(s)="London"}),
            {param s; return Name(s)}
)
```

which could also be written using the more friendly SQL-like syntax:

```
x := select Name(s) from s in PopX(Supplier)
     where Location(s)="London"
     endselect;
```

The same effect could also be achieved by declaring an inverse function.

Where possible Raleigh will use indexes to optimise such queries, although logically they perform a serial scan through the class instances.

5.7 Integrity Constraints

Constraints can be specified on the objects that belong to a class. The constraints are predicates (blocks) defined as a multivalued attribute of the class. This is expressed as a function:

```
Checks( Class ) returning Block with (Multivalued)
```

Each Block (predicate) P takes an object of class C as its parameter, and returns *true* if it is acceptable. (There is a formal difficulty here, in that P must be written to tolerate objects that are not valid objects of class C; but this does not turn out to be a problem in practice).

6 External Communication

6.1 Options available

OODL is a complete language, so in theory it could be used to write an entire application. In practice, however, the bulk of an application is likely to be written in conventional languages such as C or C++, and OODL is likely to be used only in the role of a data definition and manipulation language, including use to define constraints and transformations intimately associated with the semantics of the data.

Raleigh therefore provides the ability to make transparent use of external code and data, as well as a Service Interface that allows OODL to be embedded within applications written in other languages.

The application developer has a choice as to where various elements of the application should reside:

- in a conventional program acting as the Raleigh client;
- as functions in the Raleigh server implemented in OODL;
- as functions in the Raleigh server implemented in a conventional language.

Conventional-language programs may be used to access data stored externally to Raleigh, for example in a relational or text database. In addition Raleigh can access external files directly, by mapping them to an Object of Class *File*.

Note that the Raleigh client may be on the same machine as the server, or on a remote machine. Additionally, it will in future be possible for a Raleigh server to invoke functions stored on a remote machine.

6.2 The Raleigh Service Interface

The service interface is provided as a library of functions that can be linked (dynamically) with any application that uses "C" linking conventions. There are three main groups of functions:

- to open and close a connection to a Raleigh server. An application process may have a number of connections open simultaneously;
- to submit requests to the Raleigh server. These are expressed as strings of OODL. Requests may be synchronous or asynchronous, and results may be returned as they become available or all together at the end;
- local functions to allow data to be converted between Raleigh format and the format of the application programming language.

6.3 External implementations of Raleigh functions

In addition to implementing a Raleigh function by (OODL) code or storage, it may be implemented by an *external*. This may be a UNIX shell command or the name of a function that can be dynamically loaded from a specified library. In the latter case, the Service Interface conversion functions allow the function to interpret its parameters and set up results.

7 Raleigh Architecture

Figure 2 shows the main components of the Raleigh system.

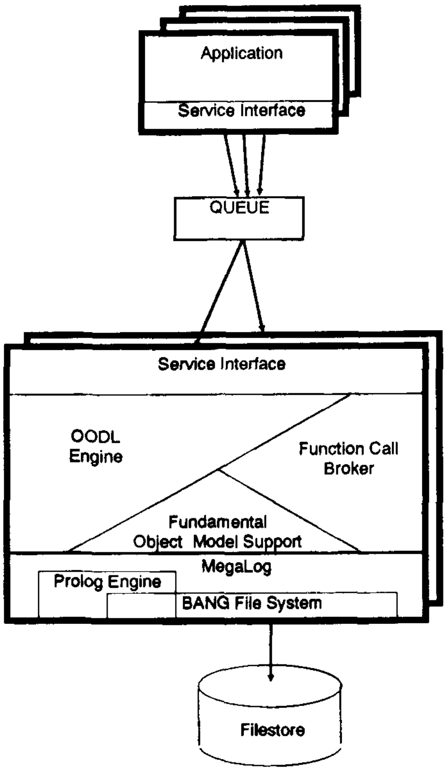


Fig. 2 Raleigh System Architecture

7.1 Service Interface

The Service Interface supports a connection between the application and the Raleigh Server. By having a Client part and a Server part it hides the communications necessary for remote server access, providing the application with location transparency.

The Service Interface also provides process decoupling between client and server: there is conceptually a single input queue on the server with a configurable number of server processes that take the next request from the queue when they are 'free'. This minimises resources tied up at the server, reduces contention, and allows performance to be tailored by changing the number of server processes.

7.2 OODL Engine

This is responsible for interpreting a block of OODL. This may have been passed across the Service Interface (in which case the OODL is first compiled) or may be the result of calling a function implemented as a block (in which case the OODL will already have been compiled). OODL is in fact semi-compiled, down to calls on a small set of Raleigh primitives. Names are resolved to object identifiers at compile-time, but binding of function calls to implementations is delayed until run-time.

7.3 Function Call Broker

This is responsible for implementing a function call, supplied with the function identifier and the parameters. It must therefore perform the (late) binding needed to determine the correct implementation of the function based on the classes of the supplied parameters, and then invoke the implementation and return the result. This may involve using the OODL engine (for implementations in OODL), calling a Raleigh primitive (for storage) or invoking an external function.

The Function Call Broker isolates the mechanism used for binding, and, in future, will be used to make requests on remote Raleigh servers to provide location transparency for function calls. The concept is modelled on the OMG Object Request Broker (OMG, 1990).

7.4 Fundamental Object Model Support

This provides primitive routines for manipulating the fundamental objects (such as classes and functions) that drive the rest of Raleigh. It provides a clean and simple interface to the underlying storage used by Raleigh, allowing optimisation and caching to be used without affecting the rest of the system.

7.5 MegaLog

MegaLog (Bocca, 1990) is a Knowledge Base Management System developed by the European Computer Industry Research Centre (ECRC) established jointly by ICL and other companies. The system provides Raleigh with both a persistent high-level implementation language based on Prolog, and an underlying datastore with the attributes required of a commercial database (for example, transaction management). For its persistent storage it uses the multi-dimensional grid file system BANG (Freeston, 1987).

The idea of basing an implementation of the functional data model on a persistent Prolog platform has also been explored by (Paton and Gray, 1990). They demonstrate how this approach can be used to support the use of rewrite rules for query optimisation.

MegaLog has proved a robust and highly productive implementation tool for Raleigh. In the longer term, we hope that the use of knowledge-base technology in the heart of the system will facilitate the development of a new generation of intelligent tools to assist in application development and system management.

8 Conclusion

Raleigh has a number of distinctive features that make it very suitable for the 'design' applications it has been aimed at, in particular those requiring the management of complex data and behaviour, a high degree of extensibility, and the ability to work in a multi-user, distributed and open environment.

At the time of writing Raleigh is about to make its first official (internal) release to the Open Dictionary and Open System Management projects. It will undoubtedly evolve as a result of feedback, and also from incorporating work in the following main areas:

- the extended model (see section 3.4)
- exception handling
- distributed data and processing
- import/export

9 Acknowledgements

The authors wish to thank the following: the other members of the Raleigh design team (David Allan, Bruce Badger, Danny Lee, Keith Simons, Tim Walters); the Raleigh Project Manager (Dick Chatterton); and members of the MegaLog team at ECRC (Jorge Bocca, Michael Dahmen, Mike Freeston).

References

ATKINSON, M. et al. The Object-Oriented Database System Manifesto. Published as Altair Rapport Technique 30-89 (August 1989).

BOCCA, J. MegaLog — A Platform for developing Knowledge Base Management Systems. ECRC KB Report #75. (1990).

BOURNE, T.J. The Data Dictionary System in Analysis and Design. ICL Tech J. Vol. 1 No. 3, pp. 292–298 (1979).

FISHMAN, D.H. et al. Overview of the IRIS DBMS. In Kim and Lochovsky (ed), Object-Oriented Concepts, Databases, and Applications. Addison-Wesley. ISBN 0–201–14410–7.

FREESTON, M. The BANG file: a new kind of grid file. Proc ACM SIGMOD Conf., San Francisco, 1987.

GALE, A.C. ICL Tech J. The Evolution within ICL of an Architecture for Systems Management ICL Tech. J. Vol. 7 No. 4, pp. 673–684, 1991.

KAY, M.H. Open Repository Technology. Proc 3rd European CASE Conference. (April 1991). ISBN 0–86353–261–6.

KULKARNI, K.G. and ATKINSON, M.P. EFDM: Extended Functional Data Model. Comp J, 29(1), pp38–46. (1986).

MOON, D.A. The COMMON LISP Object-Oriented Programming Standard. In Kim and Lochovsky (ed), Object-Oriented Concepts, Databases, and Applications. Addison-Wesley. ISBN 0-201-14410-7.

PATON, N.W. and GRAY, P.M.D. Optimising and executing DAPLEX Queries using Prolog. Comp J. 33(6) pp547-555. (1990).

POULOVASSILIS, A. and KING, P.J.H. Extending the Functional Data Model to Computational Completeness. In Advances in Database Technology - EDBT 90, ed. F. Bancilhon, C. Thanos, and D. Tsichritzis. March 1990.

SHIPMAN, D. The functional data model and the language DAPLEX. ACM TODS 6(1), pp140-173.

OBJECT MANAGEMENT GROUP (OMG). OMA Guide - ed. Richard Soley. 1990.

STONEBRAKER, M. et al. Third-Generation Database System Manifesto. 1989. Proc. IFIP TC2 Conf. on Object Oriented Databases, Windermere. (1990).

WARBOYS, B.C. The IPSE 2.5 project - a Process-Model based Architecture in "Software Engineering Environments - Research and Practice", Ellis Horwood, Chichester, 1989 ISBN 0-7458-0665-1.

Biographies

Dr. Michael H. Kay

In 1976 he gained a Ph.D from the University of Cambridge for research into database systems. He joined ICL the following year, and has specialised in database technology ever since. He led first the development unit and then the design team for the Codasyl system IDMSX; from 1983 until 1986 he was chief designer of the document retrieval system ICLFILE. He acted as Chief Architect on the INGRES programme integrating the relational database system into ICL's product range, and since 1989 he has been responsible for ICL's technical strategy for data dictionary products. The work on Raleigh has been one aspect of this.

Mike was appointed an ICL Fellow in 1989. He is a member of the team developing ICL's OPENframework architecture, with specific responsibility for Information Management. He is also a Visiting Fellow at the Institute of Software Engineering in Belfast.

Peter Rivett

Peter Rivett joined ICL in 1979 as a sponsored student, supporting VME during his industrial year. After graduating from Manchester University with a BSc (Hons) in Computing and Information Systems he joined ICL's Relational Systems Centre and played a major part in the specification and design of Querymaster 250.

He then moved onto the design and implementation of what became ICL Access, and ended up leading the team that produced the first version of the User Sponsor front end, running under Microsoft Windows.

At the start of 1990 he joined the COSMOS data dictionary project. He was seconded to the Raleigh project as its Chief Designer when it was established in collaboration with the Configuration Generation development.

OTHER PAPERS

Making a Secure Office System

Brian Moore
ICL Secure Systems, Bracknell, UK

Abstract

This paper describes the use of ICL's Secure UNIX® as the basis for the development of a major application – a Secure Office System. It describes the security policy enforced by Secure UNIX and outlines the extensions to the security policy needed for the Secure Office System. Finally some aspects of the resulting system are described, particularly the capability of the users in an office environment to cope with the restrictions imposed on them by the security policy.

1 Introduction

1.1 Security Policies

A system security policy is the part of the functional specification of a system that describes the security mechanisms to be enforced by the system and how they are to be used to achieve the required measure of security within the organisation. In addition, it describes the degree of confidence that is required to assure the managers of the organisation that the security mechanisms are being correctly enforced and not being subverted.

Security policies of existing secure systems have been based on models published by the US Department of Defense [1], the UK Government Communications Electronics Security Group (CESG) [2], [3] and, of late, the joint security authorities of France, Germany, Holland and the UK who have published the Information Technology Security Evaluation Criteria (ITSEC) [4] which is set to supersede all other standards in Europe. All of these standards have in mind the evaluation of systems and products by independent certifiers, so as to prove their security functionality, correctness and effectiveness. In the UK, a practical scheme for the evaluation of products aimed at both the commercial and government sectors has been established by the DTI in conjunction with CESG [5].

®UNIX is a registered trademark of Unix System Laboratories Inc. in the USA and other countries

A useful summary and commentary on security policies, particularly the Department of Defense B-levels, can be found in [6], which describes the VME High Security Option.

1.2 Secure UNIX

ICL has developed a Secure UNIX Operating System conforming to the United States Department of Defense TCSEC B1 assurance level and having security functions up to approximately the B3 level. In terms of ITSEC the Secure UNIX would be offered for evaluation as an F5, E3 product and is being evaluated against similar criteria specified by CESG.

Within ICL this UNIX variant is known as B1+, but for this paper it will be referred to as Secure UNIX, and the part that provides the security features will be called the UNIX Trusted Computing Base or TCB.

ICL's Secure UNIX product has been developed for use in many organisations including military, government and the financial business. It is partly for this reason that the security functionality above B1 has been added; the other reasons are the specific requirements of the Ministry of Defence. Secure UNIX is conformant to the X/Open Portability Guide. The security features have been designed so far as possible to conform to the eventual POSIX standard. It incorporates communications protocols conforming to Open Systems Interconnection standards.

1.3 Customisation

Although Secure UNIX can be configured for immediate use and will provide an environment for applications programs equivalent to an insecure X/Open conformant UNIX, it is not expected that it will be used in this way. The security policy of any practical system is likely to deviate to a greater or lesser extent from that offered by the TCB as it stands. An essential feature of the TCB therefore is that it allows for customisation. It does this by providing for the replacement of some components such as the trusted shell, and by providing for the construction of trusted programs that can exercise privileges that enable them to circumvent some TCB mechanisms.

1.4 A Secure Office System

The first major application to be based on Secure UNIX is a Secure Office System. This product has been developed by ICL as part of its Secure Electronic Office programme. It is based on Officepower, which is ICL's openly available office system. The first customer to receive it is the UK Ministry of Defence, where it has been installed for the CHOTS Project. CHOTS is the Corporate Headquarters Office Technology System which will ultimately serve around 20,000 users on a nationwide network.

The development of the Secure Office System has involved extending the UNIX TCB to meet the requirements for office functions; the security policy for the Secure Office System is a natural extension of the Secure UNIX security policy, taking into account the additional functionality and the need to be able to sell the system as an off-the-shelf product.

2 Trusted Computing Base

Secure UNIX embodies a Trusted Computing Base that carries out the functions of user authentication, control of access to objects such as files and peripherals, and the recording of security relevant events in an audit log. For the Secure Office System, various software configuration choices made available by the UNIX TCB have been made so as to meet the requirements of Secure Office security policy. The TCB has also been extended by incorporating trusted processes that take advantage of the privilege facilities in the UNIX TCB.

In addition to the usual dialogues between the user and the operating system, secure systems must provide for dialogues between the user and the TCB. These are frequently most sensitive; examples are the identification of the user, and permitting a user to change security information related to a file. For such purposes it is essential that the user cannot be fooled into conducting a dialogue with an untrustworthy part of the system. that is to say, any component other than the TCB itself. For this purpose, the UNIX TCB supports a *trusted communications path* between itself and users that is unmistakably signalled to the user by a flag displayed on the VDU screen in an area that is engineered so as to be only accessible to the TCB.

For the Secure Office System, the terminal connection is engineered so that the terminal is on Trusted Path when it is switched on. A restriction has been applied in that trusted applications are entered only from the Trusted Path. Untrusted applications cannot call trusted applications, they can only exit back to the TCB. This means that users can safely return to the Trusted Path at any time by requesting an application to exit.

3 Authentication

The UNIX TCB provides a procedure for user identification and authentication that includes password management and encryption. It also provides, however, for the replacement of this procedure by a customer-specific procedure that may, for example, use special hardware.

For the Secure Office System ICL has developed a terminal incorporating several security features including an integral badge reader. The UNIX TCB user identification and authentication procedure has been replaced by a procedure that authenticates a user's claimed identity by requiring him to input his security badge and his password. Passwords are generated by the

Secure Office System and are encrypted using algorithms chosen by the customer.

4 Mandatory Access Control

4.1 Labels

All subjects (users) and objects (files) controlled by the UNIX TCB carry a sensitivity label consisting of a hierarchical classification (an ordered set of 128 levels) and up to 32 categories and up to 64 caveats. Categories are typically codewords or project names, and, in NATO usage, caveats are used to distinguish nationalities. An example of a sensitivity label is: CONFIDENTIAL:RAGAMUFFIN:UK. In addition to sensitivity labels objects may have up to 32 informational labels for information such as Medical-in-Confidence.

In the Secure Office System the label of an object currently accessed is always displayed in the protected TCB area on the user's terminal. For military use the labels map on to familiar classifications (Unclassified, Restricted, Confidential or Secret), nationality caveats and codewords. The printed form of the label in the system as constructed for military use adopts Ministry of Defence standards.

At any instant a user also has an associated sensitivity label similar to those carried by objects and chosen by the user from the range permitted to him. This range is set within the user's profile by the System Security Officer. The label associated with the user at a particular moment is known as the user's clearance and this is also displayed in the TCB area on the terminal so that the user always knows his clearance.

4.2 Mandatory Access Rules

Mandatory Access Control permits a user to read an object only if his clearance is higher than or equal to the object's sensitivity label. This rule is known as NO READ UP. In detail, it means that the user's hierarchical classification must be greater than or equal to the object's hierarchical classification and that his caveats must be a subset of the object's caveats (the user must have at least one of the object's caveats) and that the objects' categories must be a subset of the user's categories (the user must have at least all of the object's categories). Informational labels are not taken into account. This rule is illustrated in Figure 1.

Mandatory Access Control puts on each object created by the user a label that is the same as the user's clearance. This is a restricted enforcement of the rule known as NO WRITE DOWN which requires that the label of the object must be higher than or equal to the clearance of the user. This rule prevents a user from copying information from, say, a Secret file into an

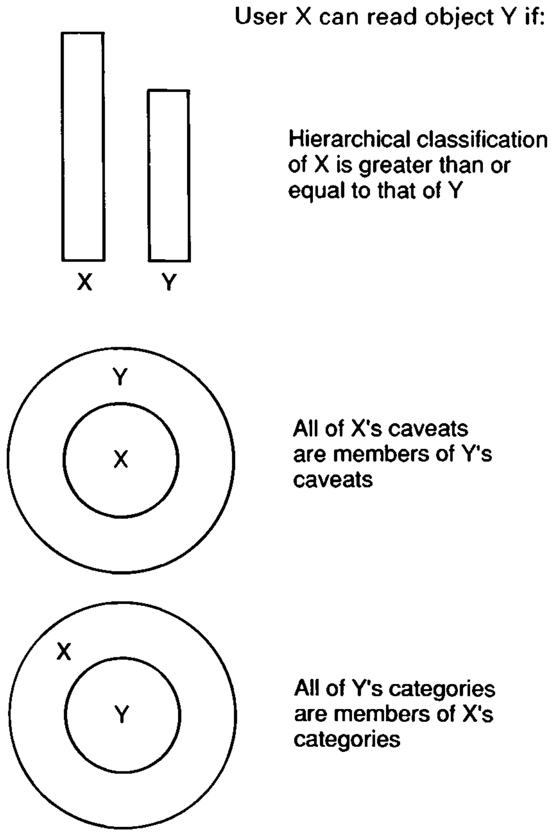

User X can read object Y if:
X
Y
Hierarchical classification of X is greater than or equal to that of Y
Y
X
All of X's caveats are members of Y's caveats
X
Y
All of Y's categories are members of X's categories

Fig. 1 Illustrating the NO READ UP rule.

Unclassified file. The TCB *does* provide a downgrading mechanism, but its use is audited and is not the prerogative of users in general.

The purpose of MAC is to ensure that wherever information is stored or printed, or by whatever route it flows through or out of the system, its sensitivity label unfailingly indicates the sensitivity of the information in it according to where the information has come from. Once an object has been labelled Confidential, all copies of it or objects containing extracts from it will automatically be labelled Confidential. For the Secure Office System, the UNIX TCB MAC facilities are used directly to implement the requirements of the system security policy.

Of all of the features of the ITSEC F3 Security Policy, it is Mandatory Access Control, and particularly the NO WRITE DOWN rule, that is likely to cause inconvenience for users. Commercial versions of the Secure Office System will alleviate the MAC rules or omit MAC altogether.

5 Discretionary Access Control

5.1 Users and Roles

Discretionary Access Control (DAC) is concerned with the explicit access rights of individual users to individual objects. For this purpose, it becomes necessary to make a distinction between the person and the job that he is carrying out. It is common in most organisations for a given job to be carried out by several people, some of who may do different jobs within the organisation from time to time. Access rights needed to do a job must therefore be associated with the job rather than the individual users enabled to do it. For example, several people may have the job of System Administrator and when one of them is acting in that role the system needs to identify the *role* for the purpose of access control (while still identifying the *actual person* for the purpose of accounting). The UNIX TCB therefore manages the relationship of the users to the roles they may assume, and then the access of these roles to objects. In general one user may select from several roles permitted him by the TCB, and one role may be assumed by many users.

5.2 Access Control Lists

In addition to sensitivity labels, objects carry *Access Control Lists* that define the roles and the *Access Control Groups* that are permitted to access them. Access Control Groups are simply lists of roles created by the security administrator. At any instant a user is working within a role and an Access Control Group, both of which are chosen by him from one of several permitted to him by the TCB.

Discretionary Access Control operates in addition to Mandatory Access Control. It permits a role to read or write an object only if that role's entry or group entry in the object's Access Control List indicates that he may read or write it. All operations concerned with Discretionary Access Control, such as the amendment of Access Control Lists, are on the Trusted Path. Applications Programs are not permitted to alter access control information.

5.3 DAC in the Secure Office System

The security policy defined for the Secure Office System requires that a user's access to the system functions is limited at any time by the role that he is performing. Thus a clerk may not have the same capabilities as a manager and a system administrator will only have available to him the functions needed to carry out that role. The system therefore associates with each role the total set of functions permitted to it and a user in that role can only select from those functions.

The security policy also requires that the original authors of objects can stipulate additional access controls to be applied to those objects that they issue to other users. These include control of the copying and printing of objects. This requirement cannot be met directly by the UNIX TCB because it must permit untrusted applications to read and write to objects and these permissions transcend finer controls such as permission to copy. To meet this requirement therefore use is made of another powerful facility provided for extending the TCB, the attachment of *Certification Indicators* to programs and to objects. This facility permits the construction of security domains in which only the *indicated programs* can operate on the *indicated objects*. Such a domain has been defined for objects to which only restricted access is allowed. Objects may be transferred into this domain and are then identified by a Certification Indicator and are operated on only by programs carrying this Indicator. These programs then perform the restricted operations in accordance with extended access permissions stored by the TCB.

It should be noted that although it can be configured for different customers to include various options and combinations of applications, the Secure Office System in its operational form is finite; it is not possible for users to add to the functionality of the operational system, which, accordingly, contains no facilities for constructing executable code. The relationship of this finite set of functions to roles and to objects is determined when the system is configured.

There is thus a many-many relationship between users, roles, functions and objects as illustrated in Figure 2.

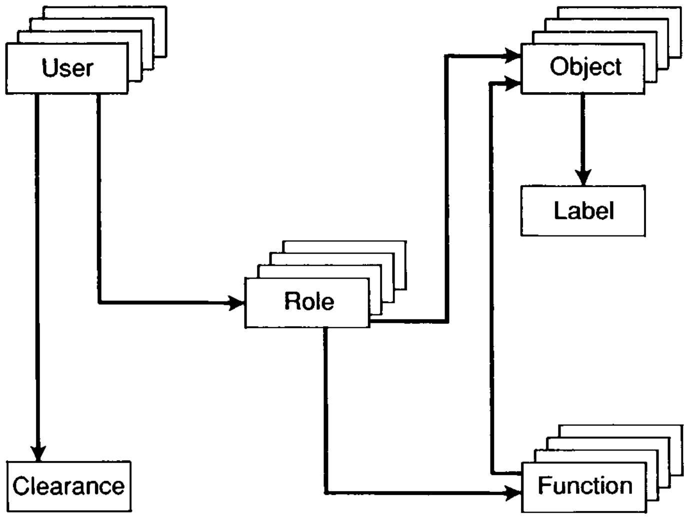


Fig. 2 Relations of users, roles, functions and objects

6 Session Management

A *session* is the total interaction between a user and the system between logging on and logging off. In office systems it is usually structured as a set of *subsessions* each of which supports a dialogue with a particular application for a particular purpose – typing a particular document, updating a particular spreadsheet. In the Secure Office System, the management of subsessions embodies some key security functions.

Officepower already contains session management facilities that permit a user to suspend the current subsession and to start another, utilising the underlying UNIX facilities for process management. For the Secure Office System these facilities have been redesigned so that users can create subsessions with different roles and clearances. The Session Management shell replaces the UNIX shell and it is executed on the Trusted Path.

After a user has successfully logged on he has established a session associated with his audit-id. He also establishes a subsession which, by default is associated with his normal role, Access Control Group, and default clearance, usually the lowest at which he can work. The user can now, however, create as many subsessions as he wishes, each having roles, Access Control Groups and clearances selected by the user and constrained by his User Profile which is defined and maintained by the System Security Officer.

Moving from one subsession to another with different security parameters needs to be simple and reasonably rapid. This feature of secure systems is a major factor in their acceptability to users. In the Secure Office System, if, for example, a user is writing to a Confidential document and needs to refer to a Restricted database he needs only five key depressions to suspend out of his Confidential subsession and enter the Restricted subsession, and a similar five key depressions to return after making the reference.

7 Accountability and Auditing

The UNIX TCB uniquely identifies each user of the system; the individual user is identified and authenticated at the beginning of each session and the user's identity, known as his *audit-id*, is associated with that session. The TCB maintains an audit log in which it records events related to user activities such as logging on, access to files and changing sensitivity labels, and related to administrative events such as changes to user profiles. For each audited event, the user's audit-id is recorded together with other information such as the date and time, names of objects to which access has been attempted, and the type of event.

The audit administrator is able to set filters to restrict the audited events according to, for example, individual users, particular files or terminals, the classification of objects concerned or the time of day.

The UNIX TCB provides facilties for other trusted programs to add records to the audit log. Using this facility in the Secure Office System many new auditable events have been defined, related to the actions of trusted applications such as the print spooler and the electronic mail system.

The Secure Office System security policy requires that auditable events shall have an associated *alarm level* and when this is reached an *alarm message* shall be sent immediately to the System Security Officer. This requirement is met by a trusted monitor which reads the audit log as it is written and when the alarm threshold is reached sends messages to nominated roles through the electronic mail.

8 Managing Security

8.1 Trusted Facilities Management

A crucial component of a secure system is the set of facilities that are used to manage the security control data itself: user names, audit-ids, clearances, labels and all of the mappings shown in Figure 2. This set is known as Trusted Facilities Management.

8.2 UNIX TCB Facilities

The UNIX TCB provides the necessary facilities for managing the security data related to users and other system objects such as terminals and communications channels. It allows for the division of these facilities between several roles; for example, the roles of Operator, System Administrator and System Security Officer can be separated and it is possible to ensure that even if any of the administrative roles were to breach the security policy they cannot prevent the recording of this in the audit log.

The Trusted Facilities Management procedures in the UNIX TCB are intended to be used by a trusted program written for each system which provides an interface for administrators consistent with the man-machine interface on that system.

8.3 Trusted Facilities Management in the Secure Office System

The primary purpose of the Trusted Facilities Management software in the Secure Office System is to provide a convenient interface for administrators in the style of the Officepower user interface. This software is, of course, all trusted and is executed on the Trusted Path.

The Trusted Facilities Manageent functions are the most sensitive in the system and are protected by the basic DAC mechanisms and especially the division between roles. To afford further protection, provision has been made to make any of these functions subject to two-person control. This means that two users, X and Y, are required to carry out the function at the same time. X, the user in the active role, initiates the function in the usual manner but the transaction will not take place unless Y is also logged-in in the corresponding passive role. The transaction is displayed on Y's terminal and finally takes place only after Y has indicated approval. An example of such a function in the Secure Office System is changing the threshold level at which an audited event would cause an alarm to be raised, which requires action by both the System Security Officer and the Audit Administrator. Another example is the downgrading (reducing the classification) of objects, in which case one role is the owner of the object and the other is the System Security Officer, and again these roles have to be adopted by different users.

9 Developing the Secure Office System

9.1 Software Architecture

The previous paragraphs have dealt with customising the Secure UNIX security policy as necessary to make a practical office system. In addition to this, ICL Product Operations has carried out any modifications needed to make the standard Officepower applications work to the TCB interface. The extent of such modifications varies considerably depending on whether the application has to be trusted or not. The system contains both trusted and untrusted applications in a structure illustrated in Figure 3.

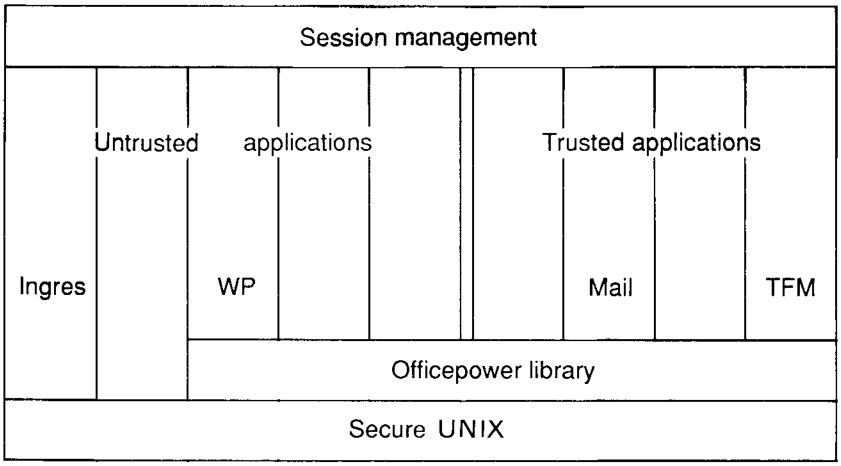


Fig. 3 The *software* structure of a secure office system

9.2 Untrusted Applications

It would be possible to construct a secure system consisting solely of untrusted applications which can be loaded and run as in any other UNIX system. Indeed, it is the implicit objective of Secure UNIX that most of the software within a system will consist of untrusted applications programs executed under the full control of the TCB. The principal untrusted Officepower applications include word processing, spreadsheet, personal database and time management. These have only been modified where necessary so that they work in the Secure UNIX environment. The system also includes the INGRES relational database which is integrated in the same way and so, for the time being, only handles data of a single classification.

9.3 Trusted Applications

There are, however, important applications that perform work simultaneously on behalf of many users and which need to handle objects at any classification. Examples from the Secure Office System are the print spooler and the X.400 electronic mail system. In addition Officepower applications use a library for the control of function menus, soft keys and other features of the user interface. This library is now used by trusted applications as well, so some parts of the code need to be examined and tested as part of the security evaluation of the TCB.

9.4 Privileges

Secure UNIX offers a range of Privilege facilities designed to enable system designers to include trusted applications within systems without having to incorporate the entire application within the TCB. Each privilege is related to a function of the TCB; if a privilege is set for a process, that process can use the TCB function which may mean, for example, that access control may be overridden. Sets of privileges can be assigned to roles and to programs which can then select which privileges available to them will be active at any instant. The objective of the designer is to use the least privilege required for the program to carry out its function.

The multi-user, multi-level components of Officepowier have therefore been modified so that they only rely on specific privileges and only activate these privileges while they are needed. In this way it has been possible to render a substantial amount of program evaluatable to the B1 (ITSEC E3) assurance level.

10 The User View

10.1 Changing from a Non-secure to a Secure System

The Secure Office System was first introduced to a population of (CHOTS) users who were already using Officepower very effectively and who had a very strong allegiance to their system. This had an advantage – the users were willing to endure some teething troubles – and a disadvantage – the users were so dependent on their system that any impediments introduced in the secure system could be unendurable. An example of the sort of pitfall that awaits the implementor of a secure system is afforded by problems that were encountered with the printing of security labels.

The Secure Office System security policy requires that all printed objects must have the sensitivity label printed at the top and bottom of every page. These labels are put on by the Trusted Print Spooler and occupy six lines on each page. The usable page length is therefore reduced by this amount and all documents that existed prior to the introduction of the secure system have to be repaginated. This was accepted, but unfortunately many users relied on pre-printed stationery so that, although they repaginated their documents, they found that the security label was printed over the letterhead.

The importance of this illustration lies in the realisation that the system design was correct and the problem could not be eliminated by changing it; the conflict was directly between the security policy and the existing working practice. In the end the problem was reduced to a managable size through the introduction of another facility that enables all logos and letterheads to be printed together with the text, incidentally solving many of the other problems associated with the use of pre-printed stationery.

When the Secure Office System was introduced we encountered several problems resulting from clashes between the security policy and the convenience of users. None of these proved to be insurmountable and the users were able to live with their secure system and are now comfortable with it. Nevertheless, there are lessons to be drawn from these problems and some of the interesting examples are outlined briefly below.

10.2 Using UNIX Directories

Users of Officepower generally make great use of the directory hierarchy to organise their files. In Secure UNIX directories are ordinary objects controlled by the TCB. They are owned by roles and are subject to MAC and DAC controls like other files. When a Secure Office application opens a file, it needs read access to all of the file's superior directories; when it creates a new file it will need write permission to access the superior directory. What then, should be the relationship of the file's label to that of the directory's label?

The MAC write equal rule leaves no latitude when an object is created, since the process must write to both the object and its directory they must both have the same label as the process. This apparently implies that all of the objects within a directory shall have the same label. The practical unacceptability of this restriction was confirmed by the first users of the system. The solution has been to introduce a trusted function that permits objects to be moved between directories regardless of their relative labels. This is illustrated in Figure 4. The effect is that although users have to create additional directories of the correct classification for the purpose of creating or copying

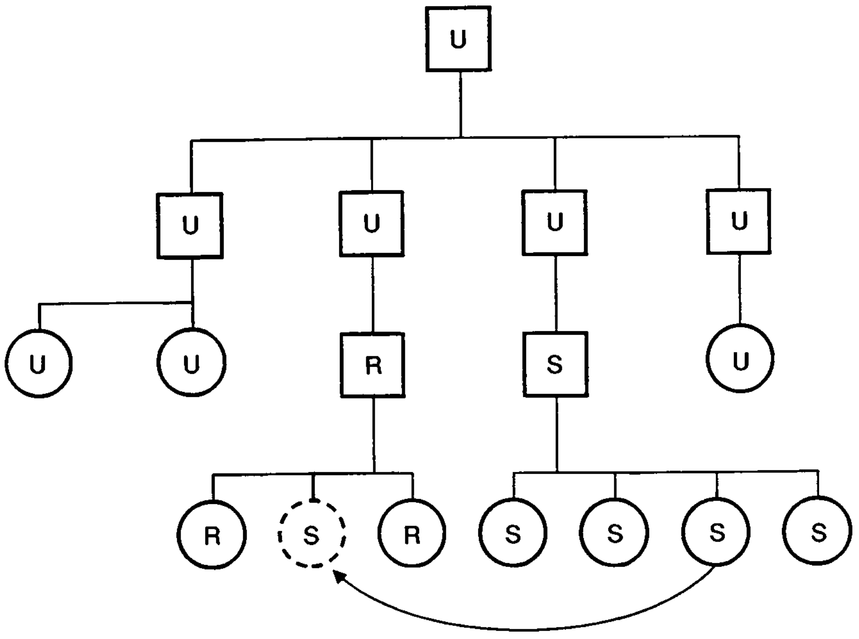


Fig. 4 Classification of directories □ and objects ○ showing movement of secret object Ⓢ into a restricted directory $\boxed{R}$

objects with that classification (and, incidentally, for extracting mail attachments with that classification), they can, with a little effort, organise their files into directories just as they would in an insecure system.

10.3 Navigation

Users are dealing with objects of different classifications in rapid succession throughout the working day. For example, mail of various classifications arrives randomly and in order to read each item the user needs to be at the corresponding clearance. The constraints of Mandatory Access Control present a major obstacle to smooth working practices and can easily lead to user frustration and resistance to the system. The important design features that mitigate the effects of Mandatory Access Control and enable users to overcome this obstacle are referred to as navigation and visibility.

Navigation means the ability to work at the correct clearance for the job in hand and hence to change the clearance and the other parameters of a session in a rapid and precise manner. It implies the ability to establish subsessions at several clearances and to be able to move rapidly between them. The Session Management facilities that provide for this have been described above. When the system was first put into live use it immediately became apparent that the effort put in to this aspect of the design was well spent. Users have indicated some improvements, such as macros that would enable them to set up and enter subsessions with commonly used parameters, but on the whole users have been able to adjust to the facilities provided.

10.4 Visibility

A severe problem in the use of multi-level secure systems is that a user has a different view of his total data depending on the clearance of the subsesison he is in. He only obtains a complete view when he is at his highest possible clearance. Many security policies stipulate that a user should not be permitted to become aware of the existence of objects that he is not, at that instant, able to access. The reason for this is that this awareness is a potential covert channel. In a finite system, however, where new programs cannot be introduced, the threat from this type of covert channel is very low. If the rule is rigorously enforced the user may be given misinformation about data to which, in general, he has right of access. The principle adopted for the Secure Office System is to allow users to be aware of the existence of all objects to which they could have access if working their highest possible clearance. Given the design of the Secure Office System this amounts to letting users see the full contents of any multi-level directories to which they have access. This feature of the system has proved very valuable in enabling users to organise their files and to minimise the inconvenience caused by Mandatory Access Control.

11 Conclusion

To make a practical secure system such as the Secure Office System demands design and development work comparable to that of making Secure UNIX itself. To serve as a basis for the development of secure systems Secure UNIX is perhaps more flexible and configurable than most insecure operating systems and it demands that certain trusted components should be modified and augmented by the secure system builder.

In all but the simplest secure systems there will be multi-user, multi-level components. These may be based on existing non-secure products but their design and development for inclusion in the secure system is never trivial. Comprehensive facilities in Secure UNIX for the support of multi-level applications is essential.

In designing a secure system there is an ever-present conflict between security and usability. Security features seldom contribute to making the system easier to use and can, if poorly designed, make the system so unfriendly that users will reject it or will deliberately flout the security policy (by treating all of their objects as unclassified for example).

Even when designed with a view to the best compromise between usability and security, the effects of security features are hard to predict and may even vary considerably between different groups of users on the same system.

It is therefore essential to monitor very closely the early experience on the live system and to be prepared to take corrective action. A secure system has little value if its users are not able to do their work efficiently. In the case of the Secure Office System, it was found that if it was possible to adapt working practices to suit the security policy users were ready to do so, but in some cases it was necessary for the system designers to make the compromise.

Acknowledgement

UNIX is a trademark of Unix System Laboratories Inc. in the USA and other countries.

References

1 Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, US Department of Defense, Washington DC, December 1985.
2 CESG Computer Security Memorandum No.2 Handbook of Computer Security Evaluation, Issue 2.0, CESG, GCHQ, Cheltenham, November 1989.
3 CESG Computer Security Memorandum No.5 System Security Policies, Issue 1.2, CESG, GCHQ, Cheltenham, September 1989.
4 Information Technology security Evaluation Criteria (ITSEC), Version 1, Der Bundesminister des Innern, Bonn, 2 May 1990.
5 UK IT Security Evaluation and Certification Scheme (UKSP 01), Department of Trade and Industry, London, 1 March 1991.
6 Parker, Tom, The VME High Security Option, ICL Tech. J. Vol. 6 No. 4, p. 657, 1989.

Biography

Brian Moore

After taking a first class BSc degree in Mathematics and Physics at the University of London, Brian Moore joined ICl in 1960 and worked as a programmer, a designer of operating systems and processor architect. He designed the Orion operating system and then specified the kernel of the George 3 operating system and its file system, designed the 2900 Processor architecture and finally designed the VME kernel.

Between 1973 and 1983 he worked for Logica and also as a freelance Computer Management Consultant. In this period he was responsible for design studies for major networks and for strategic studies and procurement.

Rejoining ICL in 1984, he was design authority for the OPCON message handling system from then to 1987, when he became design manager for the CHOTS project. This involved, first, agreeing the security policy with MOD, in 1988 defining the system and security architecture for the phase 2 prototypes, next monitoring the design and development to ensure conformance to the specified requirements and, most recently, defining the system and security architecture for ICL's proposal for the phase 3 CHOTS system.

Architectures of Knowledge Base Machines

Kam-Fai Wong
European Computer-Industry Research Centre (ECRC) GmbH Arabellastrasse 17 8000 Munich 81 Germany

Abstract

Ordinary database systems based on the relational model are widely used in many application areas. However, relational databases are unsuitable for advanced applications (e.g. Computer Aided Design CAD). This is mainly due to the limited modelling power and the lack of inferential ability in the relational model. To overcome these problems, advanced database models – commonly referred to as knowledge bases – are introduced. Due to the complexity of the knowledge base models, knowledge base systems cannot run efficiently on conventional computers. This predicament gets worse as the size of the data sets handled by such systems increases. Special purpose hardware machines have been proposed for handling knowledge bases efficiently. In this document, the architectures of five knowledge base machines are described; they are the Knowledge Base Machine (KBM, Japan), the Delta Driven Computer (DDC, France), the CLAuse Retrieval Engine (CLARE, UK), the PRISMA machine (Holland) and the European Declarative System, EDS, (Europe).

1 Introduction

In an earlier report [19] a comprehensive review of existing database machines is given; the machines described therein are mainly designed for conventional relational database applications. The relational database model is most popular compared with the other database models – network and hierarchy. It is widely used for many commercial and industrial applications. The major reason for the widespread acceptance of relational databases is simplicity – the clean and uniform approach of the relational model, the ease of implementation of the model and the simplicity of relational query languages. Nevertheless, relational database systems are not without disadvantages. Their rigid and simple semantics renders them inefficient for running complex database applications (e.g. CAD, image database, etc.). To overcome this problem, advanced database systems – such as deductive databases and object-oriented databases, generally known as knowledge-base systems – are introduced. Knowledge-base systems support complex data structures (e.g. objects in object-oriented systems) and inference rules (e.g. rules in deductive database systems). Complex data structures exhibit more representational power. Physical objects can be modelled by such data structures more naturally (e.g. representing a gate in a CAD system as an object). This makes object manipulation simpler. Furthermore, a rich set of primitives for processing these complex data structures is provided in knowledge base systems. The need for storing semantic data and deduction ability in database systems is growing rapidly. The usefulness of knowledge base systems is becoming evident. Following this trend, knowledge-base systems will be used commonly in the future.

Currently, two practical problems are undermining the usefulness of knowledge-base systems. First, the processing power required for knowledge-base systems grows with the complexity of the systems; in fact more so than with the conventional database systems. Knowledge-base systems must treat all data structures uniformly. For example, there should be no noticeable difference in performance between the retrieval of tuples and rules in a deductive knowledge-base system. Conventional computers are unable to cope with this demand efficiently. Second, the problem of processing power requirement is worsened by the increasing volumes of knowledge and data that need to be managed by knowledge-base systems. This problem is not new to the database community; however, because of the introduction of complex data structures and variables, processing a large knowledged-base is more time-consuming and complicated than processing a conventional database of the same size. At present, due to their inability to handle large sets of knowledge/data, existing knowledge-base systems are limited to small applications only.

To overcome the above problems dedicated knowledge-base machines have been proposed. In this document, the architectures of several such machines (or advanced database machines) are reported. According to how they integrate with the host computer systems, these knowledge-base machines are classified as follows:

- *Filters* – Devices which are attached between secondary and main memory and function at *operation level* – they accept and execute knowledge-base operations, e.g. retrieve, issued by the host. Generally, filters are 'slave' devices; they are programmed by the host before an operation is initiated. They perform the required operations on the data as they stream from the secondary storage devices to the main memory. In order to achieve real time (or 'on-the-fly') performance, only simple operations are usually supported.
- *Backend Machines* – Dedicated machines which function at query level – they accept and execute a work unit (or operation graph) generated by the host upon receiving a user query. These machines achieve *intra-query parallelism* by executing a knowledge-base query in parallel.

- *Dedicated Server* – Standalone computers with full KBMS capability, functioning at *transaction level* – they accept and execute database transactions (a series of queries). These machines are most complex and achieve both *intra-* and *inter-query parallelism.*

Section summary: In the following sections, the architectures of five knowledge-base machines are described: the Knowledge Base Machine (KBM, Japan), the Delta Driven Computer (DDC, France), the CLAuse Retrieval Engine (CLARE, UK), PRISMA (Holland) and the European Declarative System (EDS, Europe). These are the main knowledge-base machine research projects whose work has been widely reported. Descriptions of these knowledge-base machines concentrate on the execution sequence, the computation model, the hardware organisation and the current status of the systems and do not address design and implementation details.

2 The Knowledge-Base Machine (KBM)

A Knowledge-Base Machine (KBM) is the ultimate target of the Japanese Fifth Generation Computer System (FGCS) project [26]. As an intermediate working prototype, the relational database machine DELTA is developed (see [19]). A new information model, the relational knowledge base model, is introduced by the KBM research team. The Relational Knowledge Base (RKB) model is based on the extension of the conventional relational model. This new model represents definite Horn clauses, both rules and facts, in a relational format. Unlike attributes in relational databases, attributes in RKB can be variables or simple data structures*. For this reason, mapping of other forms of knowledge (e.g. production rule, semantic network and frame) to the RKB model is simplified. The KBM is the hardware platform specially designed to support the RKB model.

In RKB, a definite Horn clause comprising a head and a body is stored in a relation with two attributes; one stores the head of the clause and the other stores the body. Both attributes are stored as lists and share the same variable as their last elements. The relation is called a permanent relation and is persistent. For example, the permanent relation (PR) of the rule:

ancestor(X,Y):-parent(X,Z), ancestor(Z,Y).

is as follows:

PR	HEAD	BODY
	[ancestor(X,Y) | T]	[parent(X,Z), ancestor(Z,Y) | T]

*Simple data structures are unnested structures (i.e. lists or functions) – e.g. f(x,y) but not f(g(a),b).

A goal clause (a query) is stored in another relation with two attributes. One stores the expected form of the goal and the other stores the original goal clause in the form of a list, with nil attached to the last element. The relation with a goal clause is called a temporary relation (TR). For example, the TR for the goal clause ancestor(kay,X), is represented as:

TR0	ANSWER	RESOLVENT
	X	[ancestor(kay,X) | nil]

Retrieval-By-Unification (RBU) The built-in inference mechanism of the RKB model is based on a technique known as Retrieval-By-Unification (RBU) [12]. RBU is achieved by continuous unification-restriction and unification-join (U-join) operations. Initially, when a query is posed, TR0 is formed according to the query. The RESOLVENT attribute is matched against the HEAD attributes of the permanent relation (PR). The BODY attributes of the PR tuples whose HEADs match (unify) the query are extracted and stored in another temporary relation, namely TR1, as new RESOLVENTs. For example, matching the TR0 and the PR of the above example gives:

TR1	ANSWER	RESOLVENT
	X	[parent(kay,Z), ancestor(Z,X) | nil]

This action has the effect of restricting the resolution set. At this stage, query variables may be bound. The RESOLVENTs of each of the set of TR1s are then matched against the HEAD of the PR. In general, a new version of TR is generated after each RBU operation cycle on the PR. The RBU operation continues repetitively until there are no more RESOLVENTs (i.e. the RESOLVENT attribute of TRn is nil).

RBU is a complex and time-consuming operation. The primary goal of the KBM hardware is to execute RBU efficiently. The system architecture of the KBM is shown in Figure 1. The KBM is basically a multiprocessor machine with a large shared memory. It consists of a set of customised Unification Engines (UE), a Disc System comprising of a set of discs (DS), a Control Processor (CP), some Main Memory (MM), IO processors (IOP), and a Multi-Port Page Memory (MPPM).

The permanent relations are stored in the DSs and are streamed to the MPPM. The UEs get the data from the MPPM, process them (perform RBU) in a pipeline fashion, and stream the result back to MPPM. The resultant data are returned to the DS, retained for further processing or returned to the users.

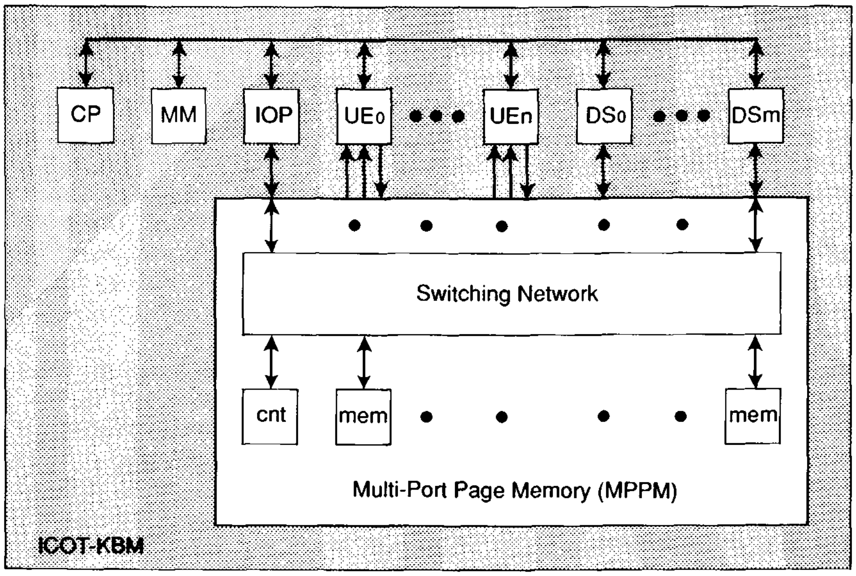


Fig. 1 The system architecture of the ICOT Knowledge Base Machine (KBM)

The Control Processor (CP) controls both the data flow and the parallel execution within the KBM. Respectively, the CP sends control commands and receives responses to and from the other functional units via the control bus. Communication with the MPPM and with the external client machines is provided by the IOP. The UE is the only novel component of the KBM. The architecture of the UE is described in detail in [12, 14]; and its performance is evaluated in [13].

The MPPM provides a wide data bandwidth between the UEs and the DSs. Internally, it consists of a set of IO ports, a set of memory banks under the control of an internal controller. Data are transferred to/from the memory in logical pages. A logical page can be horizontally partitioned across all memory banks. This will enable a page to be read from all the memory banks simultaneously. The connection between a port and a bank of memory is one-to-one and is synchronised with the system clock. In so doing, memory conflicts are reduced.

Current Status: As a milestone of the intermediate stage of the FGCS project which terminates at the end of 1988, an experimental KBM hardware was constructed, known as Mu-X [10]. In order to implement the machine quickly, the Mu-X hardware was constructed using off-the-shelf components. It only emulates the UE functionality using conventional microprocessors – MC68020 at 12·5MHz. Mu-X mainly consists of 8 processing elements (DS, UE, IOP and CP), a conventional shared memory (MM, 2M byte) and a 12M byte MPPM. Each processing element consists of a general purpose microprocessor (MC68020), a 47M byte disc, some local memory and an interface unit to the MPPM. Wisconsin benchmark tests were performed on the Mu-X (see [15]). The performance results are claimed to be satisfactory. It is targeted that by the end of the final stage of the FGCS project (end of 1991), a fully functional KBM will be developed [10].

3 Delta Driven Computer (DDC)

The Delta Driven Computer (DDC) is a multiprocessor knowledge- and data- base computer designed by Bull in France [2, 3]. It is a backend machine which connects to a host. The DDC is aimed to provide an efficient hardware platform for both relational and deductive database applications.

The DDC architecture is divided into two levels: language and hardware. At the language level, DDC supports a high level database query language – SQL. An SQL Database query is compiled to an intermediate language known as VIM (Virtual Interface Machine). The semantics of VIM are based on production rules. A VIM program is further compiled into DDCL which is considered as the assembly language for the DDC†. At the hardware level, the DDC is a parallel computer with a distributed memory architecture. It comprises a set of interconnected identical processing nodes, each consisting of a general purpose microprocessor (MC68020), a communication interface unit, some local memory and a custom VLSI microprogrammable symbolic coprocessor (uSyC) [9]. The uSyC is specially designed for symbolic processing – operations which cannot be executed efficiently on the general purpose microprocessor. The architecture of the DDC is shown in Figure 2.

Parallelism is pursued in the DDC in order to achieve high performance. It is exploited at both the architecture and computation levels. The Delta-Driven Execution Model (DDEM) is specially designed for parallel program execution. DDC manipulates database relations which are distributed among the processing nodes. This allows relations to be processed by all nodes in parallel. The DDC architecture is similar to other conventional 'shared-nothing' database machines – such as GAMMA. The novelty of the DDC machine is the Delta-Driven execution model which can handle both relational database and knowledge-base operations.

3.1 The Delta-Driven Computational Model

The DDC executes VIM programs in sequence. Each VIM program corresponds to a database query. The execution of a VIM program is done in parallel, following a new computational model called Delta-Driven (hence the name of the machine). A VIM program is compiled into the native DDCL code to be executed in the DDC. The Delta-Driven computational model is based on the philosophy of saturation – an execution strategy

†Each DDC processing node runs a DDCL emulator.

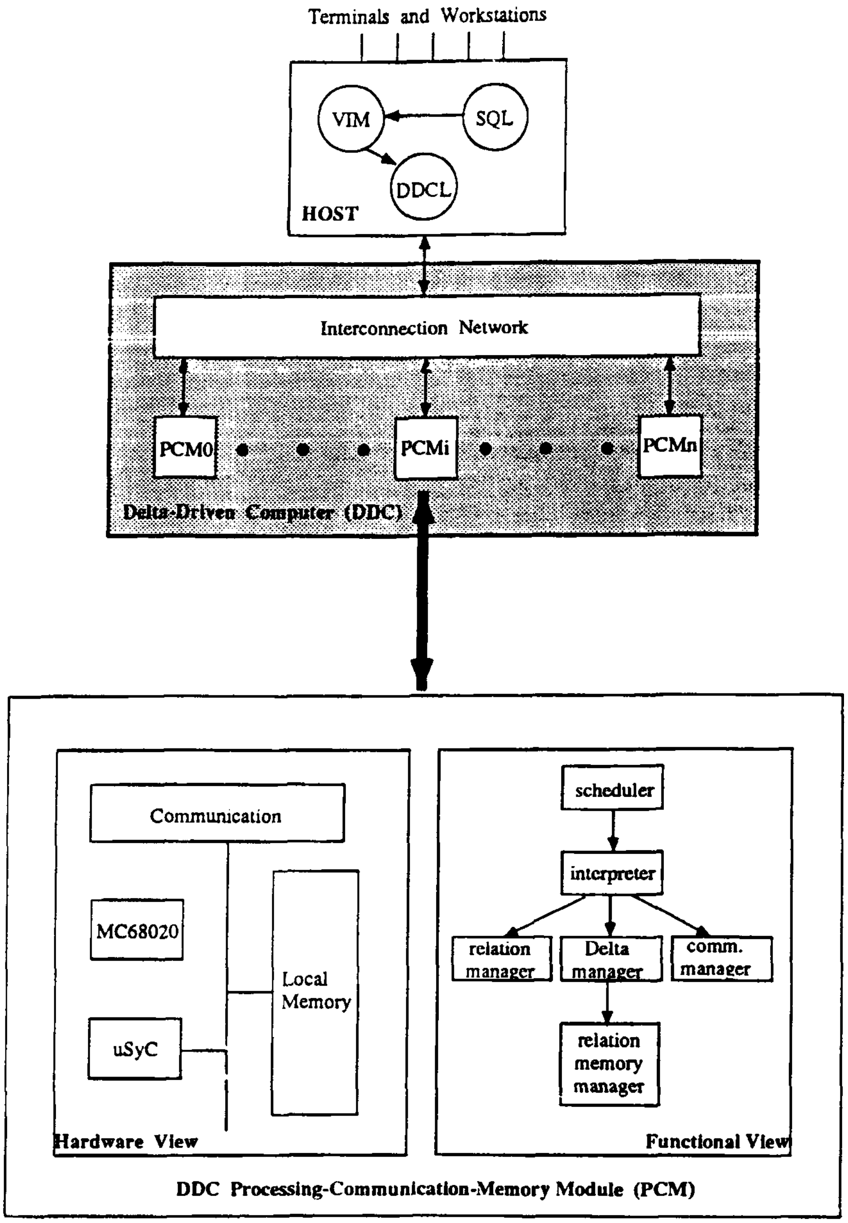


Fig. 2 The architecture of Bull DDC

commonly used in forward chaining production rule systems. A VIM program consists of a sequence of production rules. Each rule takes the form of:

$$\text{HYPOTHESIS} \Rightarrow \text{CONCLUSION}$$

where HYPOTHESIS and CONCLUSION are predicates of the form $p(X_1,\ldots,X_i,\ldots,X_n)$ and X_i is an attribute of the predicate which may be a constant or a variable. HYPOTHESIS consists of one or two predicate(s) and there must be at least one CONCLUSION predicate. Computation of a VIM rule starts from the verification (proof) of the HYPOTHESIS predicate(s). This involves matching the predicate(s) with the exisiting database predicates. If the HYPOTHESIS predicate(s) is proven true i.e. the HYPOTHESIS matches with clauses in the database, its associated CONCLUSION predicates are produced. Production of a CONCLUSION implies adding a new fact (or tuple) into the database. The execution of the VIM rule is complete when the rule is saturated – i.e. no more CONCLUSIONs can be produced or no more facts can be added to the database. The result of a VIM program is a set of new facts deduced from the facts in the existing database.

The Delta-Driven Execution Model (DDEM) applies *actions* to relations. A relation is a set of facts under the same predicate. During the execution of a VIM rule, appropriate relations are transformed to streams of data. The DDEM adopts a dataflow computation strategy which triggers actions as soon as a data stream becomes available. An action involves the consumption of streams of facts and the production of new ones. New streams of facts, after the elimination of duplicates, may then invoke further actions.

A fact produced or consumed by a rule during saturation is referred to as Delta – an increment to the existing database. There are two types of Delta: a White-Delta (WΔ) and a Black-Delta (BΔ). White delta are produced from black delta under the elimination action. The relationship between white and black delta is depicted as follows:

$$\text{W}\Delta\text{fact} \leftarrow \text{B}\Delta\text{fact}-\text{fact}_c \text{ (elimination action)}$$

where fact_c is the current relation. White-Delta are produced by removing the previous content of the facts relation from the newly produced data stream i.e. the Black-Delta. Effectively, the elimination action is to remove duplicates. During a saturation operation, White-Delta are used to initiate the rules which invoke them. Saturation completes when no more White-Delta are produced. As an example, consider the following database consisting of a "parent" relation:

parent(kay,roy). parent(ray,may). parent(tom,kim).

The parent(A,B) predicate says that B is the parent of A. Also, there is a

VIM program which encompasses a set of rules for defining an ancestor, viz:

$$\text{parent}(X,Y) \Rightarrow \text{ancestor}(X,Y).$$
$$\text{parent}(X,Z),\text{ancestor}(Z,Y) \Rightarrow \text{ancestor}(X,Y).$$

The VIM program says: if Y is the parent of X, Y is also X's ancestor (the first rule); alternatively; if Y is the ancestor of the parent of X (i.e. Z), Y is also the ancestor of X. The VIM program is compiled to DDCL code which takes the following form:

Rule-1 $W\Delta\text{parent}(X,Y) \Rightarrow B\Delta\text{ancestor}_0(X,Y)$
Rule-2 $W\Delta\text{parent}(X,Z),\text{ancestor}(Z,Y) \Rightarrow B\Delta\text{ancestor}_1(X,Y)$
Rule-3 $\text{parent}(X,Z),\ W\Delta\text{ancestor}(Z,Y) \Rightarrow B\Delta\text{ancestor}_2(X,Y)$

The saturation computation of the above DDCL is as follows: Initially, an empty ancestor relation is produced – ancestor_c (the subfix 'c' implies current). This ancestor_c relation will eventually contain the final result. It is constantly updated during saturation. Also, during the initialisation phase, a White-Delta parent stream is produced by taking a copy of the parent database relation, viz;

$$W\Delta\text{parent}(X,Y) \leftarrow \text{parent(kay,roy)},\text{parent(ray,may)},\text{parent(tom,kim)}.$$

The DDC computation then continuously executes the following algorithm until no more Delta are produced.

```
Loop:
1 Apply WΔ of parent and ancestor to RULE-1, RULE-2 and RULE-3
2 $B\Delta\text{ancestor} \leftarrow B\Delta\text{ancestor}$ UNION $B\Delta\text{ancestor}_1$, UNION $B\Delta\text{ancestor}_2$
3 $W\Delta\text{ancestor} \leftarrow B\Delta\text{ancestor} - \text{ancestor}_c$
4 $\text{ancestor}_c = \text{ancestor}_c$ UNION $W\Delta\text{ancestor}$
5 IF (WΔancestor is not empty) GOTO Loop
```

During the first cycle, there is only one Delta stream, namely the White-Delta parent (WΔparent). The White-Delta parent is streamed to the DDCL rules (1). Processes executing these rules consume the Delta stream simultaneously producing parallel Black-Delta streams of ancestor. These streams are merged (UNION) producing a single unified Black-Delta ancestor stream (2). To reduce the amount of work in later processing, duplicated facts in the unified Black-Delta stream are eliminated creating a White-Delta ancestor stream (3). The current ancestor database is updated by combining (UNION) the White-Delta ancestor stream with the facts already existing in the database (4). At the same time as step (4), the size of the White-Delta ancestor stream is tested (5). If it is non-empty the execution sequence is repeated from (1); otherwise, the computation is saturated implying the solution set, stored in ancestor_c, is complete and the execution terminates. Notice that when the loop is repeated, both the White-Delta streams of ancestor and parent are fed to the DDCL rules. Details and examples of the Delta-Driven execution model can be found in [2].

Current Status: The Delta-Driven execution model was developed and is used extensively by Bull. The first DDC prototype is a hardware emulation developed on the Bull SPS7 multiprocessor machine. Four processors are used in the prototype. A multiprocessor configuration of UNIX is used and the interprocessor communication is developed using the UNIX communication tools. Each processing node of the DDC is simulated by a UNIX process located on a physical processor. Experience of Bull on the DDC project is applied to an existing project – the EDS system (see later).

4 CLAuse Retrieval Engine (CLARE)

CLARE – CLAuse Retrieval Engine – is a database filter for Prolog database applications implemented at Heriot-Watt University, Edinburgh, Scotland [21]. Although the semantics of Prolog is ideal for database applications, the language has not been widely applied to such applications because of its inefficiency in handling large sets of data (> 100K clauses). The objective of CLARE is to provide an efficient clause retrieval facility thus making the performance of Prolog suitable for large applications. The CLARE machine is based on a two-stage clause filtering philosophy, see Figure 3.

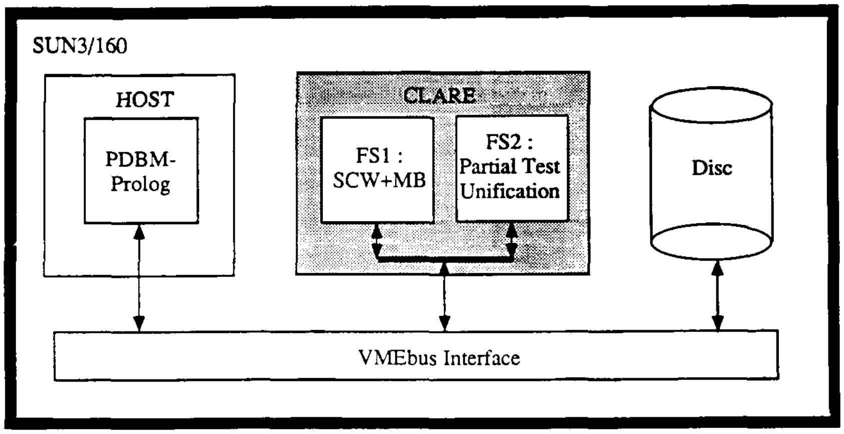


Fig. 3 The overall system architecture of CLARE

The first stage filter [23] is the hardware implementation of a partial index matching scheme – namely, Superimposed Code Word plus Masked Bits (SCW + MB) and the second filter is based on partial-test unification [22]. On the software side, a PDBM-Prolog (PDBM = Prolog Data Base Machine) system is designed in the CLARE project. One of the main goals of the design of the PDBM-Prolog system is to keep the syntax and the semantics of the system as close to conventional Prolog as possible. This is equivalent to extending conventional Prolog with database handling ability. This approach differs from other logic-based deductive database systems. Compared with the database relations of the DDC (discussed before), the domains of relational attributes are not so restrictive. In PDBM-Prolog an argument in a predicate is a full Prolog term which may be a complex structure, a constant or a variable. The PDBM-Prolog system is an integrated Prolog database system: Prolog and database are viewed as a uniform system. Another approach in implementing a Prolog database system is by coupling an existing database system to a Prolog system. Generally, an integrated Prolog system provides a uniform language interface and runs more efficiently. A survey of different implementation approaches for logic database is given in [7].

A large Prolog database, containing over 10k clauses in the PDBM-Prolog systems is stored as 2 files. (If a database is very large, more than 1 million clauses, an additional level of indexing will be introduced.) An index file consists of a dual entry index table. The left hand side of the table contain index codewords of the database clauses and the right hand side contains the corresponding clause reference. The codewords are formed by the Superimposed Code Word plus Masked Bits (SCW + MB) indexing strategy. SCW + MB is most suitable for encoding Prolog clauses because of its ability in handling variables and complex structures with a reasonable codeword size. The clause reference is the clause address on the disc. Details of a clause are stored in the code file. A clause in the code file consists of 2 segments. The header segment contains some type information of the arguments in the head; and the code segment which contains the executable code of the clause in a pseudo-compiled format. Besides being directly executable, this special code format makes de-compilation easier.

When a query is posed the PDBM-Prolog system will select the best (fastest) strategy in resolving the query from:

1 primary indexing if the key attribute is specified in the query;
2 pure software search if the predicate involved is small;
3 first-stage filtering if the expected solution set is small;
4 second-stage filtering only if the potential solution set has already been cached; and
5 two-stage filtering if the predicate involved and the expected solution set are large.

4.1 Two-stage Filtering

The two-stage filtering strategy is the default. During two-stage filtering, the encoded query is loaded into the CLARE, both the first and second stages. A disc transfer is then initiated which streams the index file to the CLARE memory space. As individual codeword passes the CLARE first stage, it is matched against the query codeword. If the result of a match is positive then the corresponding clause reference is kept in the CLARE buffer;

otherwise, the reference will be ignored and matching will continue with the next codeword. Matching completes when the search of the required portion of the index file is exhausted. The clause satisfiers of the first stage are only potential unifiers; some of them may fail at the final full-unification. This is because SCW + MB indexing is only a partial matching technique.

To reduce further the potential resolution set, the clause code of the first stage satisfiers is subjected to further filtering. The second stage filter compares the types of the database arguments (information stored in the header segment of a clause block in the code file) with those of the queries. The argument types are matched according to the partial test matching algorithm. If a database clause satisfies the test it is retained, otherwise the next clause will be tested. The resulting clauses after the second stage filter are passed to the PDBM-Prolog inference engine for full unification. By performing the two-stage filtering the resolution set of a query is drastically reduced. This has the effects of (a) leaving more room (heap space) for the host processor for other computation and (b) reducing the bus traffic between the secondary store and the host, which is advantageous in a multiprocessor environment.

Current Status: A prototype of the CLARE machine consisting of only the first stage filter (CLARE-FS1) has been developed [24]. The first stage filter is based on 188 off-the-shelf MSI/SSI ICs. The construction of the second stage has been abandoned due to the technical problems in low-level disc management. The interface of the CLARE-FS1 hardware is designed to the industrial standard VMEbus specification. The prototype is connected to a SUN3/160 workstation equipped with 4Mbyte of main memory and a 142Mbyte disc system. The CLARE-FS1 is designed as a slave device which maps into the SUN's physical memory space. The search rates of the first stage and the second stage filters are 4·5M and 4·2M words (16-byte) per second, respectively. These rates are much faster than the disc transfer rates of any existing SUN disc system and therefore CLARE-FS1 can provide on-the-fly clause retrieval. On the software side, the PDBM-Prolog system designed to run with CLARE was initially targeted for single user applications. Multi-user capability (including transaction processing, concurrency control and recovery management) is being augmented to the system. At present, prototypes of the PDBM-Prolog system are running on SUN3/160 and ICL3930. For the future, research is underway in enhancing the CLARE-FS1 hardware with set [25] and incomplete information [17] handling capability.

5 PRISMA

The PRISMA database machine (software and hardware) was designed and is being developed jointly by several Dutch universities together with Philips Research Laboratories (Eindhoven) [1, 11, 18]. It is a large scale distributed main memory database management system implemented in an object-oriented language and runs on top of a multi-computer system.

At the interface level the PRISMA supports 3 languages: the relational query language SQL, a custom object-oriented programming language POOL-X and a custom logic programming language PRISMAlog. SQL is incorporated in order to allow the PRISMA database management system to deal with existing relational applications. POOL-X is the system implementation language. It is a parallel object-oriented language and enables any tool developers to bypass SQL thus leading to a tool with better performance. The features of POOL-X include exception handling mechanism, class and inheritance, a flexible type checking scheme, floating point arithmetic and message passing facility. Cooperation of POOL-X objects is achieved via message passing.

The ultimate goal of PRISMA is a single machine for both data and knowledge processing. Knowledge is specified in the form of logic using PRISMAlog. This is based on definite, function-free Horn clauses – i.e. an attribute in a clause can only be a constant or a variable – and adopts the Prolog syntax. The semantics of PRISMAlog is defined in terms of extensions of the relational algebra. Facts correspond to tuples and rules correspond to database views. Compared to Prolog, PRISMAlog is set-oriented, which makes it suitable for parallel evaluation.

The virtual architecture of the PRISMA database machine (see Figure 4) consists of several decentralised database subsystems, known as One-Fragment Managers (OFM), running under the supervision of a Global Data Handler (GDH). The GDH is responsible for data dictionary management, transactions management, concurrency control, recovery and query compilation/optimisation. Parallelism is achieved by distributing both database management and query processing. Databases are partitioned and tuples may be stored as segments in different OFM. In so doing, evaluation of fragments can be processed in parallel. Internally, a OFM is basically a small database management system in its own right (see figure). It consists of its own local query optimiser, transaction table, access modules and local data dictionary. Ideally, each GDH and OFM should run on a separate processor.

The hardware architecture of the PRISMA machine consists of a number of processing elements connected via a high bandwidth message passing network. Each processing element consists of a processor, some local memory and a communication processor.

Current Status: The software and hardware of the PRISMA database machine are being developed independently. On the software side, a first prototype of the PRISMA database management system is running on a POOL-X interpreter on a sequential machine. A prototype of the target multiprocessor PRISMA machine is still under development. It consists of 64 processing elements. Each element consists of 16 Mbytes of local memory and 4 10 Mbits per second communication links. The topology of the interconnection network is a mesh-like or a variant of the chordal ring.

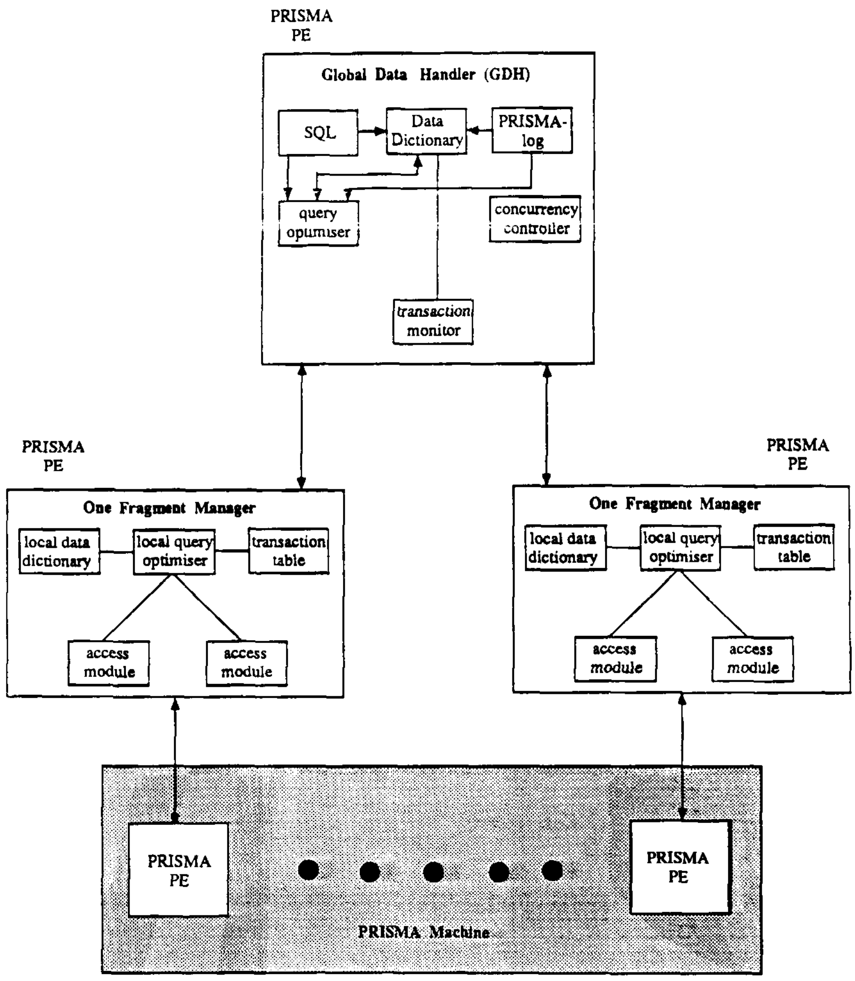


Fig. 4 Architecture of the PRISMA Database Machine

Although the PRISMA is a main memory machine, some of the processing elements will be attached with secondary disc storage. The discs will be used for stable storage and automatic recovery upon system failures.

6 EDS Machine

The European Declarative System (EDS) is a 4 year ESPRIT II project (European Strategic Programme for Research and Development in Information Technology) started in 1989 [4, 5, 20]. The main participants the project are ICL (UK), Bull (France), Siemens (Germany) and ECRC (Europe) together with various European universities. The project is to develop (and apply) parallel techniques within the fields of advanced database management systems and application systems for present and future business professionals. This involves the design and implementation of a parallel database/knowledge software environment and a parallel hardware platform for supporting it. Database is the main stream application; in addition to that, the EDS machine will support parallel functional, logic and object oriented programming languages. The conceptual view of the EDS database system

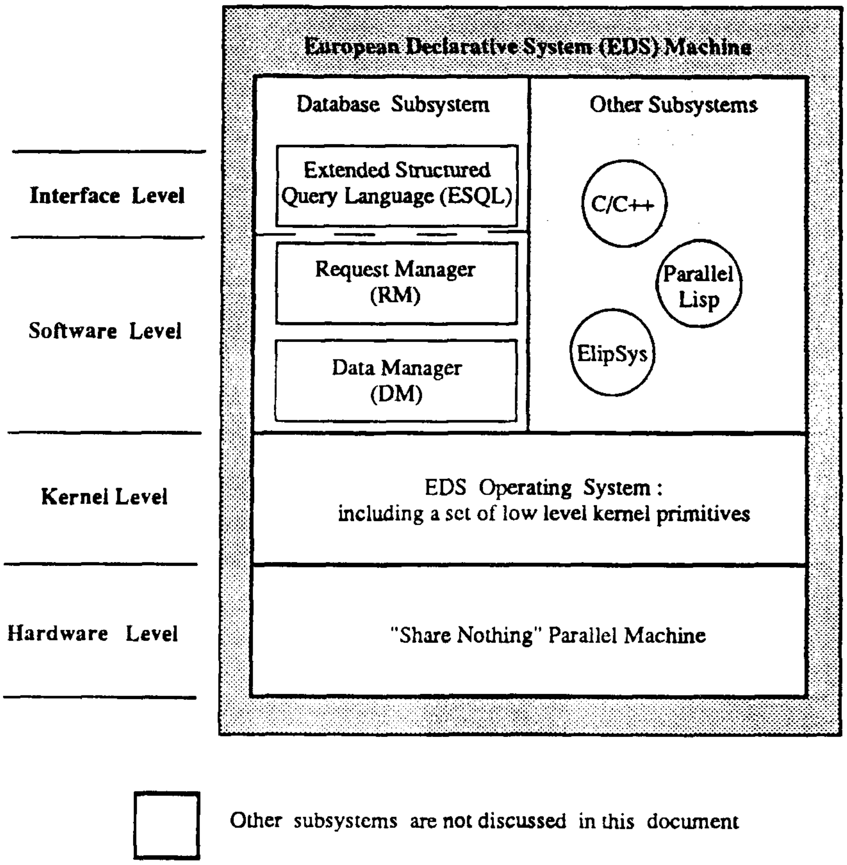

European Declarative System (EDS) Machine
Database Subsystem
Other Subsystems
Interface Level
Extended Structured Query Language (ESQL)
C/C++
Software Level
Request Manager (RM)
Parallel Lisp
Data Manager (DM)
ElipSys
Kernel Level
EDS Operating System: including a set of low level kernel primitives
Hardware Level
"Share Nothing" Parallel Machine
Other subsystems are not discussed in this document

Fig. 5 The Conceptual View of the EDS Database System Architecture

architecture is shown in Figure 5. The architecture can be divided into four levels; they are interface, software, kernel and hardware levels.

User Level: At the user interface level, database users send queries written in Extended Structured Query Language (ESQL) to the EDS database system [8]. The EDS database system adopts the relational database model as the foundation. This is because the relational model is well established and is believed to be the prime technology for database applications in the 1990s. As its name suggests, ESQL is an extension to the industry standard database query language SQL. Besides the relational semantics of SQL, ESQL can also handle user-defined complex data structures (e.g. Abstract Data Type: ADT), deductive rules and recursive queries. These extensions build the EDS system with the capability to support non-relational database applications. Similar to conventional SQL, the ESQL consists of a data definition language (DDL) to manage and process data definitions and a data manipulation language (DML) to update and retrieve stored data.

Provision of Abstract Data Types (ADTs) makes EDS suitable for advanced database application. An ADT is a user-defined data type. Apart from specifying the format of a relation, an ADT consists of a set of user defined functions, referred to as methods, which is specifically designed for and solely applicable to the data which it represents. Moreover, complex object structures can be constructed incrementally from existing ADTs. This flexibility in data structuring allows users to model real life objects easily. ADT also supports the notion of object class. An ADT structure can be defined as a sub-class of another object; in so doing, the former will inherit the properties of the latter. The following example shows the information modelling power of the ADT facility:

Example of ADT
Convention: In this example, **bold** faced words in capital letters are system primitives; *italic* words are ADTs; and words in roman font and in small letters are user defined names.
A base relation 'triangles' is defined which consists of a set of triangles with different colours. Each triangle is uniquely identified by a trianglekey.

CREATE TABLE triangles
(trianglekey **INT**, colour **STRING**, shape *TRIANGLE*
KEY IS trianglekey; *TRIANGLE* is a user defined ADT which is a subtype of another ADT, namely *POLYGON*:

CREATE TYPE *TRIANGLE*
SUBTYPE OF *POLYGON*
WITH (height FLOAT);

CREATE TYPE *POLYGON*OBJECT
TUPLE OF (polygonkey **INT**, vertices **SET OF** *POINTS*;

CREATE TYPE *POINT*
(x **FLOAT**, y **FLOAT**);

Notice that apart from the properties (or structure) inherited from *POLYGON*, *TRIANGLE* also has its proprietary attribute, namely height. **TUPLE** and **SET** are basic generic ADTs provided by the system. These generic ADTs come with *methods* – e.g. methods for a **TUPLE** may be select, filter, ... , etc.

Software Level: Next is the software level which comprises a Request Manager (RM) and a Data Manager (DM). This level is responsible to the execution of the ESQL. An ESQL query is distributed to one or more Request Managers (RM)‡. The RM performs several operations on the query: The ESQL query is compiled. This is to ensure that the query is both syntactically and semantically correct; if it is not, an error message will be returned to the requesting user. Otherwise, if the compilation is successful, the query is optimised, parallelised and bound with some run-time library modules. At the end of all the internal operations within RM, an ESQL query is translated into an intermediate language format, namely LERA (Language for Extended Relational Algebra). The LERA compiler contains information on the accessing methods available to a relation and selects the one most suitable for a query operation. Supporting access methods include:

1 ALL – SELECT – performs an exhaustive retrieval of a relation.
2 EXACT – SELECT – retrieval of a relation according to a specific attribute pattern.
3 INDEX – SELECT – retrieval of a relation using a secondary index on a relation.
4 NEST – JOIN – perform a join of relations R and S. R is first distributed to each of the processing nodes§ where S is stored. Then the join is performed according to the classic nested-loop algorithm. (Relation R is assumed smaller than S. This reduces the network traffic).
5 HASH – JOIN – performs a join of relations R and S. Tuples of R and S are retrieved from their home processing node. The tuples are then distributed to a set of processing nodes according to hashed values of the join attributes. On these nodes, the tuples are joined using a nested loop algorithm.
6 ASSOC – JOIN – performs join of relations R and S. Relation S must be originally stored in a distributed fashion based on the hash values of the join attributes. During the join operation, relation R is distributed to the corresponding processing nodes holding S (i.e. homes of S) according to the hashed values of the join attributes. Then each processing node completes the join operation by scanning R and accessing S based on their hashed indexes of the join attributes.

A LERA program is structured as an execution tree with nodes as LERA operations and stems as streams of relation fragments. Typical LERA operations include FILTER, MERGE, DISTRIBUTE, ... etc. Figure 6 shows the graphical representation of a LERA program for distributed join between relations A and B. In this program, relations A and B exist as fragments – A_0 – A_n and B_0 – B_n – in processing nodes PN_0 – PN_n. Each processing element is responsible for the join operation of the corresponding A and B fragments – i.e. A_i join with B_i in PN_i. Later, the resulting fragments

‡At the hardware level, each RM may exist on a unique processing node in the EDS parallel machine. Such a node is known as a Request Manager Machine (RMM).

§Note that throughout the description of the EDS machines, the term 'node' represents different entities in different contexts. It may represent a LERA operation, a BREM computation or a hardware processing element.

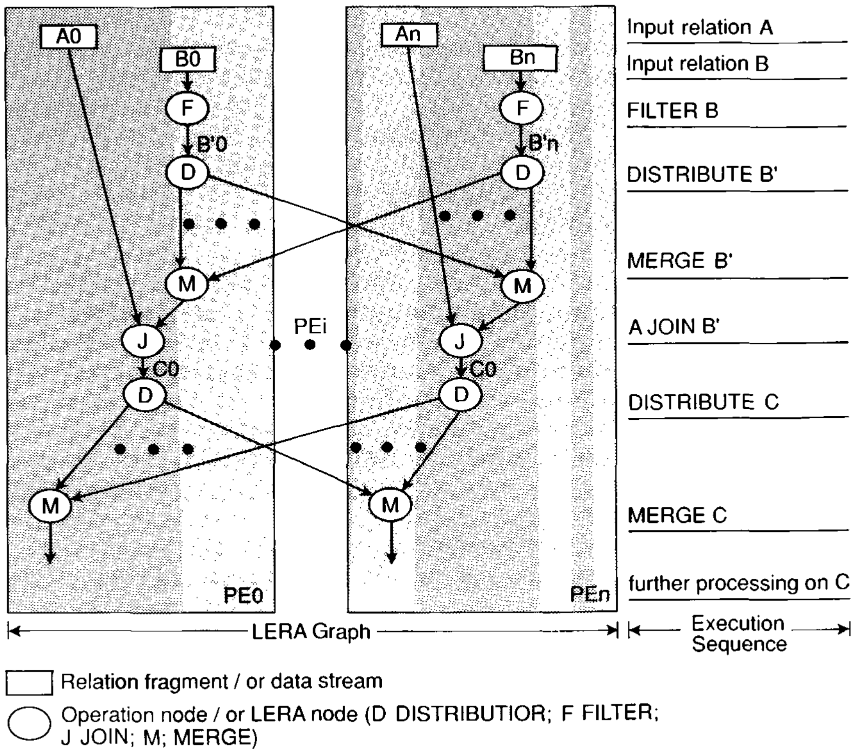


Fig. 6 A distributed join (A join B) algorithm represented in a LERA graph

C_i are redistributed (distribute followed by merge) for further operations. A relation fragment may be filtered prior to any operation – e.g. $B_i \rightarrow B'_i$. Note that a LERA program, such as figure 6, is only an abstract execution sequence.

LERA programs are executed by the Data Manager (DM). According to the information specified in the LERA code, the DM distributes the database operations to the corresponding processing elements in the EDS machine. The fundamental execution strategy of the DM is to bring processing to data. This requires the DM to know the physical storage locations of all relations fragments.

Query execution on the DM is based on the Basic Relational Execution Model (BREM). BREM is responsible for work – units of works may be tasks, teams and threads – scheduling, mapping of works to physical processing elements and provides store management facilities. The computational model supported by BREM is based on graph-reduction. A BREM graph comprising nodes connected by arcs represents the state of a computation. A node may be an operation node, a structure node or a variable node. A node (the parent) may reference another node (the child) via a reference arc. The nature of the reference between two nodes may be one of:

- *Object reference* – the parent node refers to the value of the child node; the latter may be a variable or a structure node.
- *Functional reference* – the parent node invokes the function of the child (operation) node.

An operation node may be in one of *dormant, suspended* and *active* states. At the beginning of the computational process, a *dormant* operation node is 'woken' by a system call–initiated by a user query, for example. This node turns *active* and becomes the root of a computation graph. According to its outgoing arcs, the root node may further activate other *dormant* and *suspended* BREM nodes/graphs. Computation is completed when the BREM nodes/graphs invoked by the root node inform the root that their operations are finished. Multiple BREM computation graphs may be active simultaneously and may be executed in different processing elements in parallel. Effectively, the BREM can be regarded as an executive system (or high-level operating system) implemented on top of the the EDS kernel.

Kernel Level The third level is the operating system kernel. The EDS kernel is known as the EDS Machine EXecutive (EMEX). EMEX is responsible for overall resource management of the EDS system. It supports (1) a lightweight process model which implements the concepts of threads, teams and tasks; (2) message passing: inter/intra process communications coupled with efficient buffer management; and (3) a memory model which employs a virtual address scheme that can spread across an arbitrary number of PEs. These facilities are essential for large scale parallelism. The EDS language subsystems (parallel LISP, Elip-Sys, ESQL and C/C++) can invoke EMEX facilities by using the Process Control Language (PCL). PCL has similar design objectives to other distributed operating systems (e.g. Mach and Chorus); this implies that porting existing applications written for such systems to the EDS machine will be possible. Conceptually, PCL may be regarded as a software bus which allows the different language subsystems to coexist in the EDS machines.

Hardware Level The hardware level involves the EDS machine architecture [16]. Figure 7 shows the hardware architecture of the EDS machine. The EDS machine is a "shared-nothing" distributed memory multiprocessor machine. It consists of an inter-connection network capable of integrating up to 256 processing elements. All processing elements are identical and consist of the following units:

- A Processing Unit (PU) which runs the kernel and the application code.
- A System Support Unit (SSU) which is responsible for the low level functions of message passing and store copying. It may also support additional kernel functionality such as scheduling and work distribution

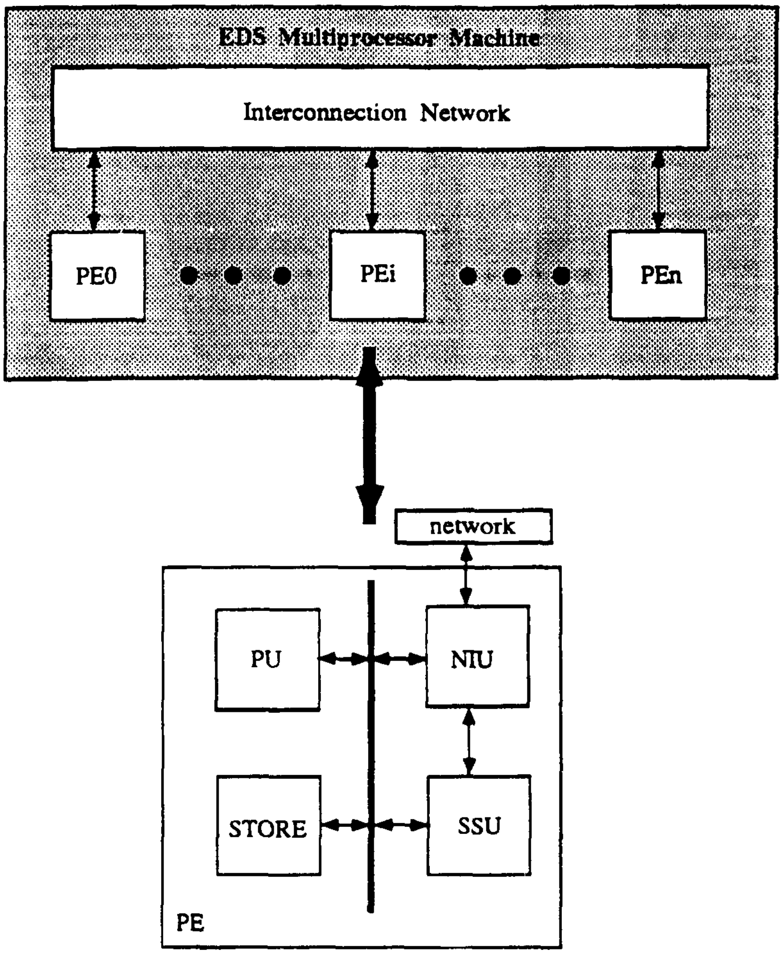


Fig. 7 Hardware architecture of the EDS machine

and any functions which it may be programmed to do to off-load the PU.

- A Network Interface Unit (NIU) which is used to initiate and control inter-processor element communication.
- A Store Unit (STORE) comprising local memory up to 2G bytes. Error detection and recovery facilities are provided by the STORE.

A database transaction is executed in parallel. The job is distributed to the PUs of multiple PEs. During the course of execution, a PU may communicate or extract information from another processing node. To do that the PU compiles the communication packet and places it in a message output queue in the STORE unit – the queue is shared by the SSU. The SSU reads the output queue and sets up the network packet by adding book-keeping

Table 1 Summary of the knowledge base machine projects.

Summary of the Knowledge Base Machine Projects

Machines	Principal Research Institutions	Sponsorship	Class of Machine	DB/KB Languages	DB/KB Models	Execution Models	Current Status
DDC	BULL (France)	ESPRIT-I (European)	Backend Machine	SQL	RDB ProdS DDB	DDEM (Delta-Driven Execution Model)	– A 4 processor prototype based on the BULL SPS7 multiprocessor computer – Design and implementation of DDEM
CLARE	Heriot-Watt University (Scotland)	ALVEY (UK)	Filter	PDBM-Prolog	DDB	– Resolution and Unification – 2-stage filtering	– 2-stage filtering hardware running on SUN3/160 – PDBM-Prolog on SUN3 and ICL3930 mainframe
PRISMA	Philips Research Laboratories University of Amsterdam University of Leiden University of Twente University of Utrecht (Holland)	SPIN (Dutch)	Server	SQL POOL-X PRISMAlog	RDB OODB DDB	Object-oriented execution mechanism	– PRISMA database manger is implemented – PRISMA hardware is still under construction

Table 1 (*continued*)

KBM	ICOT (Japan)	FGCS (Japan)	Server	RKB language	DDB RDB RDB	RKBM based on RBU and RBR techniques	– Mu-X, a 8-processor hardware prototype – Final system by end of 1992
EDS	ICL (UK) BULL (France) (Germany) ECRC (Europe)	ESPRIT-II (European)	Server	ESQL	RDB DDB OODB	BREM based on graph reduction techniques	– System Design is complete – 4-PE test bed available in mid 91 – 64 PE prototype by end of 1992

List of Abbreviations

BREM	Basic Relational Execution Model	DDB	Deductive Data Base
DDEM	Delta-Driven Execution Model	ESPIRIT	European Strategic Programme for Research and Development in Information Technology
FGCS	Fifth Generation Computer System	OODB	Object-Oriented Data Base
PDBM	Prolog Database Machine	ProdS	Production System
RBR	Retrieval-By-Restriction	RBU	Retrieval-by-Unification
RDB	Relational Database	RKB	Relational Knowledge Base
RKBM	Relational Knowledge Base Model	SPIN	Stimulerings Projection INformaticaondersock
(E)SQL	(Extended) Structured Query Language	VIM	Virtual Interface Machine

information to the original packet. The packet is then sent to the network via the NIU. There are two types of communication packet: store-copying and message-passing. At the receiving node, the NIU accepts the packet from the network and passes it to the SSU. Depending on the type of the incoming packet, the SSU may instruct the NIU to perform a DMA copying the content of the message to the STORE unit (store-copying) without intervention of the PU; or it may acquire a buffer from the PU's buffer pool, copy the content of the packet to the buffer and then place the buffer pointer to the end of the PU's input queue (message-passing). The message at the input queue will be serviced at a later time. Once a packet is sent an acknowledgement packet will be transmitted to the sender.

Current Status: The EDS project is entering its third year. The first year marked the definition phase of the project and the design of the system was completed in the second year. The definition of the EDS system has been finalised [4]. The evaluations of the definition are positive (by the commission of the European research committee). A test rig consisting of 4 processing elements is due to be completed by mid 1991. The final prototype system with 64 processing elements interconnected by a delta switching network is expected to be developed by the end of 1992. The final prototype hardware will have the following features: the delta interconnection network will be 20Mbytes per second per channel; each processing element will have a SPARC-RISC PU; units on a processing element will be tightly integrated with each other by M-bus (Module-bus, an upcoming standard for SPARC processors).

7 Conclusion

Architectures of five knowledge-base machines have been reviewed. These machines are still under development or design and are not ready for real applications. Table 1 presents a summary of these machines. The progress of knowledge base machine research is moderate as compared to similar activities on database machines (e.g. GAMMA). This is because the knowledge base system is still a relatively new area. The theory and the implementation techniques behind knowledge base models are not fully established. Therefore, in knowledge base machine research, besides the need for designing the machine architectures, research on the theory and implementation techniques for the underlying knowledge base models are also required. Both research activities are complex and the progress of one will influence that of the others.

Although the relational model is unsuitable for advanced applications such as CAD, it will not be completely replaced by knowledge base systems in the future. The key to the success of relational databases is their simplicity. A wide range of database applications is well defined and fits neatly into the simple tabular notion of the relational model. Commercial credit/debit type applications are classical examples of relational systems. The size of databases and the number of new databases are growing rapidly each year.

By the time when a practical knowledge base system is introduced to the market, relational systems will become even more widespread and databases which exist for years will be very large in size. These factors will make the task of system conversion, from a database system to a knowledge base system, unmanageable – if not impossible. For this reason, the design of knowledge base machines must not neglect the significance of relational systems. All the knowledge base machines described in this paper, apart from CLARE, support relational databases. In particular, EDS provides a simple and uniform language interface, namely ESQL, for both database and knowledge base applications. In the knowledge base field it is evident that the deductive and the object-oriented paradigms are most widely used.

CLARE is slightly different from the other four knowledge base machines being designed to run single-user applications. The CLARE hardware functions as a coprocessor unit inside a workstation. Under the CLARE design principle, implementation of a multiuser environment is a system level design issue. Multiple workstations can be networked to form a distributed multiuser environment and CLARE is the database filter on each site. In contrast, the other machines are designed specifically for multiuser applications. The hardware involved in them is more complex than CLARE. High performance in these machines is achieved by exploiting parallelism at all levels – executing multiple transactions, queries, instructions and operations in parallel. The ICOT KBM is a shared memory machine; memory sharing is provided by the wide bandwidth Multi-Port Page Memory (MPPM). The hardware architectures of the DDC, PRISMA and EDS are similar to each other. They are "shared-nothing" multiprocessor machines with distributed memory. There are many practical problems in designing multiprocessor systems, let alone designing multiprocessor knowledge base machines. Two fundamental design issues seriously affect the overall performance of a multiprocessor system, viz:

1 *Data distribution.* Storage of a database is shared between many processing elements. The sharing must be organised in such a way that the efficiency of future access to the database is optimised.
2 *Work distribution.* Program execution is performed by multiple processing elements in parallel. The work load among the participating processing elements must be balanced in order to achieve maximum throughput.

At present, intensive research on both issues is being actively conducted.

Acknowledgements

The work leading to this paper was partially funded by the Commission of the European Communities as part of ESPRIT II project EP2025, the European Declarative System (EDS). Thanks are due to Mr. Hans Benker for his useful comments on the initial draft; Dr. Mike Reeve for his support of this review work at ECRC and Dr. John Pinkerton for his suggestions and his patience in proof-reading the manuscripts many times.

References

1 APERS, P.M.G., KERSTEN, M.L. and OERLEMANS, H.C.M.: "PRISMA Database Machine: A Distributed, Main-Memory Approach", Advances in Databse Technology – EDBT'88, Schmidt, J.W., Ceri, S. and Missikoff, M. (eds.), Lecture Notes in Computer Science (303), pp 590–593, Springer-Verlag, 1988.

2 BERGSTEN, B. COUPRIE, M., GONZÁLEZ-RUBIO, KERHERVE, B. and ZIANE, M.: "A Parallel Database Accelerator", PARLE'89 Parallel Architectures and Languages Europe, vol. 1, Odijk, E., Rem, M. and Syre, J.C. (ed.) pp 397–412. Lecture Notes in Computer Science (365), Springer-Verlag, 1989.

3 BERGSTEN, B., COUPRIE, M., GONZALEZ-RUBIO, KERHERVE, B. and ZIANE, M.: "Language Levels and Computational Model for a Parallel Database Accelerator", Database Machines Sixth Internation Workshop – IWDM'89, Boral, H. and Faudemay, P. (ed.), pp 58–72, Lecture Notes in Computer Science (368), Springer-Verlag, 1989.

4 HAWORTH, G., LEUNIG, S., HAMMER, C. and REEVE, M.: "The European Declarative System, Database and Languages", IEEE Micro, vol. 10, no. 6, pp 20–23 and 83–88, Dec. '90.

5 DETTMER, R.: "Flagship and EDS", IEE Review, IEE(UK), p 68, Feb '90.

6 GALLAIRE, H. and MINKER, J. (eds.): Logic and Databases, Plenum, New York, USA, 1978.

7 GALLAIRE, H.: "Logical Data Bases Vs Deductive Data Bases", Proc. Logic Programming Workshop, pp 608–622. Universidade Nova de Lisbon, Portugal, 1983.

8 GARDARIN, G. and VALDURIEZ, P.: "ESQL: An Extended SQL with Object and Deductive Capabilities", Proc. International Conference on Database and Expert System Applications, Tjoa, A.M. and Wagner, R. (eds.), pp 299–307, Springer-Verlag, Berlin, Aug '90.

9 GONZALEZ-RUBIO, R., BRAIDIER, A. and ROHMER, J.: "DDC: Delta Driven Computer and eSyC: Microprogrammable Symbolic Coprocessor", Proc. EUROMICRO '88, Zurich, Sept '88.

10 KUROZUMI, T.: "Present Status and Plans for Research and Development", Proc. International Conference on Fifth Generation Computer System, ICOT (ed.), pp 3–15, vol. 1, Tokyo, Japan, 28 Nov–2 Dec '88.

11 KERSTEN, M.L., APERS, P.M.G., HOUTSMA, M.A.W., VAN KUYK, E.J.A. and VAN DE WEG, R.L.W.: "A Distributed Main-Memory Database Machine: Research Issues and a Preliminary Architecture", Database Machines and Knowledge Base Machines, Kitsuregawa, M. and Tanaka, H. (eds.), pp 353–369, Kluwer Academic Publishers, 1988.

12 MORITA, Y., YOKOTA, H., NISHIDA, K. and ITOH, H.: "Retrieval-By-Unification Operation on Relational Knowledge Base Model", Proc. 12th International Conference on Very Large Databases, Aug '86.

13 MORITA, Y., OGURO, M., SAKAI, H., SHIBAYAMA, S., ITOH, H. and MORITA, Y.: "Performance Evaluation of a Unification Engine for a Knowledge Base Machine", ICOT Technical Report TR-240, ICOT, Japan, Mar '87.

14 SAKAI, H., SHIBAYAMA, S., MONOI, H., MORITA, Y. and ITOH, H.: "A Simulation Study of A Knowledge Base Machine Architecture", Database Machines and Knowledge Base Machines, Kitsuregawa, M. and Tanaka, H. (eds.), pp 585–598, Kluwer Academic Publishers, 1988.

15 SHIBAYAMA, S., SAKAI,H., TAKEWAKI, T., MONOI, H., MORITA, H. and ITOH, H.: "Overview of Knowledge Base Mechanism", Proc. International Conference on Fifth Generation Computer System, ICOT (ed.) pp 197–207, vol. 1, Tokyo, Japan, 28 Nov–2 Dec '88.

16 WARD, M., TOWNSEND, P. and WATZIAWIK, G.: "EDS Hardware Architecture", Parle 90 Conference, Burkhard, H. (ed.), Lecture Notes in Computer Science (457), pp 816–827, Springer-Verlag, 1990.

17 WILLIAM, M.H. and WONG, K.F.: "Extending the Superimposed Codeword Indexing Scheme to Handle Incomplete Information", March '91, (under preparation).
18 WILSCHUT, A.N., GREFEN, P.W.P.J., APERS, P.M.G. and KERSTEN, M.L.: "Implementing PRISMA/DB in an OOPL", Database Machines Sixth Internation Workshop – IWDM'89, Boral, H. and Faudemay, P. (ed.), pp 97–111, Lecture Notes in Computer Science (368), Springer-Verlag, 1989.
19 WONG, K.F.: "Architectures of Database Machines", ICL Tech. J. Vol. 7 No. 3, May 91.
20 WONG, K.F. and et. al.: "The European Declarative System (EDS) as a Platform for Parallel Logic Programming", Proc. 2nd IEEE Symposium on Distributed and Parallel Processing, pp 339–342, Dallas, Texas, USA, 9–11 Dec '90.
21 WONG, K.F. and WILLIAMS, M.H.: "Design Considerations for a Prolog Database Engine", Proc. 3rd International Conference on Data and Knowledge Bases, pp 111–119, Jerusalem, Israel, 28–30 June '88.
22 WONG, K.F. and WILLIAMS, M.H.: "A Type Driven Hardware Engine for Prolog Clause Retrieval over a Large Knowledge Base", Proc. 16th International Symposium on Computer Architecture, pp 211–222, Jerusalem, Israel, 28 May–1 June '89.
23 WONG, K.F. and WILLIAMS, M.H.: "CLARE – A Prolog Database Machine", Proc. 1990 ACM Symposium on Personal and Small Computers, Arlington, Virginia, USA, 28–30 March '90.
24 WONG, K.F. and WILLIAMS, M.H.: "Index Matching Hardware for Selective Clause Retrieval in Large Prolog Knowledge Bases", Oct. '90, (submitted to IEEE Micro).
25 WONG, K.F. and WILLIAMS, M.H.: "A Superimposed Codeword Indexing Scheme for Handline Sets in Prolog Databases", Proc. 2nd International Symposium on Database Systems for Advanced Applications, Tokyo, Japan, 2–4 April '91.
26 YOKOTA, H. and ITOH, H.: "A Model and an Architecture for a Relational Knowledge Base", Proc. 14th International Symposium on Computer Architecture, pp 2–9, Tokyo, Japan, 2–5 June '86.

Biography

Kam-Fai Wong

Kam-Fai Wong received his B.Sc. and Ph.D. from the Department of Electrical Engineering of Edinburgh University in 1983 and 1987, respectively. For his Ph.D. thesis, he designed and developed a hardware garbage collector system for real-time AI applications. He joined the Computer Science Department of Heriot-Watt University in 1986 and worked as a research associate on the Prolog Database Machine Project. In 1988, he joined the System Software Engineering Department at Unisys. Livingston. Scotland. There he worked briefly for a year as a software engineer designing a real-time kernel. Since the end of 1989, he has been working for the European Computer-Industry Research Center (ECRC) at Munich. He is a researcher in the Computer Architecture group at ECRC and is currently leading a small team investigating performance issues of parallel database systems.

The Origins of Pericles – A Common On-Line Interface

J.W.S. Carmichael
ICL Secure Systems International, Winnersh, Berks, UK

Abstract

In common with all large companies ICL is completely dependent on computer systems for the running of its business. Over the years the company has developed a number of systems, primarily tailored to its own needs but of various degrees of generality. One of the most powerful of these, with what would now be called a particularly user-friendly Man-Machine (or Human-Computer) Interface was PERICLES; this was developed for the ICL 1900 series and first went live in 1975. The paper summarises the history of this system and its main features.

1 Historical Background

During the early 1970s, the organisation of internal IT departments within ICL had not yet been co-ordinated, and each main operating division of the Company had its own separate computer system development group. Such groups were based at Putney, Stevenage, Letchworth, Kidsgrove, West Gorton, and Sydenham.

Until 1972 or thereabouts, this separation of functions had caused no serious difficulty. Each group had been actively engaged in the development of primary application systems for its own local masters. Interchange of information between systems in different organisations had not progressed much beyond the stage of A producing a print-out and B re-keying the relevant data.

Different standards and procedures were in force in the various units, but at least one potential source of difficulty had been eliminated: all groups were operating on similar 1900 series configurations, so that there was no fundamental technical incompatibility.

Nevertheless, it was obvious that a greater degree of co-ordination would become essential, and that more efficient data interchange would involve a convergence of standards.

In one field, this convergence gave rise to one of the most important of all internal software developments: the Pericles system for running online applications.

Each of the computer groups had by this time developed one or more online applications, and they exhibited a great extent of incompatibility. Even the authors of the basic software for handling transactions from online terminals would admit that their early efforts were at a very low level, and provided ample opportunity for application builders to produce systems with widely differing approaches to system disciplines and styles of end-user interface.

This meant, at the first level, that a user of a Stevenage system would feel quite unfamiliar with the demands of a Putney system, the user of a Kidsgrove system would feel very strongly the effects of cultural difference if he ever tried to access a West Gorton system, and even that a Letchworth user would have difficulty signing in to a Stevenage system, despite the fact that it had been developed only seven miles away.

During 1973, therefore, a study was inaugurated into the existing online applications, and their underlying software structures, in order to determine what would be the best basis for a Company-wide standard for the future development of online applications. It is not the purpose of this paper to recapitulate the arguments which took place, nor to revive the pride and ardour with which the supporters of each contesting system quite properly advocated the strengths and merits of their local candidate. Let it simply record that the principles of Pericles carried the day.

2 Establishing the Needs

Pericles started from the observation – daring at the time, for all its obviousness now – that a user might want to access more than one online application from his terminal, and that he might want to do so via a common interface. Hence arose the idea that Corporate Information Services (CIS) should aim to provide an online service, within which multiple applications would be available.

Thoughtful application developers had also become aware by then that writing online applications involved much more than simply creating, accessing, and amending the data records which were their prime concern. A great deal of housekeeping was also involved: locking, security, and recovery were all topics to which they had to devote effort, and such effort was grudged as a diversion from the interesting business of building applications.

From there it was really but a short step to deciding that a common online application environment should be produced, with the most comprehensive set of facilities for controlling access – to the service itself, to applications within the service, and even to commands within each application; for maintaining the integrity of the service, through centralised record-locking, journalising, and recovery; and for monitoring the performance of the service through the creation of statistics on connectivity, activity, throughput, transaction existence times, etc.

3 What PERICLES provided

The development of Pericles, incorporating all of these common functions, occupied the whole of 1974 and extended into 1975. The very first Pericles application – Scratchpad, (of which more later) – went live in August 1974. The Orders application followed it at the turn of the year, being implemented in late 1974 and inaugurated for live use in early 1975.

Even at an early stage major advantages became apparent from the discipline of segregating application code from lower-level functions. In the earliest days, Pericles used the Scanner Housekeeping package to communicate with user terminals. The fact that this package was limited to 32k-word addressing almost immediately necessitated a progress toward full sub-programming, with WMC (Within Machine Communication) facilities used to pass data and control between application code and communications code. Later on, Communications Manager naturally became the primary vehicle for control of all communications traffic. But such changes were mercifully transparent to the application code, because the interface had been defined at a sufficiently high level.

From the start Pericles incorporated a Control Application, to provide system-wide control over user access. Initially users were identified by a distinct user number within each Pericles service to which they had access. But for many years such identification has been achieved by the use of the individual's personnel number, and this has been checked for validity against the main Personnel File.

The Control Application also allowed the system managers to monitor which files were in use at any time, to review which users were attached to the system, and so forth. It was also the focus of much of the system security functionality, which even now must not be described in a general publication.

An application-independent user interface was established by means of a series of system-level commands. These were distinguished by being introduced by an asterisk, which ensured that there could be no ambiguity between them and commands which only applied within a single application. The most common of these were:

*SI Sign-in
*SO Sign-off
*CA Change Application; (this was found to be a real boon, when contrasted with the preceding rigmarole of totally signing off from one application and re-establishing contact with a different online service for access to a second application.)
*CP Change Password
*?? This provided a primitive sort of HELP facility. Within each application it gave access to one or more screens which briefly described the available commands and their parameters. The detail was obviously application-specific, but at least the means of accessing it was common.

System controllers could also benefit from such restricted commands as:

*WU Which Users? This provided a table identifying the current population of logged-in users to an application, and could be used by an application controller to ensure that modifications could be enacted without disrupting current activity.
*WF Which Files? This displayed the current status of all files known to the system.
*RR Recovery Recall. In cases where the service had temporarily broken for any reason, this provided a means of confirming which was the last creation or updating transaction which had been successfully completed.

More for use in debugging situations than in live running there were also commands such as:

*DP Display File Contents, which output unformatted details of particular records, much as one can do with List Records or Browse File within a VME service.
*PT "Print", to display the contents of specified words or areas of main store.

Broadcast facilities were also supported as part of the Control Application: the central system control could send messages to all system users, to the users of a particular application, to the user of a particular terminal, and so on.

System controllers were also provided with the capability to activate or suppress particular applications without inhibiting the running of the Pericles service as a whole. Many of the applications accessible through live Pericles services were dependent on files which were also subject to a large amount of overnight batch running. Sometimes this running was merely concerned with re-organising to optimise the file for online access; sometimes with actually applying updates to files which in online mode were only subject to enquiry access. Often the overnight runs were concerned with the generation of voluminous print-outs. But if, for any reason, the overnight procedures did not run to time, the system controller needed the ability to bring up the Pericles service with yesterday's version of the tardy file, and to generate a suitable warning to users of the affected application. Later on, he needed the ability to suspend the application, to close yesterday's file, to assign and open today's file, and to re-activate the application. All of these facilities were available.

Pericles also offered three 'standard' applications – SCRATCHPAD, DATAPAGE, and I-SPY. Of these, SCRATCHPAD was perhaps the simplest. It simply provided to each user of a Pericles service a number of 'pages' in a personal segment of a common file, which could be used in whatever way the user chose. This proved to be a very useful medium for a form of prototyping of potential new online applications, since the user could try out the effect of various screen formats without having to incur the considerable expense of actual application development.

DATAPAGE provided, in a more extensive and better-managed way than the old Management Terminal System, a means of generating online substitutes for the printed reports which most applications churned out in profusion. It was perhaps less vigorously exploited than it deserved.

I-SPY had been developed before the inception of Pericles, but was potentially so useful that it was rapidly incorporated into the standard service framework. It offered the first truly flexible ICL enquiry language for use in an online context. At this time, of course, the language development associated with early CAFS hardware was still confined to the laboratory; equally, I-SPY could not take advantage of the fast searching capability of CAFS hardware. But the construction of selection predicates by the boolean combination of individual comparison conditions, and the formulation of retrieval and display specifications, were in principle very similar in both contexts. Not having CAFS meant that an I-SPY transaction involving a long search had the potential to be disastrously anti-social for other users of the online service. The software therefore contained its own fragmentation mechanism, ensuring that any long search would be conducted in a series of conveniently short bursts.

DATAMAIL was a later addition to the Pericles environment, providing facilities of two kinds: it could control the distribution of requested hard-copies to the appropriate printers; and it offered a simple early form of electronic mail between users. An online user could be alerted to the fact that a new message for him had been recorded in the system. Or, when he next logged in he could be informed that the system was holding one or more messages for him.

4 Later developments

Progressively over the years, as operating system capabilities became more effective, more and more aspects of the running of Pericles services became automated. Thus a service could be automatically activated in the morning by a George 3 Timer Task. The start-up procedure checked against 'application markers' to determine which applications had completed their overnight processing and were therefore available for online opening. Thereafter the service could autonomously, at suitable intervals, check to see whether any laggard applications had subsequently become available, and make them available online.

At the other end of the day, automatic close-down procedures also became the norm. The normal system close time was held in a standing file as the basic suitable time. That time could be extended, but could not be brought forward. When the normal time came, another timer task would activate a check on the file to determine close-down conditions. If the close-down time had been extended, the checks would be postponed. At the final close-down time, ongoing activity would be monitored. Applications with no active users could be closed at once; those with late users would be put under sentence of closure; no new transactions would be accepted, but transactions already within the system would be followed to correct completion; and the applications and the system would then be closed.

The availability of statistics on system usage was an invaluable aid to tuning, and greatly helped in the justification of periodic system upgrades. For many years Pericles applications covered all the data which is the life-blood of ICL, and use of those applications grew and expanded to an extent which often astounded their initiators.

The 'justification' of computer systems seems still to be stuck in the days where the worth of developing, say, an invoicing application could be quantified in terms of the number of clerks saved. The 'value' of speedy access to online information cannot be so simply calculated, yet all those who demand such access are in no doubt of the value. Pericles systems, in this innumerate sense, were of the very greatest value to users at all levels within ICL.

5 The influence of PERICLES

This paper has spoken of Pericles services as though they were things of the past. Indeed, the first services, as has been mentioned, went live as long ago as 1975. At their peak there were five such services within ICL in the UK, and there were others in at least two ICL companies overseas.

Even today there are still three Pericles services running and doing useful work for ICL in the UK. This prettily typifies a paradox of our industry:

everything is always changing, with new products and services appearing at a hectically accelerating rate; but there are also things that work well, that don't need to be improved, and that can persist for decades.

The Pericles software was never formally adopted as a product by ICL. This is sadly characteristic of a continuing short-sightedness in the product development organisations of the Company. CIS and its predecessors GIS and IMIS have consistently built, on the sound foundation of primary software products, service standards which are a model for the industry and potentially of great benefit to other ICL users. (The only exception to this general rule was the FIND information retrieval utility. This was developed by CIS, before being adopted by Dataskil. It then became, by virtue of its functionality being obviously of universal relevance, the most widely exploited software package in the ICL 1900 user community). Three UK customers did in fact purchase Pericles; the writer believes that figure should have been nearer 300.

Pericles was the fore-runner of Hero, which performs a similar service in managing multiple online applications for ICL users of online VME services. Hero also is well worthy of much wider dissemination and implementation.

6 Acknowledgements

Acknowledgements are due to many people who took part in various stages of the development of Pericles. I would mention, in particular, Arnie Shaw of West Gorton, later of the NCC, who was the brains behind I-SPY, Richard Dimond, more recently well known for his activities in the ICL CUA, who put much effort into DATAPAGE and Jerome Tucker, last heard of with Severn and Trent Water, who did invaluable work in the communications, journalising, and recovery areas. Brian Gosling, who was in at the beginning, and was responsible for much of the system design and development, has reminded the author of many important historical details. He has been in charge of Pericles support for many years, and is – even now – enhancing Pericles services to improve their interoperability with VME applications.

Finally, there's the question of the name. It has long been agreed that PERICLES is an acronym for the PERsonalised ICL Enquiry Service, but this is a clear case of the acronym having been devised after the name had been chosen. David Dace and the author (an ex-classicist) were walking down Putney High Street, pondering what we should call this great development. He said that the names of Greek gods seemed to be popular in the industry; I said that they'd all been used; he asked what other Greeks might be available – philosophers, politicians, etc. The name Pericles obviously occurred and, bingo!, it had the letters 'icl' in the middle. The conclusion was foregone. We haven't had cause to regret it.

Biography

Hamish Carmichael

Hamish Carmichael was educated at Trinity College, Glenalmond and after two years National Service in the Royal Navy, at Wadham College, Oxford, where he read Greats, taking his BA in 1958 and MA in 1960.

He joined Powers-Samas as a sales trainee in August 1958, when that company had just created a Computer Department, with a staff of three. Experience in operating and applying punched card equipment led, via the intermediate process of plugging-up 542, 550, and 555 calculators, and after a spell as lecturer in a training school, to 'proper' programming on the ICT 1301. For this machine he developed the central time-analysis module of 1301 PERT, and made a premature, over-ambitious attempt to automate the "serial-setting" process.

In the mid-60s he was involved in the formation of Corporate Information Systems, which pioneered many of ICL's now standard techniques of requirements analysis, software engineering, and database management.

A long-term interest in information retrieval and the uses of information by senior management led him to recognise the strategic potential of CAFS at an early stage. After helping to implement its earliest in-house applications, he transferred to marketing to promulgate its benefits more widely. In recent years he has pursued two themes; the importance of CAFS hardware – with its associated INDEPOL software – for intelligence and investigative systems, and the vital necessity of protecting data in secure and sensitive environments.

He is a Fellow of the British Computer Society, a Member of the Association for Computing Machinery, the author of numerous papers and articles, and a frequent lecturer.

Book Reviews

The Corporation of the 1990s — Information Technology and Organisational Transformation by Michael S. Scott Morton (Ed), Oxford Univesity Press, (New York $24.95), ISBN 0-19-506358-9

In the mid 1980s the impact of IT on how organisations work was becoming just as interesting as the technology itself — at least for more enlightened organisations. A group of them — including ICL — joined together to sponsor the Management in the 1990s research programme at the Sloan School of Management at MIT. Taking as a "given" the pervasive and ever extending impact of IT, the study sought to discover, record and codify the "best practice" in US and European business.

An overview of the results has been published in this 330 page book. Written by fourteen of the leading contributors to the programme, it pulls together an overview and a logical path through the mass of detailed research undertaken on the programme.

That there is no pretense of being a book about IT itself is clear from the brief section devoted to the "the Information Technology Platform". Containing only one obligatory "cloud" diagram of a network, a few stick-men representing users and one trend graph it makes clear from the start that the particular state of technology at any given time should not be a concern for the reader — it certainly was not for the authors.

The book improves with a distinct change of pace and style in the chapter on Competition and Collaboration by Rotemberg and Saloner. In this, and in Venkatraman's excellent following chapter on business reconfiguration, we are taken to the leading edge of MIT's thinking on the dynamics of markets and businesses. At first reading it is difficult to see why this particular line of reasoning should have emerged from a study of the impact of IT, but by the time examples have been given of competition and reconfiguration within the IT industry itself, the basic argument — that access to and the control of information are power points in many markets — becomes convincing.

Taken together the next two chapters — by Venkatramen and ICL's own Hugh Macdonald — begin to show how management can respond to these market forces — and even proactively shape them. Venkatraman's chapter presents an extremely powerful model of the different ways in which IT can be managed as an integral part of business reconfiguration, while Macdonald's chapter and appendix (the latter curiously relegated to the rear of the book — presumably because it is practical and useful rather than "academic") present a process methodology which shows organisations how to manage the "strategic alignment" of business strategy, organisation, information systems and technology.

In the final three chapters the book reverts to a more descriptive style; while there may be gems of wisdom hidden herein, all that is superficially visible is an academically upmarket version of a business magazine review of experiences of organisational change.

Taken as a whole the book betrays its origins: it is a slightly edited version of the final report to the programme sponsors, which itself seems to have been compiled as an afterthought to the real academic business of publishing the outputs of the thirty two individual research projects.

For the reader trained in engineering management the middle three chapters are quite exciting; they offer the possibility of an "engineering" approach to competition, business management and organisational development. The book is well worth reading for these chapters and associated appendices alone. Using this approach must require an organisation to have investment and change management soundly based on process competence and rational logic — no mean requirement in the era ofthe "one minute manager". The book gives no indication of whether the programme sponsors are equal to this challenge, nor whether they have gained any benefits from their considerable investment in this facinating programme.

C.L.F. Haynes
Director, Advanced Technology
AT&T Istel Limited

CALS: An Introduction to CALS, the Strategy and Standards by Joan Smith, published by Technical Appraisals at £35.00.

CALS (Computer-aided Acquisition and Logistic Support) is a US standards initiative for document and data exchange in the context of military Integrated Logistics Support. It is a set of current or proposed international standards for interchange of text, diagrams and drawings. To achieve the standards involved there are two phases of implementation of CALS.

An initial phase (Phase 1) replaces paper and the other tangible documentation with electronic media. Compliance with phase 1 was mandatory for all technical submissions to the (US) Department of Defense on January 1 1991. A second phase (Phase 2) integrates a common supplier and subcontractor database with the end-user. Data updates occur only once, since all other users up and down the chain will not duplicate that databank for their own needs but draw on it. Many problems are yet to be solved, not the least being the security of both commercial and military data.

The timescales set for implementation of CALS mean that the development of the concepts of the MIT Management in the 90's program on the integration of the complete supplier/user chain (in fact a business network) need to be understood and to be acted upon in those industries where CALS and CALS-like initiatives will be used. In this respect the results of the MIT 90 study have a key role to play. (see review of the book edited by Scott Morton in this issue).

The Phase 1 standards for each area have already been specified; for example for Text, it is SGML; illustrations, CGM; and product data, IGES. CALS is above all an evolutionary initiative. Although current standards are rigidly set, future and long term standards will be added as CALS grows and as OSI becomes more widely accepted.

The key to Phase 2 is PDES — a set of standards being developed by a consortium of US companies. Since European interest in CALS became evident the Consortium has modified its prescriptive view of PDES and allowed some inclusion of current European standards developed through appropriate programmes.

Joan Smith's book presents an easy-to-read introduction and overview of CALS. It is a useful tool for managers approaching CALS for the first time who need guidance through the CALS maze rather than IT support. In non-technical language, Mrs Smith describes the rationale for each choice of standard and summaries each without going into technical detail. Full references to the standards, publications in which they may be found and addresses from whom they may be obtained are given. Contentious issues, such as whether CALS could be used in the furtherance of unfair trade practices, or whether UK Ministry of Defence should adopt the same standards, or whether suppliers (or the military for that matter) are ready for the paperless environment are, wisely, not addressed.

Constantly flicking backwards and forwards hindered concentration; the 153 abbreviations should be on a foldout page for easy reference; the gobbledegook needs to be cleaned up ("*GOSIP is to be used as the way into OSI, FIPS PUB146 specifying a government OSI profile, known as GOSIP.*" (p59)) and the relevance of some of the necessary technical terms (X.400) should be explained. Which systems are being discussed should be make clearer. UK GOSIP and US GOSIP are not identical. Mrs Smith does not distinguish between them. This makes matters confusing for the uninformed reader. On the other hand, the end of chapter summaries and the bibliographies, the "Where to find more information" and the help sections are very good indeed.

Overall the book is eminently readable, encouragingly slim, practices what it preaches by being typeset directly from the CALS text standard SGML, but is horrifically overpriced at £35. Still, it is a recommended ‘must’ for all managers anticipating compliance with CALS to be a requirement in their business in the near future.

Graham Cooper
ICL National Accounts Division
Slough, Bucks. UK.

SGML: Standardised General Mark-up Language
CGM = Computer Graphic Meta-file
IGES = Initial Graphic Exchange Specification.
PDES = Product Data Exchange through STEP (Standard for the Exchange of Product model data)

Pages contained in each issue

(1) 1–192 (3) 447–658
(2) 193–445 (4) 659–868

Subject index
Volume 7

Author index
Volume 7

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 1986, SE-12, 2, 241–250.

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
Copyright in papers published in the *ICL Technical Journal* rests with ICL unless specifically agreed otherwise before publication. Publications may be reproduced with the Editor's permission, which will normally be granted, and with due acknowledgement.