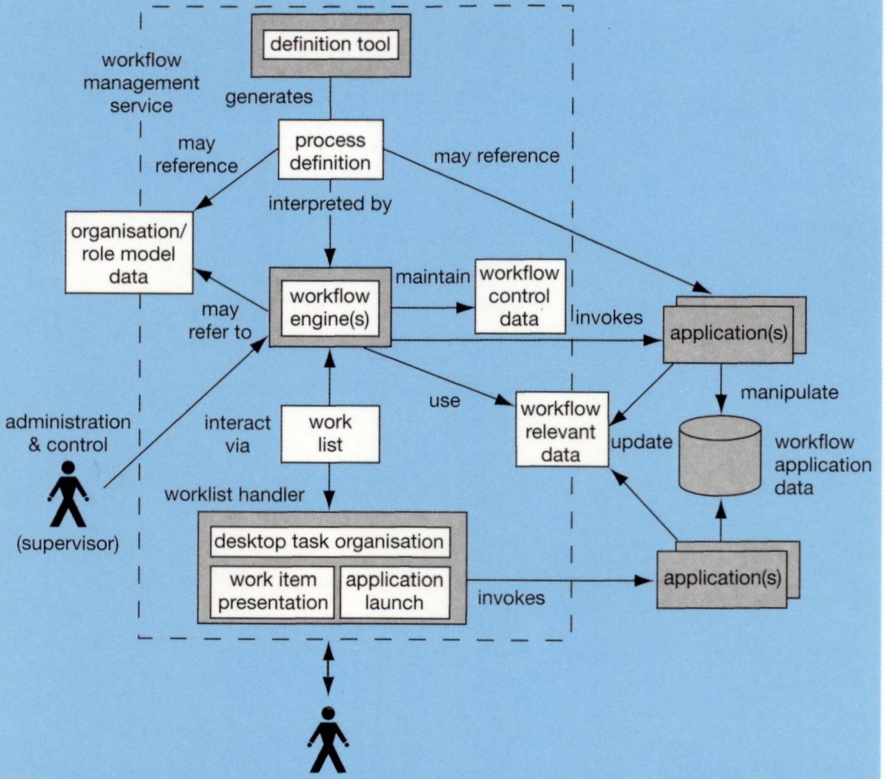
Volume 12 Issue 2 *November 1997 Edition*



ICL

The Systems *Journal*

ICL Systems Journal

Editor

Prof. V.A.J. Maller
ICL Professor
Department of Computer Studies, Loughborough University,
Loughborough, Leicestershire, LE11 3TU.

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.

Published twice a year by Research and Advanced Technology, ICL, Bracknell.

1998 subscription rates (including postage & packing):

	UK and Europe	Rest of World
Annual subscription	£72	$120
Single issues	£43	$72

ISSN 1364-310X

ICL Systems Journal

Volume 12 Issue 2

Contents

Front cover: Workflow System components. See the paper, "Workflow—A Model for Integration," in this issue.

Workflow—A Model for Integration

David Hollingsworth

Skill Centre, ICL Enterprises, Windsor, UK

Abstract

This article reviews the nature of a workflow system from a systems integration perspective, focusing on points of interaction between the workflow control software and other system components, such as process design tools, legacy applications and messaging infrastructure. The characteristics of the underlying business model and its representation to the workflow system are also discussed, including requirements for business processes to span organisational boundaries. The complexity of systems integration is identified as a major constraint on effective exploitation and indicative of the need for standards to support more effective product usage and interoperability. The article draws on the Author's experience in developing workflow related standards and concludes with an assessment of their potential impact, particularly on opportunities for their use in electronic commerce.

1. Introduction

Workflow is often seen as a key integration technology, bringing together business processes with the information to support them, and linking legacy and desktop applications into a flexible and adaptable distributed infrastructure. The external image of such systems can be deceptively simple, based on the notion that once the business process is defined, its automation merely requires the integration of a few simple tools.

According to the Workflow Management Coalition[1] [WMC, 1995–1997], workflow represents, *"the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules"*.

Whilst not explicitly stated in the above definition, a key motivation for the deployment of workflow technology is that it should provide flexibility for the business process to evolve with minimum re-engineering. This is a concept simply captured within Open*framework* [Pratten and Henderson, 1993] as *"potential for change"*.

Workflow technology typically achieves this by enforcing separation

[1] The Workflow Management Coalition (WfMC) is a non-profit making consortium of vendors, users, analysts and academia, with the goal of developing standards for workflow systems operation and promoting knowledge of the technology within the industry.

between:

- the definition of the various activities within the business process and their data requirements
- the business rules governing the flow of control between activities within the process
- the roles and responsibilities associated with the work undertaken within the process activities
- an underlying organisational model, which relates roles and responsibilities to the actual work performers

In theory any aspect can change independently by simple amendment of the relevant control parameters, without affecting the ongoing operation of any other aspects of the process.

Despite this apparent utopia, the reality in many workflow systems implementations has been much more earthly - substantial systems integration issues to be faced in bringing together the component systems elements, lack of interoperability between different systems, major cultural and organisational issues to be resolved in the introduction of new working practices, and so on. Once operational, many systems prove less adaptive than expected to the future organisational or business changes.

2. Integration Requirements

Reliable statistics from within the industry are not always easy to find. However a recent market survey undertaken by the Workflow Management Coalition indicates that for virtually all workflow systems, integration with other industry software is vital - and a major cost component of implementation. At the time of writing the full survey results are still being collated but preliminary findings are as follows:

Integration Requirement	% of Respondents
World Wide Web	89%
Java Applications	75%
Legacy Applications	73%
CORBA based infrastructure	69%
Security Services	66%

Other technologies frequently mentioned included Business Process Modelling tools, Document Management and Imaging systems.

Informal estimates have indicated a ratio between workflow software implementation and overall project integration costs of between 1:5 (for ad-hoc office based systems) to 1:7 or more (for highly structured production workflow applications). Even in an industry where integration is increasingly the major cost component in the introduction of new technology, these are high figures.

This informal view is supported by a recent study from Ovum Group [Ovum Group, 1996], which shows workflow vendor revenues split between product and service currently in the ratio of 1:4.5, with a fall projected by end 2000 to 1:3. This is consistent with an increasing degree of standardisation, de jure or de facto, and of product consolidation leading to simpler integration.

The following sections consider two factors, the fragmented emergence of workflow within the market and the technical complexity of product interfaces, which have contributed to this cost.

3. The Evolution of Workflow Technology

One of the reasons for the complexity of the systems integration task is the fragmented way in which workflow technology has developed in the market.

3.1 Workflow—the first phase market

Software to control process operations is not a new concept. Many types of product in the IT market have supported certain aspects of workflow functionality for a number of years, although often embedded within other, related products rather than as a technology in its own right.

Image Processing

Workflow has been closely associated with image systems and many image systems have some workflow capability built-in. Once paper based information has been captured electronically as image data, it is often required to be passed between a number of different participants for different purposes within the (previously paper based) process.

Document Management

Increasingly, the management of electronic documents has included facilities for routing documents (in whole or part) between individuals and repositories, for example to facilitate shared authoring or filing services. Standardisation within the document management area has already recognised the requirement for extensions into workflow[2].

Electronic Mail & Directories

Electronic mail provides facilities for distributing information to individu-

[2] The Open Document Management Association (ODMA) first identified an API for simple workflow functionality in 1995. More recently the DMA (Document Management Association, representing major vendors of document management software) has entered discussions with the WfMC to address the integration of workflow and document management standards.

als; associated directories provide a means of recording information about user attributes, such as organisation roles or other attributes relating to business procedures. Through the addition of routing mechanisms to define a sequence of recipients for a mail item, electronic mail systems have themselves been progressing towards workflow functionality.

Groupware Applications

Groupware applications are designed to support and improve interactions between groups of individuals. Initially many of these applications supported improvements in group working via informal processes, accessing group bulletin boards or diary/scheduling applications on an ad-hoc basis. As the scope of such applications has spread towards more formal business processes there has been an increasing move to incorporate workflow into work-group software[3].

Project Support Software

Software to handle complex IT project developments has often provided a form of workflow functionality within the project environment, for the sequencing and routing of development tasks between individuals and routing information between individuals to support these tasks.

Transactional Workflow

As traditional TP applications have become more distributed in nature some have moved to fully distribute transactional tasks to desktop environments. In parallel workflow vendors have been adding transactional characteristics to workflow systems, particularly in the area of task commitment and recovery co-ordination. In these situations there is an increasing degree of overlap between the two technologies.

BPR and Structured System Design Tools

Whilst workflow has been emerging as a fragmented technology, Business Process Re-engineering tools have appeared in significant numbers. These provide IT based support for analysing, defining and modelling the business processes of an organisation and the potential effects of change in such processes or organisations. The use of such products forms a natural precursor to workflow implementation.

In summary, there are now many products in the market providing workflow capability[4]. Such products are often derivatives of products from other market areas, incorporating elements of workflow technology in an incompatible manner, making integration costly and negating the "poten-

[3] Examples of this include the integration of Lotus Notes product with several workflow packages and the introduction of Fujitsu/ICLís TeamWareFlow as the workflow component within Team Office.

[4] Of the WfMC membership, there are approximately one hundred different organisations which categorise themselves as product vendors.

tial for change" factor.

3.2 Workflow—the second phase market

The GIGA group recently presented an interesting analysis of the development of workflow technology and concluded that the industry is entering the second phase of automation [GIGA Group, 1997]. Most of the first phase automation projects have been at departmental or workgroup level, with relatively little co-ordination[5]. The continuing business pressures of globalisation, contracting and electronic trading are leading organisations to reassess their business processes at enterprise level with ever increasing frequency.

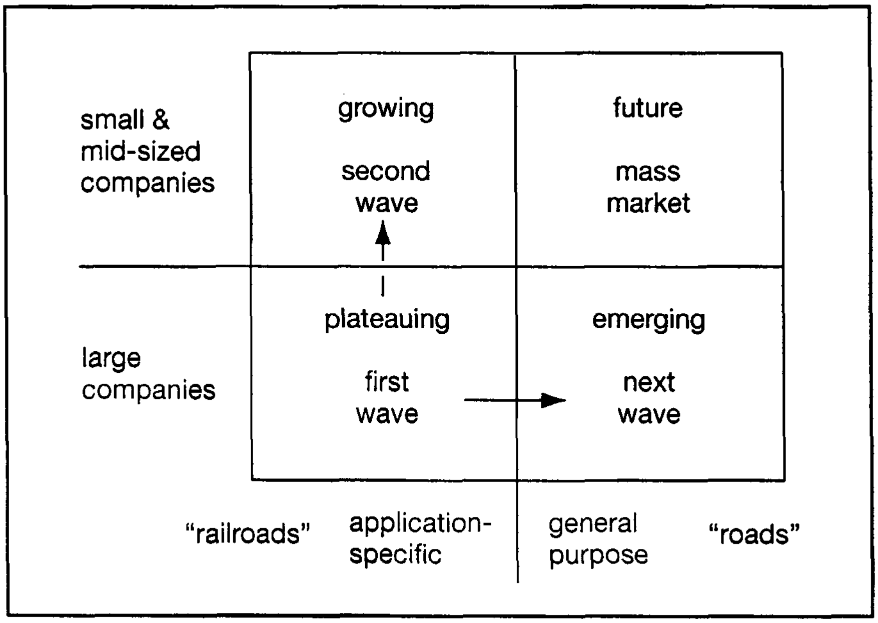


Figure 1: Workflow & the shift to IT Infrastructure (Giga Information Group, adapted from CAP Ventura)

The GIGA view is that, *"second phase, messaging based enterprise-wide workflow will be dominant in 1–2 years time"*. This will see workflow positioned as general purpose middleware across the enterprise. Electronic trading between organisations will increasingly push workflow into smaller and medium sized organisations, leading towards ubiquity of use.

This prognosis, however, depends upon the consolidation of the indus-

[5] At a recent conference a large multinational organisation indicated that of the eight workflow applications implemented to date all were incompatible in terms of interoperability and use of common infrastructure components.

try around a cohesive set of standards to support integration and interoperability, considered in Section 6.

4. Workflow and Software Integration

Integration complexity arises from the requirements of most workflow systems to interact with numerous other software components, ranging from standard desktop tools such as forms, spreadsheets and word-processors, to server applications such as document repositories and legacy applications, often based upon TP technology.

A key aspect of many workflow system is the incorporation of an organisational model, enabling workflow procedures to be defined relative to organisational roles and responsibilities. These may be separately maintained, for example by means of a directory subsystem, with associated role privileges.

Workflow systems may also require integration with process definition and modelling tools so that a proposed system can be fully specified and simulated prior to introduction.

Finally, as with any distributed application, integration with the underlying infrastructure (Electronic Mail, Object Request Broker domains, etc.)

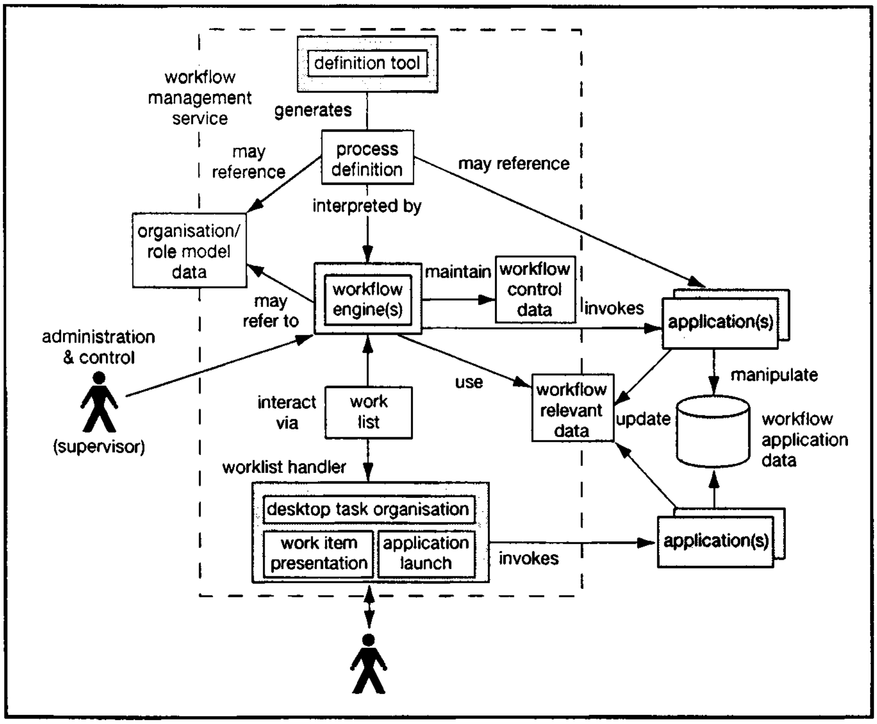


Figure 2: Workflow System components

is a further requirement.

The schematic, shown in Figure 2, gives some indication of the potential components and points of integration of a typical workflow system.

Several different systems construction paradigms exist within the industry. Common models include:

- Object based, using CORBA as the main distribution mechanism
- Electronic mail based with autonomous desktop environments
- Centralised workflow engine with tightly coupled desktop task management
- Document-centric with shared repository

The boundaries of the workflow management software are often unclear, for example some vendors include a directory component or interfaces to access legacy systems, others see this as part of the system integration task.

5. The Characteristics of the Business Process

Workflow is a process centred technology. To quote from Koulopoulos [Koulopoulos, 1995] of the Delphi Group (a Boston based workflow consulting group): *"Workflow emphasises the importance of the process, which acts as a container for the information. ... This is a process-centred model, as opposed to an information- centred model."*

5.1 The nature of the business model

Although the majority of workflow systems have tended to automate administrative processes (the so called "paper factory") in an essentially human environment, there are often certain activities which are wholly automated by software components. In the manufacturing or process industries many activities are fully automated with little or no human involvement. In this context the specification of the work "performer" for a particular activity must incorporate the concept of machine automata.

Various characteristics of a business process need to be considered.

5.1.1 Responsibilities

"Ownership" of a business process is often an alien concept, but once an electronic representation is achieved, this becomes an important attribute of the process, if only to determine who has permission to modify the process and under what circumstances.

The realignment of business thinking from organisation to process marks a major shift in organisational culture, likened by Giga Group to the *"dismantling of the industrial age"*[GIGA Group, 1997].

This emphasises the move away from functional organisation towards virtual teams and processes supporting business collaboration.

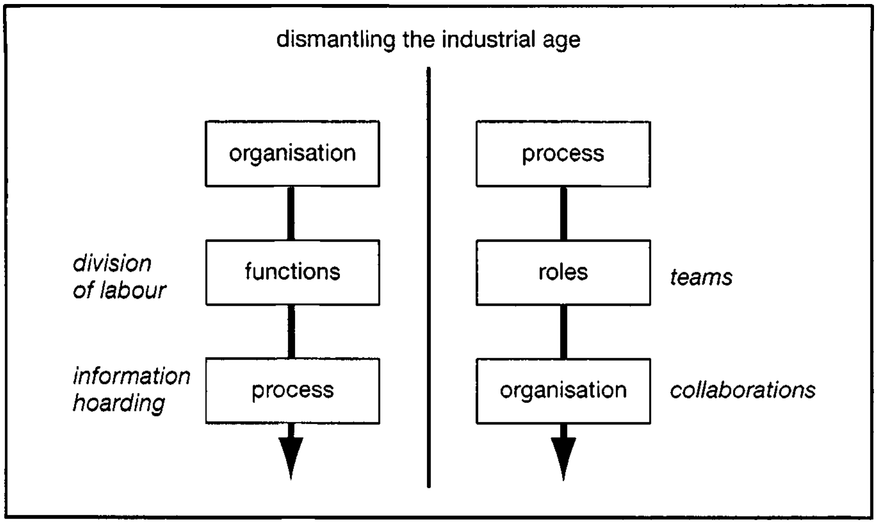


Figure 3: Dismantling the Industrial Age (source: ICL & Nortel)

This emphasises the move away from functional organisation towards virtual teams and processes supporting business collaboration.

Since responsibility is increasingly defined in terms of role rather than person, there is a requirement for workflow systems to maintain an audit trail of the work performer who actually undertakes a particular activity. In some cases this is complemented by a supervisory role for individual activities or the overall process which is invoked if various criteria (for example deadlines) are not met. The concept of responsibility also needs to cope with activities which are wholly automated with no human involvement (for example by IT application).

5.1.2 Process Modification

Adaptive processes are fundamental to the ongoing value of workflow; in practice adaptation can occur in several ways with different associated complexities of automation.

(i) An ongoing change to the process, introduced by the owner. An example might be the introduction of some additional checking on an authorisation task, or amendment of the value at which additional checking is undertaken. The changes may need immediate application to all existing open business cases, or may just apply to new cases.

(ii) A variation to the normal process behaviour may be pre-defined under certain conditions, for example an activity may be skipped or delegated to a subordinate role if a certain business criterion is met. This variation is defined as part of the persistent business rules applied within the process behaviour, but needs to be separately monitored (and reported) in each case.

(iii) In some cases process behaviour (i.e. the business rules) may be arbitrarily adapted or developed during operation, without any prior constraints imposed when the process was designed. This is a behaviour pattern often associated with ad-hoc workflow systems in co-operative workgroup situations, where only a skeleton process may be defined within the process definition. This is amended dynamically as process execution proceeds to add new tasks or amend responsibilities, etc.

5.1.3 Process Structure & Organisational Boundaries

One consequence of the flattening of organisational structures and increasing business integration across organisations is that process scope is extending not just across departmental boundaries but also between enterprises. A model of workflow put forward by the Japan Standards Association (JSA) Groupware Committee (1997) illustrates this industry direction by a three tier framework embracing workflow at departmental, enterprise and inter-enterprise levels.

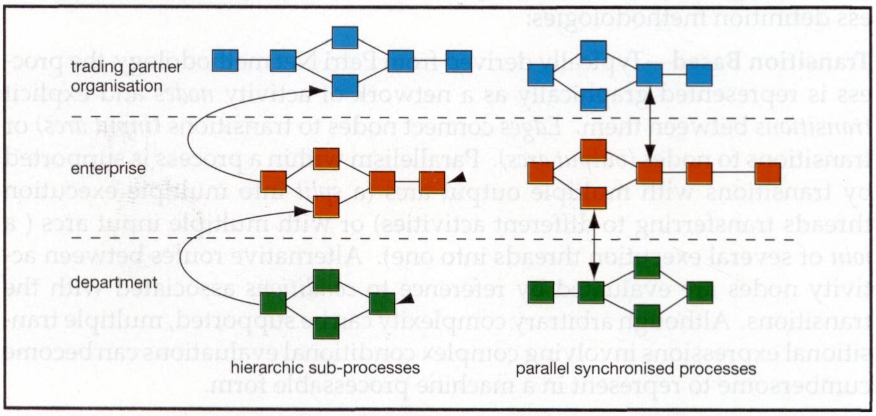


Figure 4: Inter-organisation Business Processes

This leads to requirements to support process structures which:

- enable a sub element of a process to be initiated in a different organisational domain (hierarchic or chained sub-processes)
- support the periodic synchronisation of process activity between two (or more) essentially independent processes operating in different domains (parallel synchronised processes).

Such process models often impose additional constraints on automation in the areas of security between domains and conventions for object naming and organisational mapping.

5.1.4 Process Duration

This impacts a number of engineering issues, particularly the likely concurrency of active process instances and possible requirements for the support of a dormant process state. Most typical processes have a relatively short duration, typically from seconds to weeks. Some, which are customer-centric, may be defined in terms of a customer life cycle lasting many years. Since most workflow systems carry a significant overhead per process instance there may be a requirement is such cases to remove dormant cases to some form of secondary process data storage.

5.1.5 Activity navigation

One characteristic of all business processes is the thread of control which links together the various activities during the life of the process instance. Typically this involves conditional logic and a number of alternative routes (navigation paths) through the process. These paths generally need to support a mixture of sequential and parallel activities within a process.

This logic may be defined in quite different ways within different process definition methodologies:

Transition Based—Typically derived from Petri Net methodology, the process is represented graphically as a network of activity *nodes* and explicit *transitions* between them. *Edges* connect nodes to transitions *(input arcs)* or transitions to nodes *(output arcs).* Parallelism within a process is supported by transitions with multiple output arcs (a *split* into multiple execution threads transferring to different activities) or with multiple input arcs (a *join* of several execution threads into one). Alternative routes between activity nodes are evaluated by reference to *conditions* associated with the transitions. Although arbitrary complexity can be supported, multiple transitional expressions involving complex conditional evaluations can become cumbersome to represent in a machine processable form.

Block Structured decomposition—In this approach any single node in a model may be decomposed to a lower level of underlying process (a paradigm based upon the hierarchic sub process model). In this approach parallelism is constrained to operate only within the context of a single level of decomposition (i.e. parallel threads cannot transcend block boundaries). A product based upon this approach cannot cope with an arbitrary complexity of split and join constructs (for example an unbalanced split where one path continues beyond the context of the current block).

Activity Pre- & Post-conditions—In this approach no explicit transitions between activities are declared. The process is defined as a set of activities each having entry *(pre-)* and exit *(post-) conditions*; parallelism is implicit and when pre-conditions are met the activity is initiated, independently of the status of other activities within the process. To provide sequential operation, pre-conditions may relate to the completion of a particular prior activity (and by extension to multiple prior activities, providing an *"and-join"* capability). Post-conditions may be used to control looping within an activity.

Each of the above approaches has its pros and cons and its own particular devotees. The problem for the systems integrator is that it is not easy to transfer process information between design tools and/or workflow control software based upon the different design paradigms.

5.1.6 The Organisational Model

Virtually all business processes are based around the concept of an individual's roles and responsibilities for the various activities within the process. As far as possible processes need to be isolated from the vagaries of organisational change, leading to the requirement for a (dynamic) organisational model. This can map the ongoing roles and responsibilities at process level against the current organisational entities and the current set of individuals who undertake the various roles.

The identification of an activity "performer" within a process may embrace a mixture of organisational and role information (... "the fault analyst in the European Customer Support Unit"). Organisational relationships often expressed include:

- manager of
- deputy to
- alternative to (proxy)
- role or skill profile.

Responsibility models may introduce additional constraints on work performers (for example .."not the person who authorised the original loan"), which require process history to be maintained.

Thus in many cases an organisational model will need to accompany a business process to enable its automation.

5.1.7 Security

Security is often developed separately from the business process and may have to added at automation stage by reference to a separate organisation security policy document. Many of the security requirements during automation will be related to roles, responsibilities and authority within the process.

5.2 Representing the Business Process

In order to provide automated support for a process, it must be first be captured in a machine interpretable representation. This representation must have the flexibility to structure and maintain all the process related information necessary to enable co-ordination of enactment using IT infrastructure.

The WfMC glossary introduces the term *"Process Definition"* for this representation, describing it thus:

> *"The automation of a business process is defined within a Process Definition, which identifies the various process activities, procedural rules and associated control data used to manage the workflow during process enactment".*

The process definition may be represented by a combination of any or all of textual script, graphical notation, or formal programming notation, with many different process development tools available to manipulate such information. Typically their use follows a cycle of analysis, modelling, implementation, feedback and further analysis.

Several attempts have been made to define a standard representation of all or part of a process specification.

IDEF [IDEF, 1997] is a series of modelling notations introduced by the US Air Force, several of which are published as FIPS by NIST. Included are methodologies for modelling business functions (IDEF0), information models (IDEF1X) (both widely used), dynamic system behaviour (IDEF2) and Process Description Capture (IDEF3).

CDIF [CDIF, 1997] has defined a core architecture for CASE tools and data interchange bindings, based around a meta-meta-model. Foundation and Common meta-models are defined and work has been completed on data definition, data flow and data modelling. An extension to cover business process modelling is under discussion, but work is not yet mature. UML (unified modelling language) is a similar initiative under the auspices of the OMG, with its own modelling notation and meta-model.

None of the above currently provide a machine processable process definition as a basis for workflow automation.

PIF (Process Interchange Format & Framework) [PIF, 1997] has been developed by a working group drawn from a number of US and UK universities. Its underlying philosophy is that of generality over computational efficiency; this is reflected in the organisation of its entity classes which is not necessarily well suited to the performance of any specific task, such as workflow management or process simulation. It has been used for experimental translation of process related information within the research group. As with other process representations it has been found necessary to structure into a minimal core set with add-on classes. PIF is designed to be machine processable, but is not specialised to the entities and attributes required for workflow.

The **NIST PSL** (Process Specification Language) group [NIST, 1997] is a study group formed by NIST in April 1997, working towards a common process specification language, rather than interchange format. It has members drawn from industry, government and academia but has a particular interest in the application of process technology to manufacturing industry. There is no current specification produced by the group, but it is reviewing inputs from other industry organisations.

WPDL (Workflow Process Definition Language) [WMC, 1995–1997] is specified by the WfMC and despite its name was conceived as a text-based, machine processable interchange format, rather than a definition language.

The WfMC has produced a process definition meta-model, shown below, which attempts to capture the highest level objects and relationships which, as a minimum, must be defined to support process automation. This meta-model underpins the WPDL grammar.

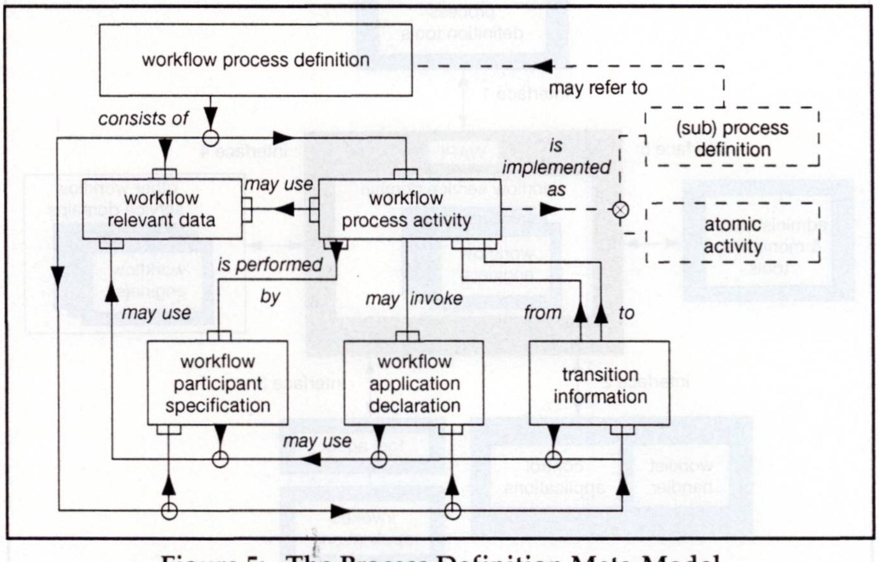


Figure 5: The Process Definition Meta-Model

The route followed by the WfMC is to define as standard attributes the most commonly required properties of these top level objects, but to allow extensibility through an extended attribute list and library functions within the WPDL grammar.

The model and the WPDL constructs are focused specifically on workflow and provide more detailed structures for defining the workflow related aspects of a process. They do not attempt to incorporate the level of generality of other approaches such as PIF. WPDL will shortly be released in beta form and several prototype implementations have been made against interim specifications with reasonable success.

One key difficulty with all approaches remains the capture of all the potential dynamics of a business process within a single model. It is probable that automatic translation of 100% of such business processes between different products is an unreachable goal in the foreseeable future. However the meta-model provides a structure for mapping a large part of the business process into WPDL or, potentially, other interchange forms.

6. The Systems Integration Model

The WFMC is the principal organisation defining standards for workflow and is attempting to cover much of the ground discussed in earlier sections. The standardisation programme is based upon the "Reference Model" [WMC, 1995–1997] shown in Figure 6.

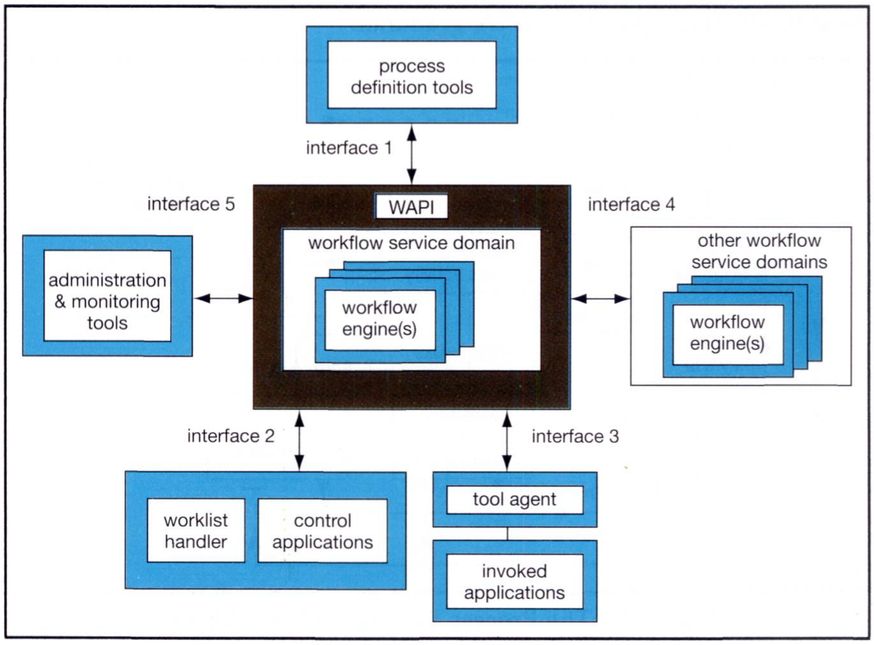


Figure 6: The WfMC Reference Model

Whilst this is an oversimplification of workflow from the integration perspective, it has proved useful within the industry as a focus for standardisation work. It concentrates on modelling a workflow service as a black box object viewed from its external interfaces, whilst ignoring the internal construction architecture (and hence a number of the integration problems).

The internal components of the *"workflow service"* are assumed to be homogeneous, and typically supplied by a single vendor's product(s). This avoids issues associated with service administration and security, which essentially lie inside the *"box"*. Also no distinction is made between a single centralised *"engine"* and co-operating, *"distributed engines"*, which need to support shared process state data in order to support a single homogeneous service image. The model also attempts to avoid dependence on the nature of the underlying distributed infrastructure through the specification of APIs, or interchange formats, by which system components interact, which are assumed to be supportable through the infrastructure[6].

Five "interfaces" are identified within the Reference Model, realised by a combination of APIs, protocol and format conventions.

6.1 Process Definition Interchange

The purpose of this interface is to support the exchange of process definition information between BPR tools, workflow systems, and process definition repositories. The interface is based upon the meta-model described in Section 5; information exchange is supported in two ways:

1. the WPDL grammar supports the transfer of complete process models via file transfer, typically using an import/export mechanism from native product formats. The import process can check the process model for structural integrity, for example flagging isolated activities with no transitions. The export process must flag any structures which cannot be represented in WPDL

2. APIs are defined for reading and writing individual objects and attribute data within the Process Definition. These are typically used for ad-hoc process modification or control functions, rather than bulk process transfer. There is no automatic mechanism for checking the integrity of the resultant modified process.

6.2 Client Applications Integration

This provides a standard interface for work allocation to the desktop environment, allowing desktop applications portability and re-use across different workflow environments. APIs are defined for:

1. Process & Activity Control functions, for example starting, suspending, terminating a process instance or sets of process instances

2. Worklist Handling, to allow users to log on and process (or re-assign) individual work items.

[6] One exception to this is the interoperability protocol between workflow domains, which is discussed later.

6.3 Applications Invocation

This provides a single interface which may be used for two purposes:

1. To provide a common framework for the integration of software agents providing access to other industry services such as document repositories, meeting schedulers and email which use their own specific industry APIs
2. To support access to legacy applications via application specific methods (for example terminal emulation or proprietary TP protocols).

A simple API set supports Connect/Disconnect, Invoke Application, Request Status and Terminate Application.

6.4 Process interoperability

This supports the remote invocation of a sub-process on a different workflow system, allowing a single business process to be implemented over two or more workflow systems.

Two variants of interchange protocol are defined:

1. MIME (Multipurpose Internet Mail Extensions) defined in RFC 1341
2. IDL bindings for use with CORBA (typically via ORB interoperability services).

This interface uses essentially the same API set as that for process initiation from client applications[7].

A missing element in the current specifications is support for synchronisation points between parallel execution threads; this is identified for future development.

The sub-process interoperability model as currently specified makes no requirement to dynamically share state data between the two interoperability domains and specifies a minimal level of prior co-ordination. (This is essentially limited to a knowledge of the called address for a particular sub-process and any related security attributes.) Thus it is more suited to "loosely coupled" distributed process enactment across different organisational entities than tightly bound workflow systems within a single department or workgroup.

Some issues of detail still remain; for example, which properties of sub-process operation are inherited from the superior calling process and which from the local process definition. In general, it is an accepted principle that where remote process "hand-off" occurs it will not be feasible to retain all process attributes through the call and return. Details of name space usage across the two environments also remains to be fixed in detail.

[7] Since remote invocation can occur via an asynchronous interface such as e-mail some additional optimisations are provided to allow grouping of calls (or call responses) into an underlying MIME transfer.

6.5 Audit and monitoring

Auditing is an important requirement for many workflow systems. This "interface" comprises a specification of standard audit events and their recording format, thus enabling the integration of audit trails across different systems during workflow interoperability. The means by which audit data is accessed or retrieved on any particular system is undefined but is typically via SQL for many workflow products.

APIs are also defined to retrieve status information on current process instances or activities.

7. Ongoing Standardisation Work

The standards currently defined will make a significant contribution to workflow systems integration—provided they are adopted by product vendors. An important indicator of intent is the current OMG standards approval cycle for workflow technology [Object Management Group, 1997], in which the WfMC standards are currently supported by more than thirty-five organisations.

Various important extensions have been identified to improve the potential of the model as a basis for integration.

Object Integration—the work with OMG has identified potential requirements for developing the architecture "internal to the workflow manager", to facilitate the integration of other complimentary OMG services such as OTS (transaction services), naming, security and versioning, etc.. This approach can support closer integration between different workflow products where all use the same underlying object services architecture. There is also interest in positioning workflow within the OMG *Business Objects Framework* to identify reusable service elements which can be consolidated into a business application environment.

Security—the approach here is to specify how existing security standards should be applied in the context of workflow. The most important area is seen to be the use of authentication, integrity and confidentiality services applied to workflow interoperability, particularly between domains in different organisations.

Support for Event synchronisation—Event synchronisation represents a significant extension of the interoperability model to support transitions (and potentially associated data flow) between different, essentially independent processes, running in different domains. Issues to be resolved include process, thread and event naming and event management functions applied across distributed, heterogeneous products (e.g. to detect and prevent deadlock and persistent wait states).

Process Integrity and recovery—This is an area which has not been widely addressed and will take some time to mature. Recovery may require the basic process state data, shared workflow and application data and wholly application related data (for example within legacy applications). Different techniques include 2-phase commit and rollback (whose use may be impractical through asynchronous messaging infrastructure and/or long transaction times), compensating transactions, or alternative transactions. Many products rely on at least some manual recovery elements.

Internet and electronic commerce—There is considerable interest in support for inter-organisational workflow functionality carried via the Internet. Existing functionality via electronic mail will be augmented by support for more dynamic process binding (for example using traders or yellow page services). The use of XML[8] to encode process based exchanges is also under discussion.

8. The Future

We shall not know for a year or two whether this standardisation programme will really contribute to the integration task. There are encouraging signs that the industry has recognised the benefit of a common architectural framework to assist with product interoperability and most products are structured in broad alignment with the reference model. In practice the number of conflicting products is bound to fall, if only through market consolidation. Interest continues in capturing and reusing automated process fragments within an applications framework architecture.

The notion that workflow will evolve into ubiquitous middleware, in the same way as, say, electronic mail, is perhaps more questionable. This requires both standardisation and a re-orientation of commercial thinking towards the value of automated processes. There is certainly every likelihood that workflow interoperability will substantially increase in inter-company trading situations. A demonstration of an automated supply chain process scenario [Anderson, 1997], in which the overall business process was automated across seven diverse organisational systems, attracted huge industry interest[9].

Within this style of operation it is possible to enact business processes which automatically call other organisations to implement those parts of the process which lie within their domain of responsibility, for example manufacturing, wholesaling or supply logistics. Such business interactions go far beyond the simple transfer of order data or supply notes, bringing opportunities for expressing a complete supply chain business logic in a

[8] XML (Extended Markup Language)—a more generalised version of HTML, also derived from SGML principles.
[9] Workflow Canada, Toronto, June, 1996 and repeated at the Giga Workflow conference, Amsterdam, October, 1996.

manner which can be seamlessly automated across diverse business entities. This may well point the way towards a second generation of electronic commerce based on process interoperability rather than simple electronic data interchange.

Acknowledgements

Many colleagues within the WfMC have contributed important ideas to the subject of workflow integration and interoperability. Particular thanks are due to Mike Anderson, from the TeamWare Integration Centre, who provided comments on this paper and was also responsible for the WfMC interoperability specifications.

Bibliography

WORKFLOW MANAGEMENT COALITION; Documentation: Glossary, 1996; The Workflow Reference Model, 1995; Workflow API Specification, 1995; Workflow Interoperability Specification, 1996; Process Definition Interchange Specification (draft), 1998. Details available via http://www.wfmc.org

PRATTEN, G.D. and HENDERSON, P., "Creating Potential-for-Change," ICL Technical Journal, Vol 8, Issue 3, 1993.

OVUM GROUP, "Ovum Evaluates Workflow," 1996.

GIGA GROUP, "Business Process & Workflow," Conference Proceedings, London, 22–24, October, 1997.

KOULOPOULOS, T., "The Workflow Imperative," Van Nostrand Reinhold, 1995 (ISBN 0-442-01975-090000).

IDEF, (Integrated Computer Aided Manufacturing Definition), details of IDEF0, IDEF1X, IDEF3, plus work in progress available via the IDEF home page http://www.idef.com.

CDIF, (CASE Data Interchange Format) specifications available via http://www.cdif.org.

PIF (Process Interchange Format), details available via PIF home page http://soa.cba.hawaii.edu/pif/

NIST Process Specification Language (PSL) project documents: Proceedings of the First PSL Roundtable, NISTIR 6081, National Institute of Standards and Technology, Gaithersburg, MD (1997). Unified Process Specification Language: Requirements for Modelling Process, NISTIR 5910, National Institute of Standards and Technology, Gaithersburg, MD (1996). Both available via NIST PSL home page http://www.mel.nist.gov/psl/

OMG, Object Management Group, Framingham, MA 01701-4568, RFP for

Workflow Technology, 1997 and associated proposals, details via OMG http://www.omg.org/library/schedule/Workflow_RFP.htm

ANDERSON, M.J., "Workflow Engine Interoperability—What's in it for Users," Document World (May/June 1997).

Biography

David Hollingsworth has spent in excess of 25 years in the IT industry. His career with ICL spans roles in product development, market requirements, strategic planning, systems architecture and major projects consultancy assignments. His interest in workflow systems dates from 1992 and as ICL architect working on future office systems he was involved in the establishment of the Workflow Management Coalition as the industry standards body for workflow. He is currently chairman of its Technical Committee and has authored several of its reference documents. He holds an honours degree in Economics from the London School of Economics and is an ICL Distinguished Engineer.

SuperVISE—System Specification and Design Methodology

S. Hodgson and M.M.K. Hashmi

ICL High Performance Systems, Manchester, UK

Abstract

This paper describes the system design methodologies and tools being developed by ICL under the name ***SuperVISE*** [ICL/WWW, 1997]. The paper covers the origins and principles of the methodologies, and explains the benefits of the ***SuperVISE*** tools which support these methodologies. The ***SuperVISE*** language **VHDL+** is introduced and the key features of the ***SuperVISE*** products are described. ***SuperVISE*** won the 1997 ICL Innovation Award for Technology.

1. Introduction

The creation of the ***SuperVISE*** methodologies, languages and tools grew from the basic need to 'Improve Time-to-Market' for large electronics systems which include significant hardware design content. The requirement was for more than just an incremental improvement in conventional design methodology. What was needed was a 'step function' improvement, which would cope with the increasing size and complexity of the systems expected over the next 20 years and beyond.

These requirements came initially from the mainframe development group at the ICL High Performance Systems Division in Manchester. The mainframe systems designers were about to embark on a major new design—significantly larger and more complex than anything developed before at ICL—a design that turned out to be a system of more than 10 million gates. The design had to be complete in less time than the previous, smaller, mainframe design and with less engineers.

To meet these ambitious timescales, the whole system design had to be 'right first time'. There would not be time for redesigns or chip iterations, so a major improvement was essential in the design verification phases.

The good news is that the design mentioned has been completed—the ICL Trimetra (SY) [Allt et al., 1997]. The methodology and tools achieved their objectives and, in fact, exceeded the ambitious quality targets set at the start of the project. The methodology used (originally called CHISLE [Jebson et al., 1993], but since extended and now called 'the ***SuperVISE*** methodology') is now being introduced across the Fujitsu Group and promoted externally.

Having defined the methodology and the language extensions, it was recognised that the introduction of these new concepts would take time. So the ***SuperVISE*** User Group was formed to bring together a set of companies, individuals and universities who together have the objective of developing and establishing ***SuperVISE*** as a major step forward in system design. Active members of the User Group include Ericsson, Nortel, Alcatel and, of course, various members of the Fujitsu Group.

2. Requirements

2.1 Business Needs

Design capture and verification at the earliest possible stage increases the commercial viability of a product by reducing the overall time to market.

One way to compress and effectively manage the complete product timescale is to formally capture the design earlier in the design cycle so that it can be verified earlier, and problems conforming to the requirement specification solved before reaching the later phases of the design cycle. ***SuperVISE*** makes this possible by introducing powerful specification-level features within a VHDL design environment.

Ensuring the correctness of a design early on in the design cycle not only improves the specification phase, but also has a beneficial 'ripple' effect on later stages. This is magnified by the growing complexity of the design; see Figure 1.

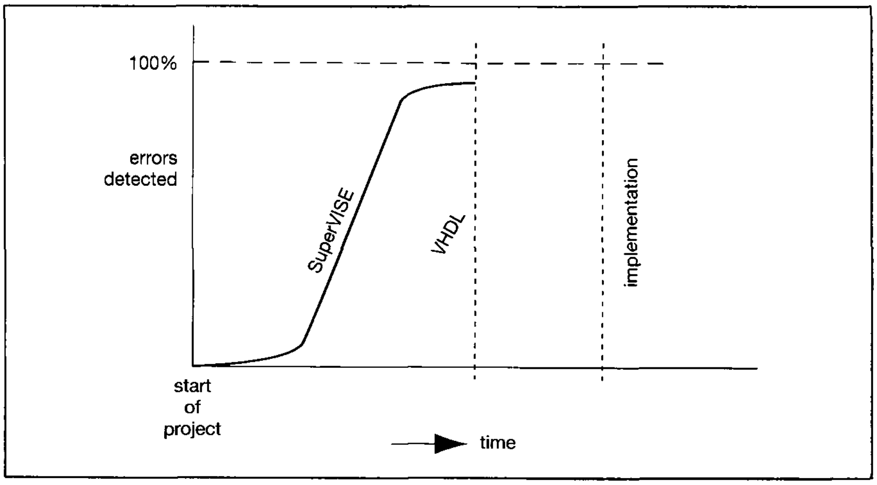


Figure 1: Benefits of early error detection

Traditionally, in the electronics industry, system specifications and requirements have been captured as natural language documents. These specifications are then refined and elaborated in a series of manual design steps until they are detailed enough to be captured formally in schematic or textual form for simulation.

Normally, the first simulatable level that is formally captured is at Register Transfer Level (RTL) or below. Once captured, the design can be verified by simulation with test patterns or a test harness. It is then refined until it can be synthesized or manually translated into a gate level description.

SuperVISE brings the process of design verification up to the specification level, removing the conventional design 'gap', by supporting multi-level simulation from *concept* through to *implementation*; see Figure 2.

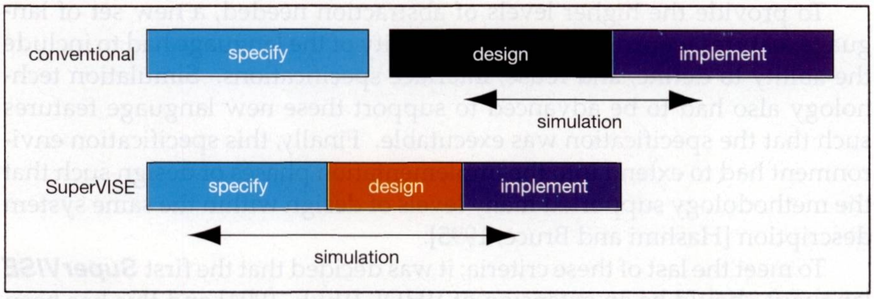


Figure 2: Simulation from Concept to Implementation

Since specifications are by nature 'loose' rather than 'fixed' descriptions they tend to be reusable and capable of supporting many design implementations.

2.2 Technical Needs

To achieve the overall objectives, three technical requirements came to the fore:

- In previous complex designs which involved many engineers and even many teams of engineers, the definition of the interfaces between teams and between individual engineers was of great importance. These interface definitions needed to be much more than a static definition, since they had to describe the communication protocol between units of design in enough detail to allow the separate design units to be developed independently. What was required was a language to support the capture of such protocols and these descriptions had to be separate to the unit descriptions themselves. The methodology needed the concept of *separate interface specifications*
- A means of specifying the design at a very high level was vital. System and architectural design decisions had to be made before moving to the next level of design. For this to be possible there had to be a language capable of supporting *higher design levels*

- Once captured this specification had to be checked for correctness. Finding system design errors later in the design flow was too expensive in time and resources. So the new methodology had to include an *executable specification*. Having captured and verified the system design at the highest level, it was vital that this description could be taken forward to the lower levels of design. It is inevitable in the development of large systems that different parts of the design progress at different rates. To support this, the methodology had to support *mixed multi-level modelling*.

To provide the higher levels of abstraction needed, a new set of language features were defined. The capability of the language had to include the ability to define, and reuse, interface specifications. Simulation technology also had to be advanced to support these new language features such that the specification was executable. Finally, this specification environment had to extend into the implementation phases of design such that the methodology supported many levels of design within the same system description [Hashmi and Bruce, 1995].

To meet the last of these criteria, it was decided that the first ***SuperVISE*** language should be an extension of VHDL [IEEE, 1994] and this has been provisionally named **VHDL+**.

The ***SuperVISE*** methodology is, however, applicable to more than just a VHDL design environment and additional languages will be supported in the future.

3. Methodology

3.1 The Interface Specification

The Interface serves three purposes:

- It serves as an unambiguous specification of the protocols to which all users (units) must conform
- Permits units to communicate despite being at different stages of design
- Provides a firewall between units, enabling them to be designed separately, but guaranteeing that they can communicate.

A high level interface specification tends to be declarative whereas, generally, a unit description represents an implementation and is therefore more likely to be imperative.

3.2 Interface Items

An Interface Specification is composed of interface items—*messages* and *transactions*—which can be hierarchically decomposed. The lowest message level, at which communication with pure VHDL models takes place, is composed of *signals*.

The Interface Specification contains all the information necessary to allow units, which can be defined at different levels, to communicate; see Figure 3.

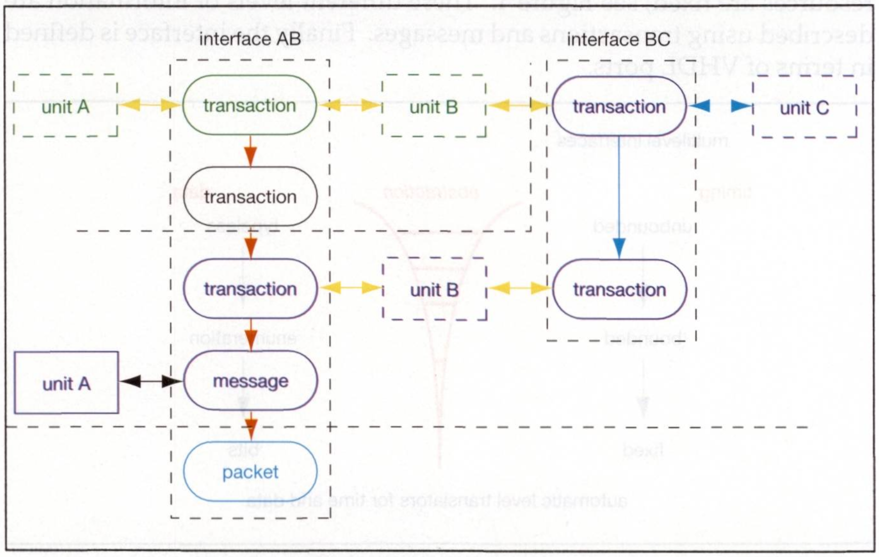


Figure 3: Interface between units

SuperVISE uses this information to automate the translation of information across the interface and to check that the communication obeys the interface *protocol*.

Interfaces can have one, two or more ends. Also, since they do not need to know to which units they are connected, *interface specifications* can be reused.

The main items in an interface definition are:

- *Transaction* definitions specify two-way conversations across an interface
- *Messages* define a one-way stream of information from one end of the interface to another end. They can be decomposed into other messages and can be defined at any level
- *Messages at the lowest level* specify static connectivity and are defined in terms of *signals*. If the designer wishes to co-simulate with VHDL models or to decompose into pure VHDL, this level is essential.

3.3 Multi-level Interface Specification

A ***SuperVISE*** interface specification describes the communication between design units. This single specification defines the communication at ALL levels. At the highest level the specification will be loosely defined—in time, data and resource. As the specification develops more detail is added and more design decisions are recorded, until eventually all time, data and resources are fixed; see Figure 4. These different levels of information are described using transactions and messages. Finally the interface is defined in terms of VHDL ports.

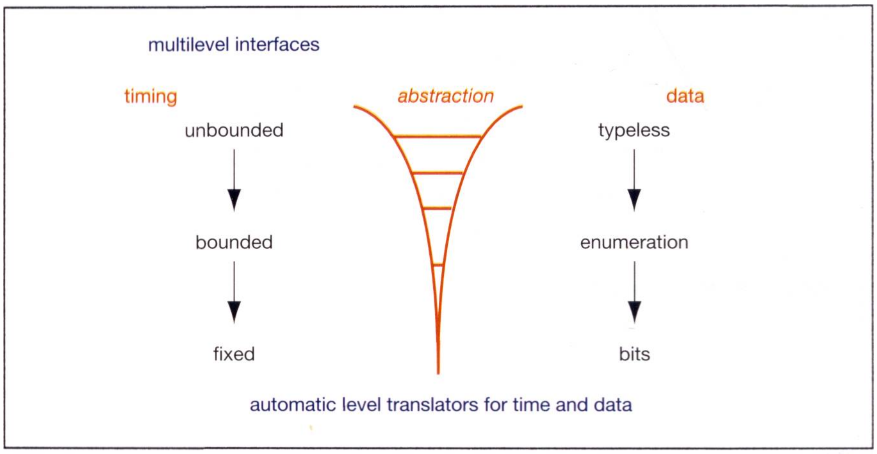


Figure 4: Multi-level Interface

4. Description Levels

As designs become larger and more complex the need for working at different levels of description becomes more important. The design of a complex electronic system is often a task that necessitates deployment of the best concurrent engineering practices. Different parts of the design will be developed at different speeds using different design styles.

For example, some functionality may be bought 'off-the-shelf' by purchasing or reusing models ('Intellectual Property Blocks') already proved against a specification. The descriptions of these 'off-the-shelf' models may be quite detailed with little or no abstraction (i.e. all decisions concerning the design have been concluded). The properties required of the language used to described these models demand little by way of abstraction features.

Other areas of the design may be designed 'top-down' starting with little or no constraint on the design characteristics. Here the designer needs to defer any decisions until later in the overall design flow. It may be that only when other areas of the design are complete constraints will become clear. The designer, therefore, needs a language which will not force unnecessary decisions.

4.1 Implementation

Register Transfer Level (RTL) and gate-level are implementation levels. The models represent the actual hardware and data is closely mapped to physical implementation. All timing is absolute and resource characteristics are fully defined. The move from RTL to gate-level is often via synthesis; see Figure 5. The most popular two languages used to define RTL and gate-level are VHDL [IEEE, 1994] and Verilog [IEEE, 1995].

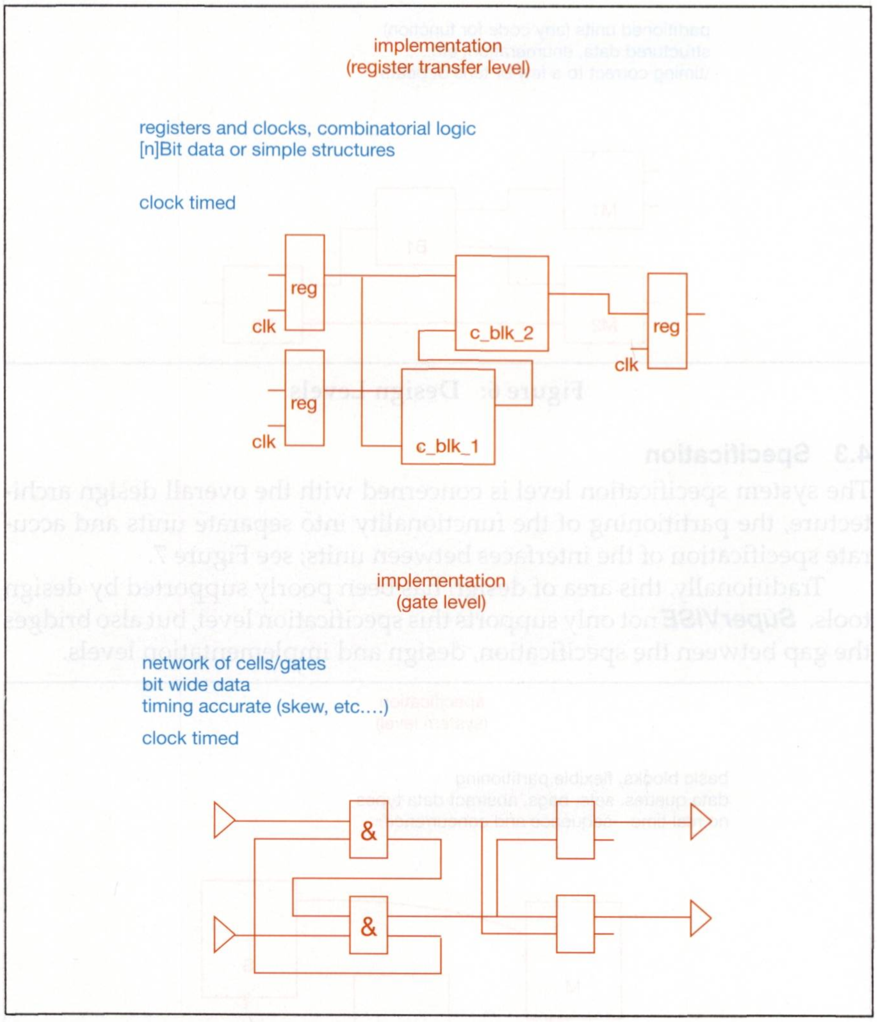


Figure 5: RTL and gate-level Implementation Levels

4.2 Design

Design levels are normally considered to be behavioural or algorithmic descriptions; see Figure 6. Not all decisions have been made (i.e. designed). Data is often still defined using enumeration and integer types. The designer may like to defer decisions on timing, but the HDLs available do not support timing abstraction. The designer is probably also forced to decide on resource allocation during this stage of design.

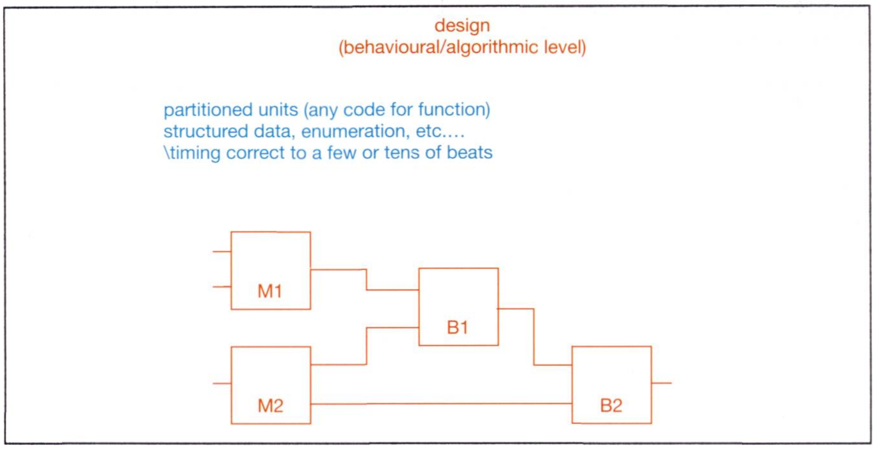

design
(behavioural/algorithmic level)
partitioned units (any code for function)
structured data, enumeration, etc....
\timing correct to a few or tens of beats
M1
B1
M2
B2

Figure 6: Design Levels

4.3 Specification

The system specification level is concerned with the overall design architecture, the partitioning of the functionality into separate units and accurate specification of the interfaces between units; see Figure 7.

Traditionally, this area of design has been poorly supported by design tools. ***SuperVISE*** not only supports this specification level, but also bridges the gap between the specification, design and implementation levels.

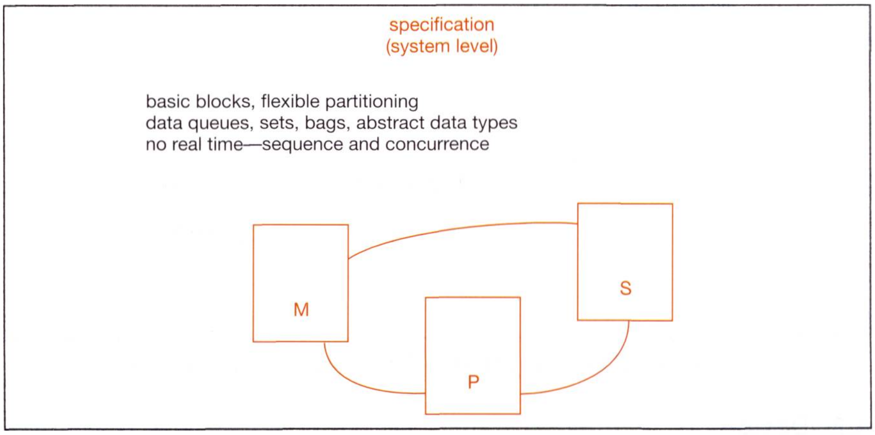

specification
(system level)
basic blocks, flexible partitioning
data queues, sets, bags, abstract data types
no real time—sequence and concurrence
M
S
P

Figure 7: Specification Levels

5. Verification

5.1 Executable Specification

Simulation is an established and essential verification tool in hardware design [Hodgson et al., 1995]. Today, most simulation occurs at RTL and gate level and simulation of the design described at a higher (e.g. specification) level is rarely achieved.

Achievement of an executable specification is fundamental to the ***SuperVISE*** methodology. It must be possible to simulate a ***SuperVISE*** specification and, furthermore, it must be possible to simulate a ***SuperVISE*** specification with lower levels of descriptions.

5.2 Mixed level modelling

SuperVISE tools manage the connection of different levels of unit specification to the interface and, where necessary, automatically perform the translation between different levels of communication.

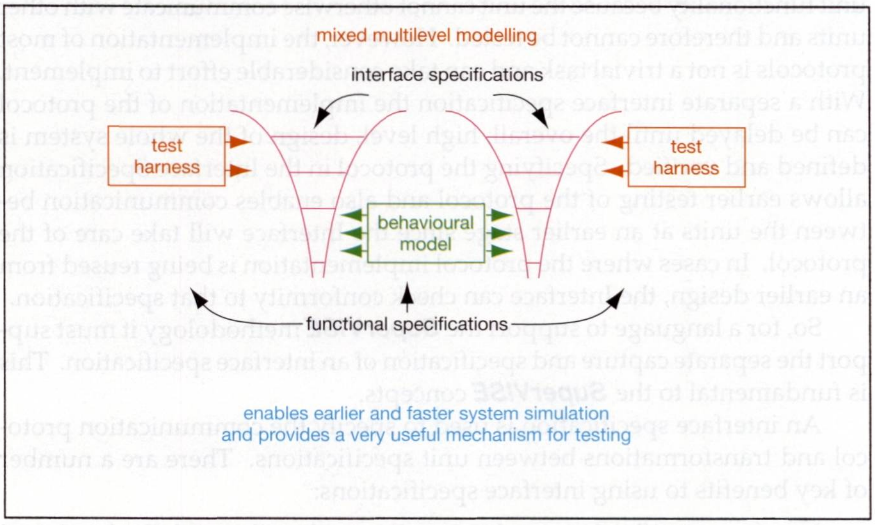

mixed multilevel modelling
interface specifications
test harness
test harness
behavioural model
functional specifications
enables earlier and faster system simulation
and provides a very useful mechanism for testing

Figure 8: Mixed Multi-level Modelling

One very useful application of the Mixed Multi-level Modelling capability, supported by ***SuperVISE***, is the reuse of a test harness across the many levels of unit description; see Figure 8. A test harness can be developed at a very high and abstract level, and then applied throughout the design cycle. The test harness remains unchanged despite the many iterations of design on the other side of the interface. Translation across the interface is automatically performed by the ***SuperVISE*** tools.

6. Language Requirements

6.1 Interface Modelling

The most important addition in ***SuperVISE*** methodology is the Interface Specification. The interface specification is not just a repository for the static data connections between units, but provides a full description of the communication protocol between units—it specifies the valid ways of using the interface. With a conventional, static, interface definition, if an error occurs in the protocol between two units (say) it is difficult to determine which end is 'at fault'—each unit will implement its understanding of the protocol and that will be assumed to be correct. However, if the protocol is defined as part of the interface where both ends must be considered the specification is agreed by the using units and there is a common understanding of the communication. It then becomes much easier to detect and diagnose protocol faults.

Also, it is necessary to consider the implementation of the protocol in the units. Normally, this has to be done before any implementation of the unit functionality because the unit cannot otherwise communicate with other units and therefore cannot be tested. However, the implementation of most protocols is not a trivial task and can take considerable effort to implement. With a separate interface specification the implementation of the protocol can be delayed until the overall, high level, design of the whole system is defined and verified. Specifying the protocol in the Interface Specification allows earlier testing of the protocol and also enables communication between the units at an earlier stage since the Interface will take care of the protocol. In cases where the protocol implementation is being reused from an earlier design, the Interface can check conformity to that specification.

So, for a language to support the ***SuperVISE*** methodology it must support the separate capture and specification of an interface specification. This is fundamental to the ***SuperVISE*** concepts.

An interface specification is used to specify the communication protocol and transformations between unit specifications. There are a number of key benefits to using interface specifications:

- Interface specifications are freestanding and can be usefully developed in their own right
- Unit specifications, communicating with one another only via interface specifications, are assured of working to the same protocol without risk of different interpretation
- An interface specification can be used to define communication between unit specifications that have been described at different levels of abstraction
- The interface specification itself can be defined at many levels of abstraction

- Interface specifications are not dedicated to any particular unit specifications. Unit specifications wishing to use a particular interface specification can do so by mapping themselves on to named 'ends' of the interface specification. So, interface specifications can be reused between different combinations of unit specifications
- At high levels of design, almost all communication functionality is handled by the interface specifications. As design work progresses down the levels towards gate level, the balance of communication functionality changes until it is eventually all hard-wired into real functional hardware at a traditional VHDL level.

The **`interface`** construct is used to declare an interface specification. Like VHDL **`entity, configuration`** and **`package`** declarations, the **`interface`** declaration is a primary unit. An **`interface`** has a name, a list of *end* identifiers and a list of interface items.

Any interface item of an **`interface`** can be instantiated by unit specifications as long as the unit specification is mapped to the appropriate *end* name of the **`interface`**.

As the design implementation progresses, the communication protocol between unit specifications is implemented in the unit specifications themselves and less of the interface specification provides functionality. Thus the driving functionality in an interface specification is eventually implemented in the unit specifications; see Figure 9. Finally, the whole design may be described using ports rather than instantiating interface items. However, at all stages of design it is the interface specification that defines the communication rules and this can be used to check the correctness of any implementation (e.g. statically or via simulation).

interface
at high levels, units just send and receive data
design unit
interfaces mediate between design units, translating between high level communication and low level communication, and checking the low level interface implementations
design unit
at lower levels, more of the interface is implemented in the units
interface driver

Figure 9: Interfaces

During a project, it is likely that units will be at different stages of implementation at any one time and interfaces between units often change their form at the different levels even though the interface capability remains the same. Therefore, it is often impossible to co-simulate the different units until they have all reached the same stage of development, but Interface Specifications can contain the specification of the interface at many levels and it will automatically translate any communication between levels thus allowing Mixed Multi-level Modelling.

Capturing the different forms of the interface at the different levels also allows the designer to verify a new level earlier since the Interface Specification will check that the different levels adhere to the same overall protocol, and that one is translatable to the other.

At a low level, every message sent or received needs an explicit connection between the relevant units, but at higher levels there may not be explicit connections between units or these connections may need to change quickly. Basically, the semantics of connections at low levels attempt to model the hardware equivalent whereas, at a higher or system level, we are more concerned with the design and content of protocols, transactions and messages than with their detailed implementation.

In ***SuperVISE***, a connection at the interface level merely states that there is some communication between the units. The content of the interface specifies the types of messages that can be exchanged and these can be complex and/or composite in themselves. The unit can use any message or transaction it chooses from the interface—it does not need a new signal added to the network and across hierarchies every time a new message is used by the unit.

SuperVISE allows the system designer initially to capture the system specification naturally as a set of communicating processes without hardware inferences, then to design the high level using this specification, adding only as much detail as required and to validate the high level implementation against the system level specification.

6.2 The Unit specification

A unit specification is defined as an entity/architecture pair; i.e. a description that defines functionality. A unit specification can be very complex (e.g. a description of a computer system) or very simple (e.g. an OR gate). A unit specification can be composed of other unit specifications communicating through ports, as in a normal VHDL structural description, or it can be composed of unit specifications communicating via interface specifications. The extensions of **VHDL+** enable instantiation of interface components in a manner similar to instantiation of architecture components.

Unit specification descriptions reference the interface specifications that they use, and the unit specifications into which they decompose. Note that interfaces are in themselves multi-level—so in Figure 10, unit AB would communicate with units B, or BA, via the interface 'i'.

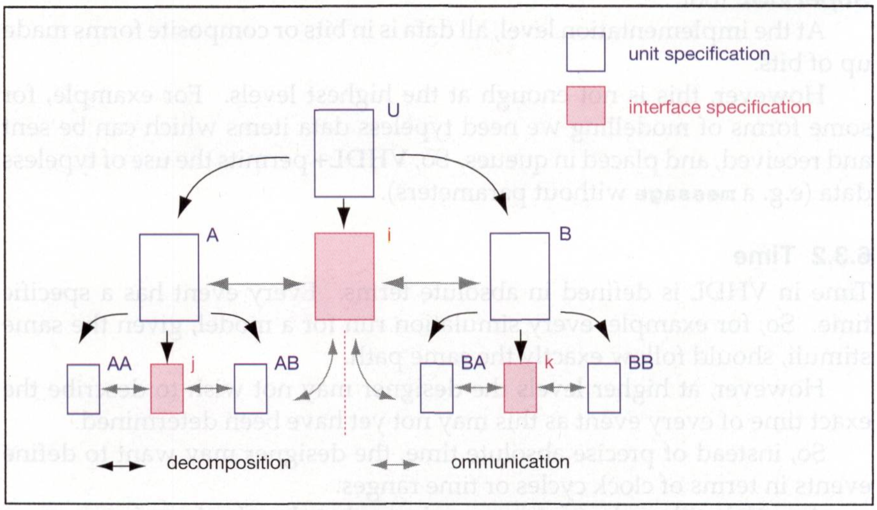


Figure 10: Specification Decomposition

A unit specification can, therefore, communicate with another unit specification via an interface specification or through ports. It is possible, and, in the early stages of a design, quite likely, that a design will have no ports and all communication is via interface specification using **`send`** and **`receive`** statements.

In **VHDL+** the unit specification may describe behaviour using the **`activity`**. The activity is a behavioural item which may be used to describe serial and parallel behaviour. The **`activity`** is instantiated and acts as a behavioural resource; e.g. an attempt to use an **`activity`** which is in use will cause queuing. The statements available within an **`activity`** include the **VHDL+** statements **`pause`**, **`after`**, **`choose`**, **`send`** and **`receive`** (see below).

6.3 Design Abstraction

6.3.1 Timing Abstraction

SuperVISE has also relaxed the specification of time in its models. At lower levels, timing delays in sequences need to be precisely defined but at higher levels it is more convenient to specify simple sequences of activities and place requirements on the start and finish of these activities. This is one of the main reasons most system specifications are not written in traditional HDLs.

SuperVISE allows events to be specified in sequence and in parallel, and to relate events to the beginning or end of other events. Time relationships can be unspecified, bounded intervals or precise—either as absolute time or clock intervals—and these can be simulated and validated with the ***SuperVISE*** tool.

At the implementation level, all data is in bits or composite forms made up of bits.

However, this is not enough at the highest levels. For example, for some forms of modelling we need typeless data items which can be sent and received, and placed in queues. So, **VHDL+** permits the use of typeless data (e.g. a **`message`** without parameters).

6.3.2 Time

Time in VHDL is defined in absolute terms. Every event has a specific time. So, for example, every simulation run for a model, given the same stimuli, should follow exactly the same path.

However, at higher levels the designer may not wish to describe the exact time of every event as this may not yet have been determined.

So, instead of precise absolute time, the designer may want to define events in terms of clock cycles or time ranges.

A model with such constructs can be analysed to check performance or simulated. During simulation a pseudo-random choice of time in the range is made. This simulation will then check that the functionality can cope with the time range. Obviously the more simulation performed the better the check.

At even higher levels, there is sequence or concurrency of activities in the model even when the overall timing has not been determined. So the language needs to enable the designer to describe dependencies between activities without having to connect them explicitly.

The **`pause`** construct introduces time ranges. The **`after`** construct provides a method for describing dependencies between interface or activity items. **VHDL+** also introduces a **`clock`** declaration such that an interface can be defined in terms of more abstract time units.

6.3.3 Resource Abstraction

System design is like software design with resources being created as needed rather than as in traditional hardware design where resources are precious and difficult to get. This is simply because a system level design spends much time in an experimental phase with ideas being tried out and discarded. It is more useful simply to provide any extra resource necessary and also allow the designer to view the number and type of resources being used at any one time by the model.

In ***SuperVISE*** it is possible to just specify that a resource or activity may be needed more than once and the tool will clone new concurrencies as required. A maximum concurrency can also be specified, and the concurrencies can be constrained to be pipelined. Information on concurrency creation and use can be kept during simulation for later analysis.

The full set of **VHDL+** extensions does include fixed and open concurrency. In **VHDL+** the **`message`** and the **`activity`** are treated as resource and instantiation of a resource which is in use will cause that instantiation to be queued.

6.4 Statement Scheduling

There are three basic statement scheduling mechanisms which may be used in **VHDL+** specifications:

- Statements that activate in sequence: A after B after C ...
  This applies to statements between a **`serial`** ... **`end`** pair.
- Statements that activate in parallel.
  This applies to statements between a **`parallel`** ... **`end`** pair.
- One, and only one, of a choice of statements will activate.
  This is achieved in a **`message`** with the **`choice`** ... **`end`** construct, and in an **`activity`** using the **`choose`** ... **`end`** construct.

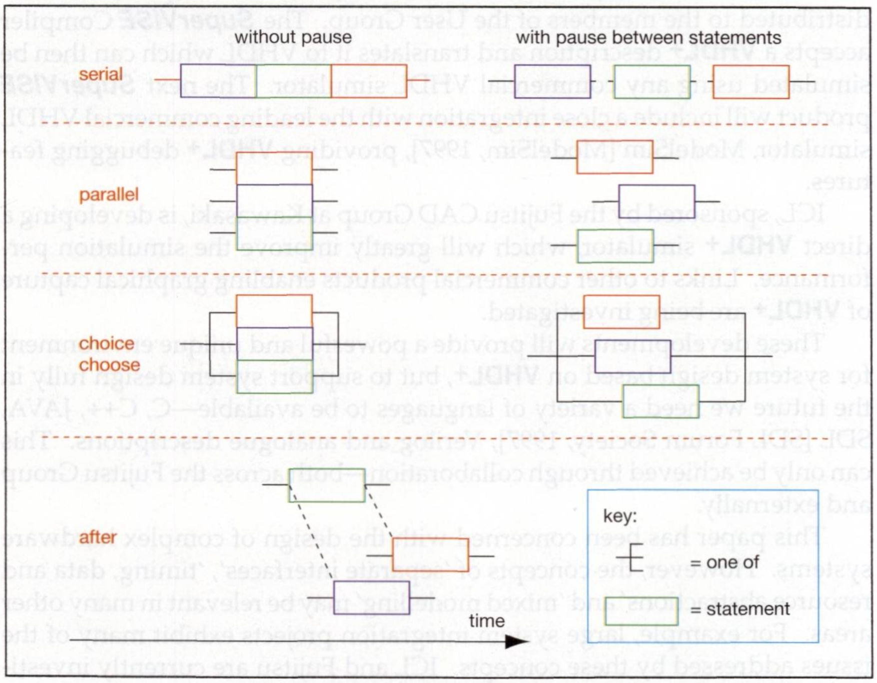


Figure 11: Statement Ordering

Additionally, there are two methods of order and time specification :

- Synchronizing the end or beginning of an item with another.
  This uses the **after** ... **then** ... **end** construct.
- Specifying some passage of time.
  This uses the **pause** statement.

The diagrams shown in Figure 11 illustrate some examples of statement ordering.

7. Future Developments

As mentioned earlier, the development of ***SuperVISE*** is being driven by the ***SuperVISE*** User Group and the pilot studies currently underway in a number of major electronic systems companies. In particular the telecommunications industry is showing great interest and **VHDL+** is being extended to fully support their requirements.

VHDL+ is now the subject of an IEEE Study Group which is looking into standardizing language extensions for system and interface descriptions. ICL is chairing this group.

The first of the ***SuperVISE*** tools, the ***SuperVISE*** Compiler, has been distributed to the members of the User Group. The ***SuperVISE*** Compiler accepts a **VHDL+** description and translates it to VHDL which can then be simulated using any commercial VHDL simulator. The next ***SuperVISE*** product will include a close integration with the leading commercial VHDL simulator, ModelSim [ModelSim, 1997], providing **VHDL+** debugging features.

ICL, sponsored by the Fujitsu CAD Group at Kawasaki, is developing a direct **VHDL+** simulator, which will greatly improve the simulation performance. Links to other commercial products enabling graphical capture of **VHDL+** are being investigated.

These developments will provide a powerful and unique environment for system design based on **VHDL+**, but to support system design fully in the future we need a variety of languages to be available—C, C++, JAVA, SDL [SDL Forum Society, 1997], Verilog and analogue descriptions. This can only be achieved through collaboration—both across the Fujitsu Group and externally.

This paper has been concerned with the design of complex hardware systems. However, the concepts of 'separate interfaces', 'timing, data and resource abstractions' and 'mixed modelling' may be relevant in many other areas. For example, large system integration projects exhibit many of the issues addressed by these concepts. ICL and Fujitsu are currently investigating the value of further research in this area.

8. Conclusions

The new methodology introduced for the Trimetra (SY) project proved an exceptional success. ***SuperVISE*** has enhanced that methodology and made it acceptable to a wider set of applications.

Further developments to ***SuperVISE*** are underway with the ambitious, but achievable, goal of establishing a new standard for the design of large, complex electronic systems.

Bibliography

ALLT, G., DESYLLAS, P., DUXBURY, M., HUGHES, K., LO, K., LYSONS, J.S.M. and ROSE, P.V., "The SY Node Design," ICL Systems Journal, Volume 12, Issue 1, May, 1997.

HASHMI, M.M.K. and BRUCE, A.C., "Design and Use of a System-Level Specification and Verification Methodology," Proceedings of EuroDAC Conference, 1995.

HODGSON, S., SHAAR, Z. and SMITH, A., "A High Performance VHDL Simulator for Large Systems Design," Proceedings of the IEEE European Design Automation Conference, 1995.

IEEE, "Standard VHDL Language Reference Manual," IEEE Std 1076–1993, The Institution of Electrical and Electronic Engineers, New York, USA, 1994.

IEEE, "Verilog Hardware Description Language Reference Manual," Draft Std 1364, The Institution of Electrical and Electronic Engineers, New York, USA, 1995.

ICL/WWW, http://www.icl.com/da

JEBSON, A., JONES, C. and VOSPER, H., "CHISLE: An Engineer's tool for hardware system design," ICL Technical Journal, Vol. 8, No. 3, May, 1993.

MODELSIM, www.model.com

SDL FORUM SOCIETY, ww.sdl-forum.org

Further information may be obtained from:
Steve Hodgson, ICL MAN05
+44 (0) 161 223 1301
email: sh@wg.icl.co.uk)

Biographies

Steve Hodgson

Steve Hodgson joined ICL West Gorton in 1973 after graduating from the University of Manchester with a BSc in Physics. He has worked on a number of projects, nearly all of which have been connected with the production and support of high performance Design Automation tools. Steve was one of the originators of the MSIM simulator which has been used for hardware design in West Gorton for over 15 years. Since 1992 Steve has been the manager of the Design Automation (DA) department. During this time DA have formed a close relationship with the Fujitsu CAD Group in Japan, who are key supporters of the SuperVISE technology. The DA group in Manchester are now responsible for establishing the use of SuperVISE and VHDL$^+$ worldwide.

Kemal Hashmi

Kamal graduated from Leeds University with a BSc. in Mathematics. He is an Associate Fellow of the Institute of Mathematics and a member of IEEE. He joined ICL West Gorton in 1982, where he initially worked on data management systems. In 1987 he led the development of a new version of the DGEN test generation system and then, in 1992, started work on CHISLE. Over the last few years his time has been dedicated to the creation and promotion of SuperVISE and VHDL$^+$. He is involved in many system-design related activities in Europe, the United States and Japan.

Process Modelling using the World Wide Web—ProcessWise™ Communicator

Peter Davies

Process Solutions, ICL HPS, Kidsgrove, Staffordshire,UK

Abstract

This article describes the exploitation of Web technology to provide business process modelling capabilities across the Web. It was driven by the vision of putting process modelling on everyone's desks in an organization with minimal cost per seat. The concept for this solution is based on the experience of designing ProcessWise WorkBench, a PC based application for process modelling. The new solution, called ProcessWise Communicator, uses JAVA™ to provide the editing facility on the desktop and also a JAVA application on the Web server to control the multi-user and database aspects.

1. Introduction

The following statement was the main motivation behind the implementation of ProcessWise Communicator.

"Put Business Processes on everyone's desks so they can see, understand and change their way of working."

Firstly, in order to put process descriptions on everyone's desks, the technology must be widely available and not platform dependent. It also implies that the cost of the client solution must be minimal, otherwise organizations will not be able to afford the solution for everyone. The emphasis on "see and understand" means the solution must be highly graphical and tailorable to different styles of process description. Finally, to change their way of working, the process description must be easily modifiable by the person looking at the diagram. Any complex editing and publishing mechanism will slow down improvements in the process description.

2. Structure for viewing models

The following diagrams describe the main components and structure of ProcessWise Communicator. ProcessWise Communicator is a JAVA based solution for use across an Internet or Intranet, although it can be used locally on one machine. The ProcessWise Communicator Applet resides on a

Web server along with HTML files which contain references to the Applet. This is shown in Figure 1.

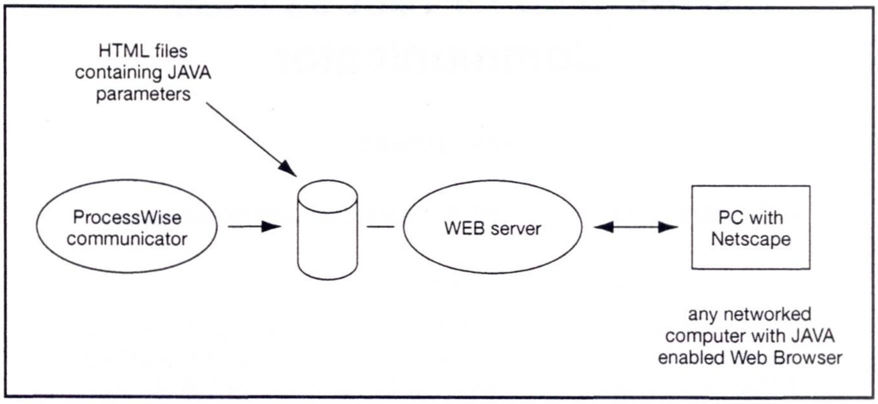

HTML files containing JAVA parameters
ProcessWise communicator
WEB server
PC with Netscape
any networked computer with JAVA enabled Web Browser

Figure 1: Viewing—Set-up

When the browser visits one of the HTML pages, the JAVA Applet is also automatically downloaded and run. The Applet extracts data from the rest of the HTML page which describes the diagram which should be displayed. There is therefore one generic Applet and many HTML files, each containing the data for the generic Applet, to display different diagrams. As the browser displays different pages there may be several copies of the Applet running at one time displaying different diagrams. The browser is responsible for navigating around the diagrams as each diagram is a separate HTML page. The standard browser "Forward", "Back" and history list are used to jump to different diagrams. Bookmarks can also be used in the browser to mark particular pages. The following diagram, Figure 2, shows one diagram downloaded into the browser.

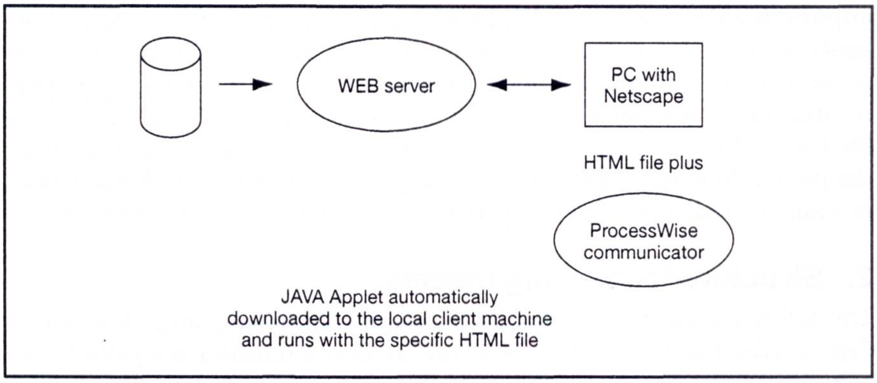

WEB server
PC with Netscape
HTML file plus
ProcessWise communicator
JAVA Applet automatically downloaded to the local client machine and runs with the specific HTML file

Figure 2: Viewing—at Runtime

3. Drivers, issues and benefits

The above mechanism for viewing process models was driven by working closely with a large international telecommunications company. They had a large volume of quality manuals and had also invested significant effort in producing process diagrams which they wanted published across the Web. The solution had to be platform independent since they had a mixture of UNIX, PC and MAC machines. The global nature of the company also required a solution which could be accessed from any location without any special software on the client machine. This directed the solution to be delivered through a Web browser. The options which were considered at this stage were converting the process diagrams to images in GIF format or developing a JAVA Applet which could draw the required diagram based on data supplied to it. Although the GIF approach would have been relatively simple to implement, it provided only limited functionality. The JAVA approach was more complex but allowed a much richer functionality. The customer preferred the richer functionality solution.

In designing the JAVA Applet several criteria were taken into account. Firstly, when in use each diagram would be accessed randomly and not always from the "first"/"top" page. When viewing the diagram the user would want it to be visible as quickly as possible and, therefore, it was important that the Applet was small in code size, since it would be downloaded on demand, and could also draw the diagrams quickly. This was matched with the requirement to deliver a solution to the customer and, therefore, dictated the use of JAVA 1.0 as this was the only version supported in available Web browsers at that time. To use this version to draw the types of diagrams required, it was necessary to implement an arrow drawing algorithm (based on trigonometry) and a text centring and wrapping algorithm. Both of these facilities were not available in JAVA 1.0.

To ensure the design was viable, it was compared with a solution using GIF images for the pictures. The comparison showed that the GIF images are around 2 to 3 times larger in file size than the data for the JAVA Applet. If the overhead of the size of the Applet is included, (which is downloaded once) then after viewing six diagrams the amount of data downloaded for the JAVA option will be less that the equivalent GIFs. The JAVA approach also includes all the information about the attributes and their values for each object and link information to other diagrams. This comparison is therefore only useful as a guideline. In terms of speed the time to display a diagram was broken down into the browser load time, the Applet load time and the drawing speed. This was compared to typical load times for a GIF file e.g. for a 14.4Kbs Modem, a 10KB GIF file takes 7 seconds. This gave a typical target of "a few seconds" to display the diagram. Depending on the complexity of the diagram this was achievable.

In summary, the viewing mechanism allowed processes to be published successfully, allowing access to all, irrespective of platform. No special software or licence (apart from the browser) was needed on the client machine and, because JAVA uses UNICODE characters, it supports all language requirements (e.g. Japanese Kanji characters).

4. Structure for editing models

There are two JAVA classes in the Applet which allow diagrams to be drawn. The first class will only allow viewing of the diagram as described so far. The second class, which is a subclass of the first, will also allow editing. If the HTML data names the second class, the Applet will start in view only mode but there is an option to allow the diagram to be edited. To control editing of the diagram a component called the Model-Manager is run on the Web server. This will listen for calls from many clients and when a request to edit arrives, it will lock the requested file and stop other clients from editing that file. Once the client has finished editing that diagram it is saved through the Model-Manager back into the file system on the Web Server. This is shown in Figure 3.

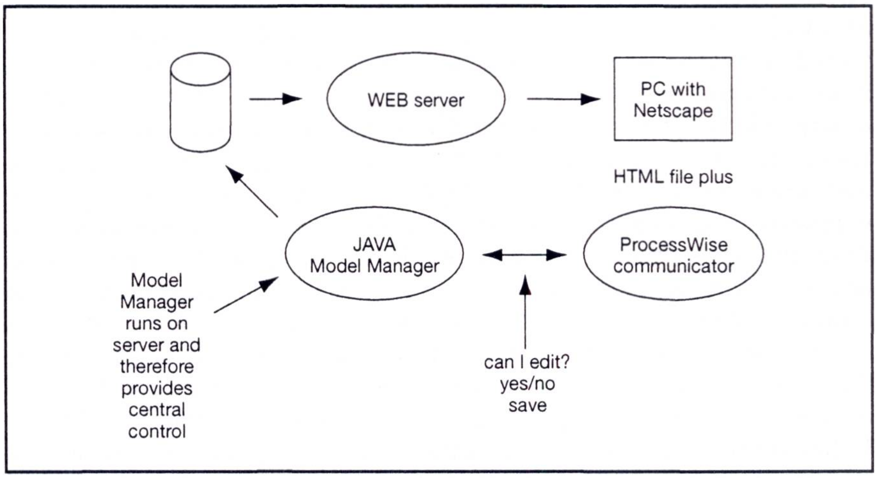


Figure 3: Editing

This facility to edit models now puts the capability to edit the process model on everyone's desks provided they have Internet/Intranet access and a Web browser. Demonstration models can be put on a Web server and potential customers can test it out without needing special "demonstration" software. A process modelling service could also be provided by renting out space on a Web server where all the models are saved. Note that because of the client/server architecture, if the Applet comes from a Web server, files will be saved back to the Web server and therefore provide a controlled environment for developing models. The Web browser can load

HTML files from the local machine and, in this mode, a local copy of the Model-Manager is needed to save the data. This way of working allows "off-line" or local process modelling.

The Model-Manager can also link to a database so that the information held in each HTML file can be shared within a model. This structure is shown in Figure 4. Whenever a diagram is saved the Model-Manager populates the linked database with the same information that is stored in the HTML files. The HTML files are retained for speed, so that diagrams can be browsed quickly without needing to open and query the database. The database link is built from the Java Database Connectivity (JDBC) package. This allows the Model-Manager to link to any registered ODBC or SQL compatible database. It has currently been tested with Microsoft Access. If no suitable database is provided, the Model-Manager will continue operating but will not support the sharing facilities described next.

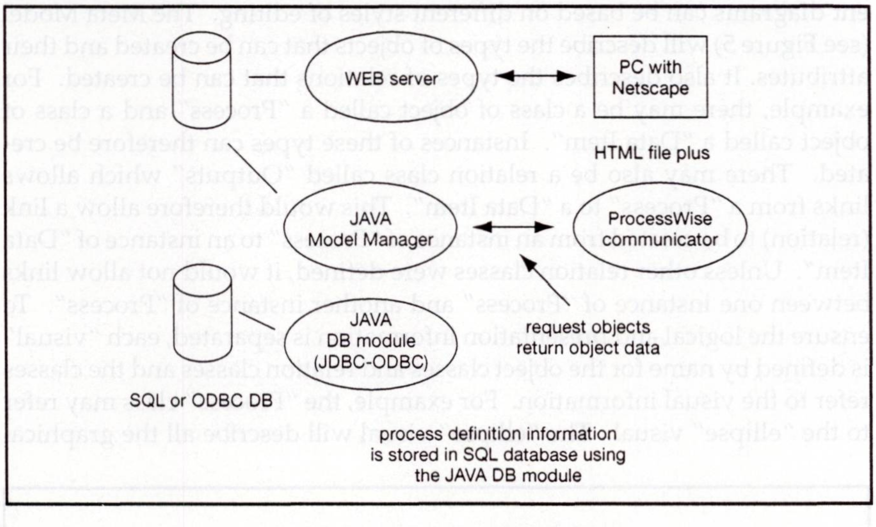


Figure 4: Link to Databases

Each diagram that is created is assumed to be self contained in that an object called X on one diagram is separate from another object called X on another diagram. However, with the database, objects can be shared between diagrams. There is one diagram where the object is defined, but on other diagrams, the Applet can fetch the definitions from the database of other existing objects. These can then be added to the current diagram. These objects can be linked (to/from) and moved around but they are only references to the real object definition on another diagram. The user is therefore not allowed to edit the attributes of this object reference.

By using an external "standard" database to store all the process information, it becomes very easy to integrate the information with other systems. This openness allows a corporate asset of process models to be built up where standard database reporting utilities can be used to access the information. The database also allows consistency to be maintained between different diagrams. The database and Web server components are the only components which will affect the performance as models become larger and larger. This architecture is expected to cope with models containing many thousands of diagrams.

5. The Meta Model

When diagrams are created through the JAVA Applet only certain types of objects can be created and only certain links (or lines) can be added between objects. Also, the way objects and links are displayed on the diagram depends on the types of objects and links. This information is called the Meta Model and is also stored in the HTML file. This means that different diagrams can be based on different styles of editing. The Meta Model (see Figure 5) will describe the types of objects that can be created and their attributes. It also describes the types of relations that can be created. For example, there may be a class of object called a "Process" and a class of object called a "Data Item". Instances of these types can therefore be created. There may also be a relation class called "Outputs" which allows links from a "Process" to a "Data Item". This would therefore allow a link (relation) to be created from an instance of "Process" to an instance of "Data Item". Unless other relation classes were defined, it would not allow links between one instance of "Process" and another instance of "Process". To ensure the logical and presentation information is separated, each "visual" is defined by name for the object classes and relation classes and the classes refer to the visual information. For example, the "Process" class may refer to the "ellipse" visual. The "ellipse" visual will describe all the graphical

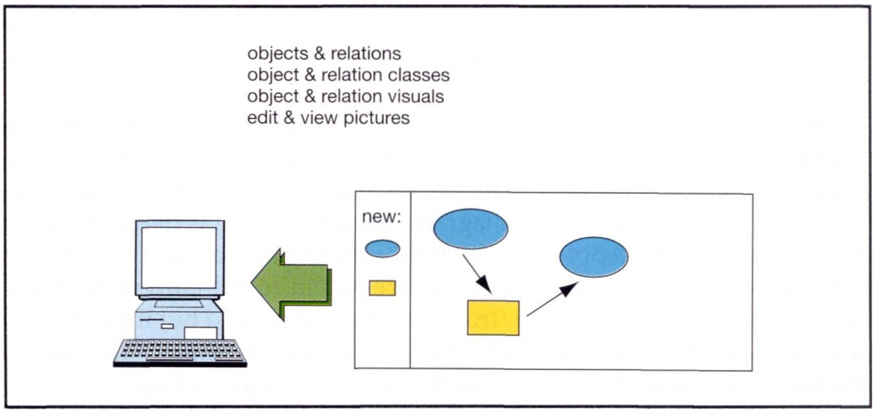


Figure 5: Structure

information needed such as size, colour, text size, font, etc.. Graphical information about the lines between two objects is handled similarly.

6. Modes

When a diagram is viewed in ProcessWise Communicator it can be in one of three modes: Read Only, Read Only + Request and Read/Write. The modes can be changed as shown in Figure 6.

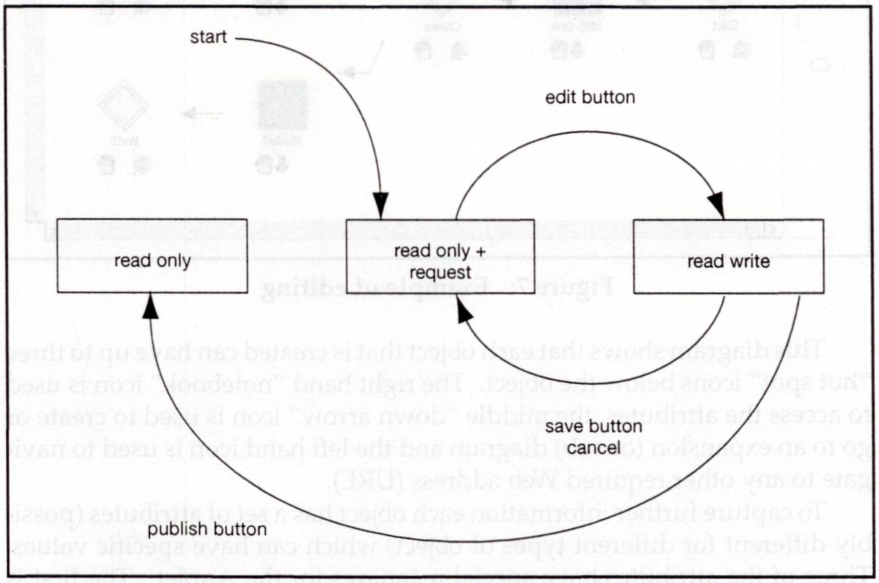


Figure 6: Changing Modes

7. Editing Diagrams

When a user wants to edit a diagram, the Model-Manager is contacted and, if the diagram is not already being edited, the Applet is allowed to switch to edit mode. This causes a toolbar of available types of objects to be displayed down the left hand side. New objects can be created by simply clicking on the required toolbar icon. This causes a new instance of this type of object to be drawn on the screen which the user can move around using the usual mouse dragging operations. The new object can be linked to other objects by using the Link Mode and dragging out the required lines. Only lines (relations) which conform to the relation types described in the Meta Model will be drawn. Other attempts will produce error messages about incorrect lines. These operations together with delete allow the user to build up a diagram of different interconnected objects based on a set of object types and relation types. An example diagram is shown in Figure 7.

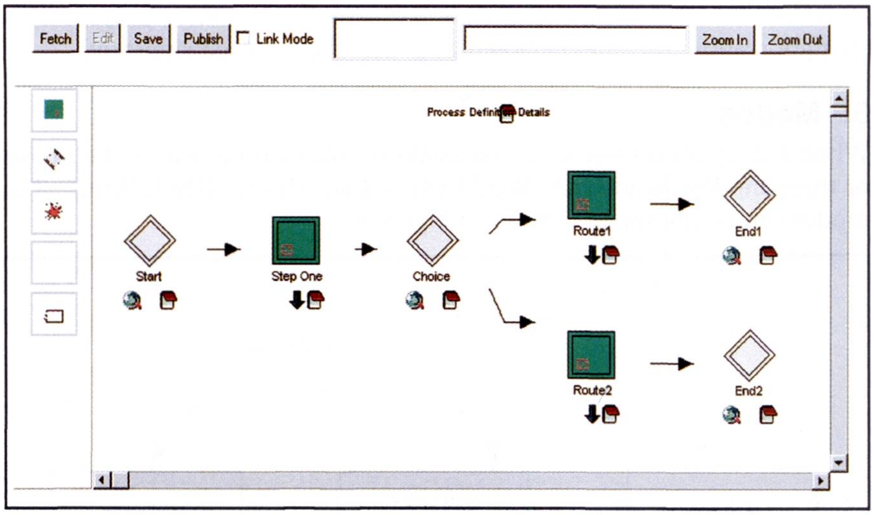


Figure 7: Example of editing

This diagram shows that each object that is created can have up to three "hot spot" icons below the object. The right hand "notebook" icon is used to access the attributes, the middle "down arrow" icon is used to create or go to an expansion (or sub) diagram and the left hand icon is used to navigate to any other required Web address (URL).

To capture further information each object has a set of attributes (possibly different for different types of object) which can have specific values. Three of the attributes have special meanings for the Applet. The first is "name". If the value of this attribute is changed, the text name of the object on the screen is changed. The second is the "expansion" attribute. If the "expansion" attribute is changed, the new value is used as the URL to go to when the centre "down arrow" icon is pressed. This attribute is normally updated automatically when a new expansion is created by clicking the "down arrow" icon while in edit mode. Editing the attribute directly allows a manual mechanism to link up otherwise disconnected diagrams. If the "expansion" attribute was previously empty, the shape of the object may change on the screen. This is because the Applet allows two alternative "visuals" to be used; one if there is no expansion and another if there is. The Meta Model may be set up to use the same "visual" in both cases, in which case there will be no difference on the screen. The third is the "link" attribute. If the "link" attribute is changed, the new value is used as the URL to go to when the left "world" icon is pressed. In the context of the Applet the URLs can be any valid URLs such as "http:", "mailto:", "news:", "ftp:", "javascript:", etc.. This allows the process diagrams to be linked to supporting documentation or other multi-media information. The javascript

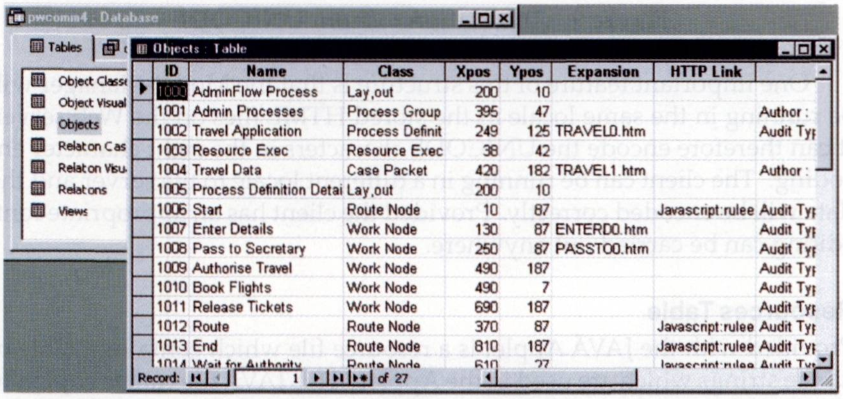
URL can allow the diagram to invoke some external functionality, written in javascript, which can use the JAVA APIs described in the appendix to carry out specific tasks on the process model data. For example, a javascript program can be created to add up all the values in the "cost" attributes in the diagram. Javascript can also be used to create programs which allow alternative ways of inputting and displaying the process model information. For example, a tabular data entry mechanism can be produced which updates the attributes stored in the model. For published quality processes the use of the "mailto:" URL is very useful in providing a feedback mechanism where comments on the process can be automatically e-mailed back to the author.

8. Database Tables

When a model is saved the following tables are updated in the database by the Model-Manager.

- Objects
- Relations
- Object Classes
- Relation Classes
- Object Visuals
- Relation Visuals
- Views

These are generic tables since there is no *a priori* information about the types of objects which will be in the model. Attributes for a particular type (class) of object are therefore stored in one field as a list of name/value pairs. The "Objects" table contains references to the "Object Classes" table which in turn contains references to the "Object Visuals" table. The "Rela-



ID	Name	Class	Xpos	Ypos	Expansion	HTTP Link	
1000	AdminFlow Process	Lay,out	200	10			
1001	Admin Processes	Process Group	395	47			Author :
1002	Travel Application	Process Definit	249	125	TRAVEL0.htm		Audit Typ
1003	Resource Exec	Resource Exec	38	42			
1004	Travel Data	Case Packet	420	182	TRAVEL1.htm		Author :
1005	Process Definition Detai	Lay,out	200	10			
1006	Start	Route Node	10	87		Javascript:rulee	Audit Typ
1007	Enter Details	Work Node	130	87	ENTER00.htm		Audit Typ
1008	Pass to Secretary	Work Node	250	87	PASST00.htm		Audit Typ
1009	Authorise Travel	Work Node	490	187			Audit Typ
1010	Book Flights	Work Node	490	7			Audit Typ
1011	Release Tickets	Work Node	690	187			Audit Typ
1012	Route	Route Node	370	87		Javascript:rulee	Audit Typ
1013	End	Route Node	810	187		Javascript:rulee	Audit Typ
1014	Wait for Authority	Route Node	610	27		Javascript:rulee	Audit Typ

Figure 8: Example Database

tions" table refers to two objects in the "Objects" table and the name of its class in the "Relation Classes" table. The "Relation Classes" table refers to the two types of objects which could be linked by references to the "Object Classes" table and also a reference to the "Relation Visuals" table. The "Views" table is a triple of view name, object reference and position information. An example Access database is shown in Figure 8.

9. Support for Internationalization

JAVA supports 16 bit UNICODE characters as standard within the language so support for different character sets is built in. Where Java communicates with files or other applications the translation to/from UNICODE is as shown in Figure 9.

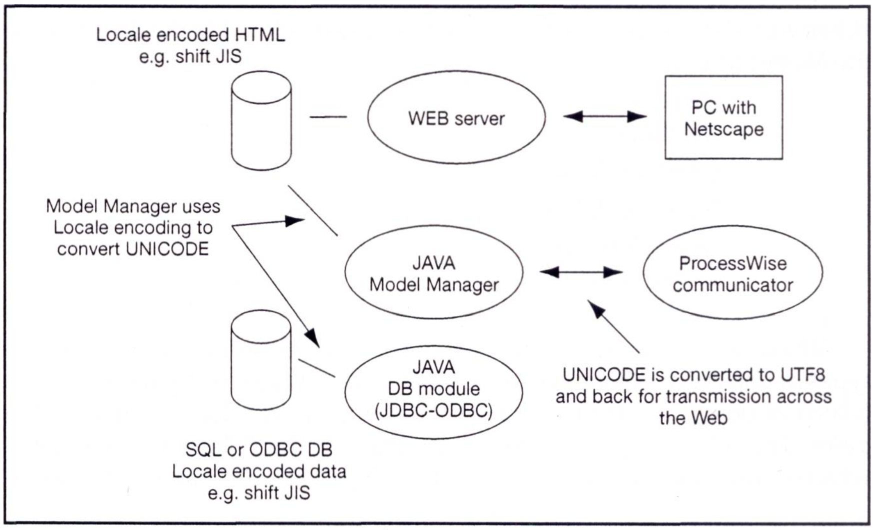

Locale encoded HTML
e.g. shift JIS
WEB server
PC with Netscape
Model Manager uses Locale encoding to convert UNICODE
JAVA Model Manager
ProcessWise communicator
JAVA DB module (JDBC-ODBC)
UNICODE is converted to UTF8 and back for transmission across the Web
SQL or ODBC DB Locale encoded data e.g. shift JIS

Figure 9: Translation to/from UNICODE

One important feature of this structure is that the Model-Manager will be running in the same locale as the stored HTML files on the Web server. It can therefore encode the UNICODE characters in the right character encoding. The client can be running in a different locale to the server and the data will be encoded correctly. Provided the client has an appropriate font, editing can be carried out anywhere.

Resources Table

Provided with the JAVA Applet is a resource file which contains a table of all the strings which are used in the Applet. This JAVA file can be copied to the locale specific name and the strings translated. This locale specific named file can be added to the JAVA Applet and it will now display using that information when it runs in that locale. Note that there is still only one version of the software and it will choose at run-time which language to use depending on the locale of the client machine. This means that a Japanese person in Japan looking at a model on a Web server in the UK, will see all the buttons and messages in Japanese. The model itself will still be in English. Below, in Figure 10, is shown an example model running under Netscape™ on a Japanese PC.

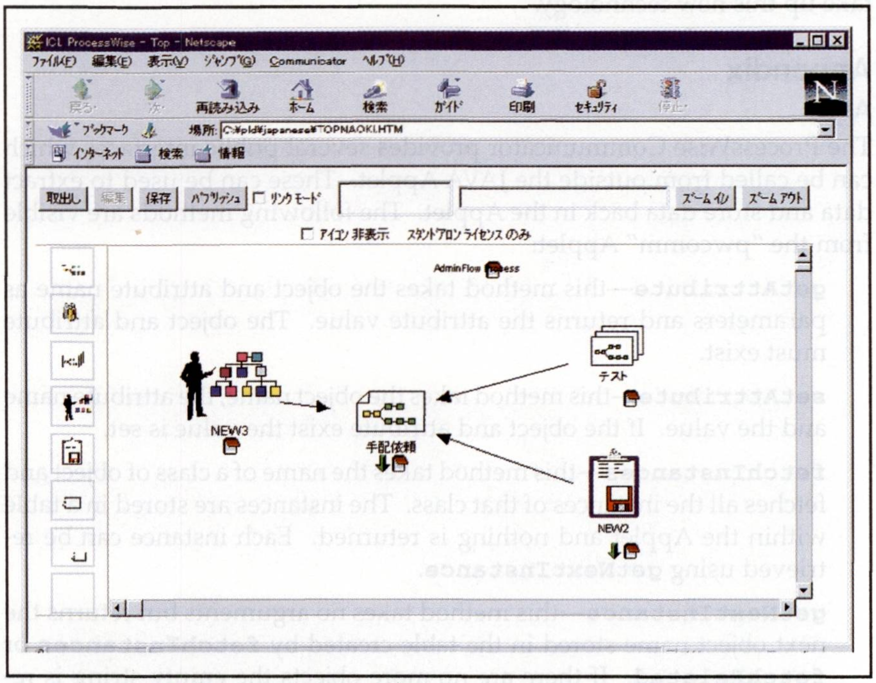

ICL ProcessWise - Top - Netscape
ファイル(F) 編集(E) 表示(V) ジャンプ(G) Communicator ヘルプ(H)
戻る 次へ 再読み込み ホーム 検索 ガイド 印刷 セキュリティ 停止
ブックマーク 場所: C:¥pld¥japanese¥TOPNAOKI.HTM
インターネット 検索 情報
取出し 編集 保存 パブリッシュ リンクモード ズームイン ズームアウト
アイコン非表示 スタンドアロンライセンスのみ
AdminFlow Process
NEW3
手配依頼
テスト
NEW2

Figure 10: Example Model running on a Japanese PC

10. Future

One of the more obvious applications for process modelling is to provide the necessary input for a Workflow or Process management system. This is the current focus of development in partnership with a US based computer manufacturer who has launched a new Workflow and Process Engine. The ICL group, ProcessWise, where this work was carried out, no longer exists so the development work will probably continue outside ICL.

11. Conclusions

This article has described a new way of modelling processes based on the latest JAVA and Internet technology. This solution is commercially viable and is currently in use by several companies. The benefits of easy access and providing process capabilities on everyone's desks have also been described.

Acknowledgements

The author would like to thank Ian Horton for his unfaltering belief in the above approach and his skill in selling these ideas to customers. The author would also like to thank all the customers who were bold enough to take up this new technology.

Appendix

APIs

The ProcessWise Communicator provides several public interfaces which can be called from outside the JAVA Applet. These can be used to extract data and store data back in the Applet. The following methods are visible from the "pwcomm" Applet:

`getAttribute`—this method takes the object and attribute name as parameters and returns the attribute value. The object and attribute must exist.

`setAttribute`—this method takes the object name, the attribute name and the value. If the object and attribute exist the value is set.

`fetchInstances`—this method takes the name of a class of object and fetches all the instances of that class. The instances are stored in a table within the Applet and nothing is returned. Each instance can be retrieved using **`getNextInstance`**.

`getNextInstance`—this method takes no arguments but returns the next object name stored in the table created by **`fetchInstances`** or **`fetchRelated`**. If there are no more objects the empty string is returned.

`fetchRelated`—this is similar to **`fetchInstances`** in that the result is stored in a table in the Applet for retrieval by **`getNextInstance`**. This method takes the name of the start object, the name of the type of relation to follow and a boolean flag. If the flag is true, the relation is inverted and therefore the start object name will be treated as the end object name and the objects returned will be the start objects.

`getSelected`—this method takes no arguments but returns the currently selected object or the empty string if no objects are selected.

Server Functionality

The server functionality is provided by the Model-Manager which is a JAVA application. This application is run in a continuous loop on the Web server (or locally) waiting for calls from different Applet clients. The following calls can be received from the client:-

`NEW`—this is used to create a new sub-diagram. The client passes the name of the parent diagram and the MASTER page to be used as a template for the new diagram.

`EDIT`—when an edit is requested, the name of the requested HTML file is checked in a "lock" table. If the file is already locked a message is sent back to the client refusing the edit request. If it is not locked the filename is entered in the lock table and the edit allowed. The lock information is held internally in the Model-Manager and therefore restarting the Model-Manager will reset all the lock information.

`SAVE`—when the client saves or publishes a diagram this command is used. All the Applet data is passed from the client which the Model-Manager saves into the HTML file. The Applet data is also converted into SQL statements which are used to update the connected database. The old HTML file is backed up to a file with a ".bak" extension before the new HTML file is created. Information in the HTML file other than the Applet data is preserved when the file is updated so that headers, footers and other HTML information can be in the files. For example, a MASTER page could contain the Meta Model for a diagram but also javascript functionality and company standard page layout information including logos etc.. This will then be used in all new diagrams. Once a diagram is saved it is unlocked.

`CANCEL`—if the Applet is destroyed without the diagram being saved, the cancel command is used. This causes the Model-Manager to unlock the file to allow other users to edit the diagram.

`SELECT`—this command causes a query on the database to find all the instances of a particular type of object. The list of objects found in the database is passed back to the client.

`GET`—this command is used to extract the details (attributes) of a particular object from the database. The details are sent to the client so that the object may be created on the screen.

Biography

Dr. Peter Davies is a Senior Consultant within ICL Services and has been active in the area of Business Process Modelling for ten years. He has been involved in several UK and European funded research programmes addressing different aspects of process modelling. He developed the commercially available process modelling product, ProcessWise WorkBench, in 1992. He has a first class honours BSc in Computing and Electronics from Durham University where he also completed his PhD in 1986.

Mobile Applications for Ubiquitous Environments

Jean Bacon and David Halls

University of Cambridge Computer Laboratory, Pembroke Street, Cambridge, UK

Abstract

Future ubiquitous computing systems are predicated on plentiful net-work bandwidth. End-systems for this mass marketplace must be cheap and, above all, simple-to-use. Proposals are already being made for this style of ubiquitous environment, augmented by processor banks and storage services.

We have exploited network bandwidth to make both clients and servers stateless, that is, needing to maintain *no application-specific software.* All persistent knowledge about applications is maintained in the documents they exchange. We have achieved this as one style of use of a platform which supports mobile and distributed computations. The platform allows mobile continuations to be used to terminate an application at any point, transfer its state in the form of plain text, and resume it elsewhere.

Advantages of the approach include trivially simple client software, the ability to maintain and upgrade applications transparent to their clients, client-server session maintenance, application mobility, server reselection in response to user mobility or server overload, application history logging and the ability to resume an application from any point in its history for backtracking or failure recovery.

The platform is fully implemented as is its support for stateless clients and servers. We illustrate the approach by means of examples based on Web servers.

1. Introduction

1.1 Motivation

Ubiquitous computing systems involve many machines. For deployment on a large scale, each should be as simple and inexpensive as possible. We believe that end-user machines should ideally do nothing more than display information, rather than run applications loaded from the network. Obtaining an application (from a storage server) should be a simple matter of selecting from a catalogue. It is difficult to store state or maintain execution threads on servers permanently for each client since their number is potentially unbounded, as is the duration of any client-server session. Strategic use should therefore be made of powerful processing servers that provide execution facilities only.

Commercial development of the *network computer* [Oracle, 1996] has achieved simpler workstations, for example the JavaStation [Sun, 1996a] and the Acorn Network Computer [Acorn, 1997a], but at the expense of making servers more complex. All permanent storage is associated with servers and network computer clients download applications from them.

As computers become faster, it becomes feasible to build inexpensive machines from commodity hardware which act as general processing servers [Becker et al., 1995], [Anderson et al., 1995]. Their only task is to read code sent to them by clients, execute it and return the results. Knowledge about application functionality can be held elsewhere and sent to them. Permanent storage is provided as a separate facility; they simply provide processors and memory on which anyone can rent time. This paper shows how applications can be written to execute on such machines but not to reside there continuously during their lifetimes.

Interactive applications without a real-time requirement do not need a long-term presence on client machines. They only require to interact at certain times with the user through an interface that is perhaps specified in a document mark-up language. Client machines for these applications can be simpler than network computers because they do not have to run new software; the applications are run on servers and user interfaces are sent to the client when appropriate.

It would be inappropriate to run interactive applications on general processing servers because they would require a permanent presence there while waiting for the users' input. The number of such users could be large and too many applications permanently residing on a server might result in it being overloaded or charging clients for excessive use. In the case that a client does not return with the user's input, its state would be left residing on the server indefinitely.

Network bandwidth is becoming more plentiful. Video-on-demand is planned for the home [Acorn, 1997b], which requires data transfer rates in the order of 3 megabits per second. With high-speed networks, it is feasible to hold application state in the messages exchanged between client and server.

1.2 Exchanging application state

Servers which support state-saving of whole programs can save applications as dormant parts of the user interfaces they generate. We have built applications which move to the server when they need to be executed and move to the client as dormant state when they need to interact with the user. Servers are stateless because they hold no permanent knowledge of applications. User interfaces (documents) are stateful because they are used to store application state.

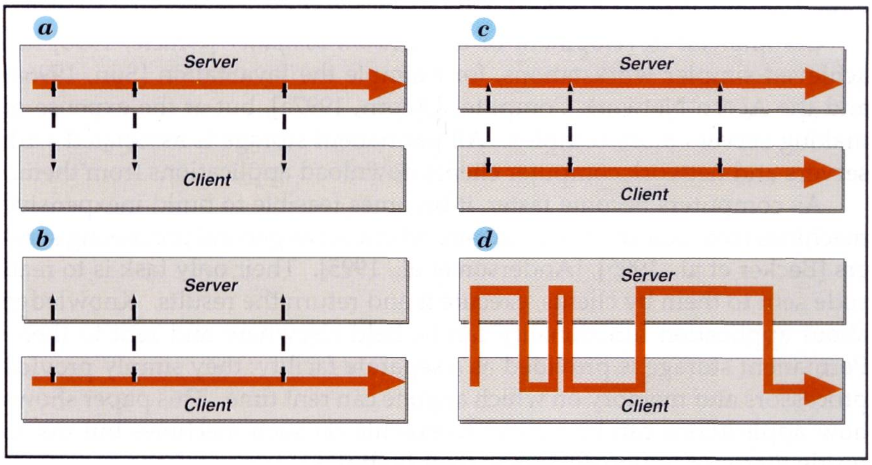


Figure 1: Application residence
(a) On a server
(b) On a client
(c) On both client and server
(d) Alternating between client and server

Any application running on a computer has a state. The state of an application that runs to completion on a single machine is held at all times in the memory and processor of that machine. A client-server based application involves interaction between a client process and a server process over a network. Its state can at any one time be held in a combination of the following three locations:

The client: A process on the client side might be running.

The network: The client may have dispatched a request or the server may have returned some pertinent results. Communication also passes through their operating systems.

The server: A process on the server side might be running.

Where application functionality is maintained varies according to resources available on the client side, server side and in the network. The complexity of the application and of the data sent across the network are also relevant. In current systems, knowledge of application functionality persists in the client side, in the server side or in a combination of both (see Figure 1a-c). The term session is used to denote an instance of related interactions (and connections) between the client and server components. Session state may be retained in the client, in the server or in the data transferred between the two.

We have designed and implemented a mechanism for remembering an application session's state in information exchanged between the computers used to provide client side and server side computation. Persistent knowledge of an application is held in neither the client nor in the server (see Figure 1d).

Exchanging application state between client and server places a greater burden on the network than holding it permanently in client or server. Distributed systems that have large numbers of lightweight client machines, high bandwidth networks and servers that must handle many requests, and are therefore supplied with substantial processing power and memory, are particularly suited to this technique. Ubiquitous computing environments have precisely these characteristics.

Section 2 shows how a mobile code system can be used to capture and save application state. Section 3 discusses related work. Section 4 discusses the security implications of our approach. Section 5 describes an implementation made using the World-Wide Web. Section 6 discusses the relationship between an application and the user interfaces into which its state is embedded. Section 7 presents an application written using the Web-based implementation that embeds its state in Web pages and performs only transient computation on servers. Section 8 shows how the ability to save application state can provide a powerful logging mechanism. This can be used to resume an application from any saved state either to recover from failure or to backtrack within the application. Section 9 discusses the advantages of being able to move application state around once it is saved into user interface documents. Examples are that a mobile application can rebind to a new server after moving or when a given server appears to be operating under heavy load. Also, automatic load balancing would be easier to build for anonymous processor banks than for dedicated servers. Finally, Section 10 summarises and concludes the paper.

2. Exchanging Application State with Mobile Code

2.1 Saving state using higher-order mobile code

A mobile code system is ideal for placing application state in data sent between a client and a server. Higher-order mobile code systems, which support state-saving of closures[1] and continuations[2], allow (parts of) applications to be transparently saved into byte-streams. Applications can be restored and restarted from saved byte-streams.

We have gained experience from building and deploying our own mobile code system, the Tube [Halls, 1997]. It provides a transparent and portable method for saving applications written in Scheme and is briefly described here.

[1]A closure is a function together with its defining scope.

[2]A continuation is a closure that represents the current state of execution. Calling a continuation results in the computation resuming from where the continuation was captured.

For a host to be able to send and receive mobile code it must run an instance of a Tube site. Programs are written in the Scheme language; they are transmitted over a network as marshalled expressions. If tagged as executable, they are passed to an interpreter for execution. The marshalling respects duplicate objects and cyclic references. We are able to transmit the full range of expressions, even closures and continuations. This means that programs can create arbitrary functions and send them elsewhere, with their closing environments.

Through being able to transmit continuations, a program can be stopped, moved and restarted elsewhere in a single call. The Tube does much more than just execute byte-code compiled scripts remotely—programs can modify themselves, create and dispatch other programs and treat their functions and state as first class, transmissible data.

Mobile programs are given a special environment within which they can access only certain symbols. They execute in a restricted interpreter and are only allowed to access those functions that the platform's owner makes available.

Tube sites are multi-threaded. Full access to a POSIX threads interface provided by the underlying operating system is given to programs. A noticeboard is provided, which a program can post messages to or read messages from. Data values can thus be left by a mobile program for others that may arrive later to use. The noticeboard can be divided into different areas; an access control list is associated with each.

Tube sites can dynamically load at run-time compiled libraries of code and call functions contained in them. This allows facilities to be added without having to stop and recompile.

A user interface toolkit has been integrated with the Tube. Functions to create and manipulate widgets are available to programs. We have also enhanced the toolkit with the ability to return the state of any user interface, along with any call-backs registered on widgets, as a series of bytes. A corollary function takes a saved user interface and recreates it in a visible form. This allows a mobile program to create user interfaces on one machine and retain them as embedded state when it moves. We have also written a Netscape plug-in that allows the Tube to be used for writing Applets embedded in World-Wide Web pages.

The core state-saving functionality of the Tube is written entirely in Scheme. It is portable across Scheme interpreters and compilers, and thus also across operating systems. Code to interface with other systems, such as POSIX functions or a user interface toolkit, is not portable between operating systems. Currently, the implementation is UNIX-based. We use the Bigloo [Serrano and Weis, 1995] Scheme compiler/interpreter and the XForms [Zhao and Overmars, 1997] user interface toolkit. Versions are running on DEC Alphas under Digital UNIX, SUN SPARCs under Solaris, Intel

Pentiums under Linux and HP 9000s under HP-UX.

Besides stateless servers, the Tube has been put to use in building distributed object systems [Halls et al, 1996], ATM network control [Halls and Rooney, 1998] and supporting mobile users of multimedia applications [Bates et al, 1996]. More details on the Tube and its uses can be found in [Halls, 1997].

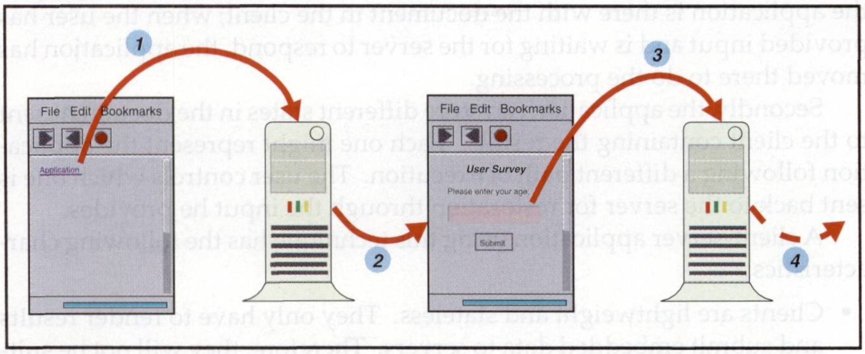


Figure 2: Application state in documents

2.2 Running applications on stateless clients and servers

Using higher-order mobile code, applications can be marshalled for saving to persistent store or transmitting over networks. This can be used to embed client-server applications in a user interface displayed at the client when input is required. The following progression is proposed for such an application so that it does not impose a permanent burden on the server and the client does not have to run any new code (see Figure 2):

1. The user simply clicks on the name of an application in a document browser. The browser extracts the application from the document and sends it to a generic mobile code server for execution there. The application starts up on the server.
2. If execution can proceed until termination without further user interaction, results are sent back to the client (browser) and the application terminates. If on the other hand user interaction is required, the application embeds its current state in a document sent back to the client asking for some input. The application then terminates itself at the server. It has become dormant and moves back into the client.
3. The client renders the document it receives, allowing the user to enter the required input. When the input has been entered, the user simply clicks on a button to submit it. The client extracts the application's state from the document and sends it, along with the user's input, to the mobile code server. The mobile code server restarts the application in

its previous state and gives the user's input to it.

4. The application sends itself and some results to the client and the cycle repeats.

There are two points to note here. Firstly, the application's state always accompanies its point of control. That is, when the user is providing input, the application is there with the document in the client; when the user has provided input and is waiting for the server to respond, the application has moved there to do the processing.

Secondly, the application can store different states in the document sent to the client containing the results. Each one might represent the application following a different path of execution. The user controls which one is sent back to the server for restoration through the input he provides.

A client-server application using this technique has the following characteristics:

- Clients are lightweight and stateless. They only have to render results and submit embedded data to servers. Therefore, they will not be suitable for user interfaces with real-time requirements (e.g. three-dimensional graphics viewers, multimedia presentations and games).
- Servers are completely stateless.
- Documents exchanged between clients and servers are heavyweight. They are stateful documents. This is not necessarily a problem if network resources are plentiful.
- A document sent to the client replaces the previous one. If the document's mark-up language does not support splitting it up into areas that can be updated individually, the user will see the whole display being refreshed each time. This might be a problem for applications that require finer grain control of a user interface.

Using stateful documents is not proposed as a way to do all client-server computing. Rather, it matches specific requirements—non-real-time applications that have to operate in distributed systems with very lightweight clients (e.g. display tiles), servers with large processing capabilities but without dedicated storage to maintain many persistent application states and a high-performance network (see Figure 3). Servers might not want to maintain application (session) state both because of storage limitations and due to the nature of the applications likely to be run on them—they might be expecting a large number of clients to connect for example.

To support stateful documents, a server has to provide facilities for executing applications and saving, marshalling and unmarshalling their states to and from the network. This places an extra load on them.

The rest of this paper discusses the use of stateful documents to make servers stateless in more detail. An implementation made over the World-

Wide Web and some experiments made with it are presented. First, we provide a context by discussing related work.

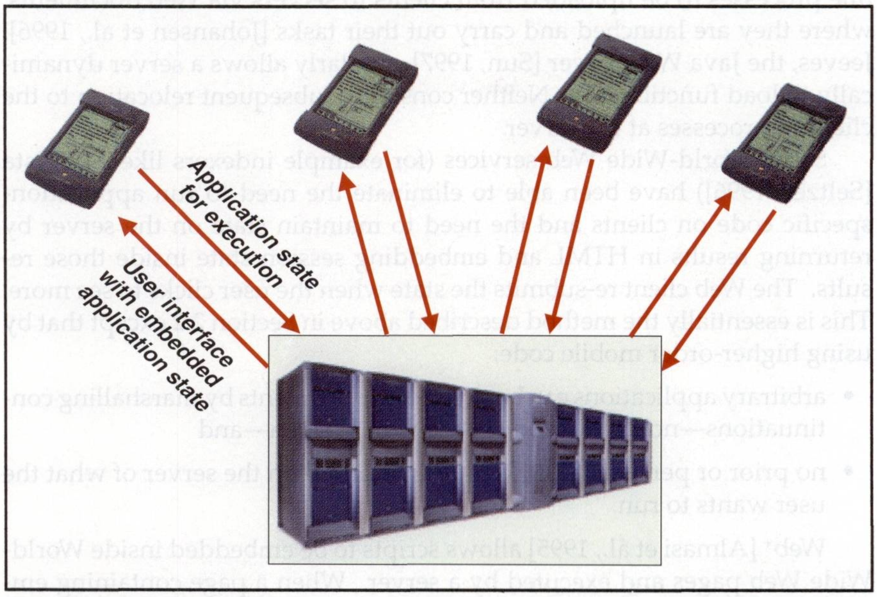

Application state for execution
User interface with embedded application state

Figure 3: Stateful documents, stateless servers and lightweight clients

3. Related Work

The design trade-offs explored in the 1980s for LAN-based distributed systems for the workplace are being revisited for the ubiquitous environments of the 1990s. Diskless workstations were used in the V system at Stanford [Cheriton and Zwaenepoel, 1983] and the idea has reappeared in the network computer proposals. The Cambridge Distributed Computing System (CDCS) was the first to investigate the potential of a "processor bank" [Bacon et al., 1990]. The CDCS processor bank comprised heterogeneous hardware, to allow for system evolution, and support for loading heterogeneous software into processor bank machines. Our approach has this functionality but employs a different mechanism.

Simple mobile code can be used to install software on client and server machines as it is needed. This is just a convenience, comparable with installing compiled binaries from traditional transportable media. The following issues remain:

- Application-specific code has to persist on clients and/or servers
- Clients and/or servers have to maintain session state (whether in the form of data or program threads)

PYTHON [CNR, 1997], ML [Rouaix, 1996], JAVA [Arnold and Gosling, 1996] and many other languages can be sent to run on client machines through the World-Wide Web. The TACOMA project has allowed its mobile processes to be uploaded from clients to servers via Web documents, where they are launched and carry out their tasks [Johansen et al., 1996]. Jeeves, the Java Web Server [Sun, 1997], similarly allows a server dynamically to load functionality. Neither considers subsequent relocation to the client of processes at the server.

Some World-Wide Web services (for example indexers like AltaVista [Seltzer, 1996]) have been able to eliminate the need to run application-specific code on clients and the need to maintain state on the server by returning results in HTML and embedding session state inside those results. The Web client re-submits the state when the user clicks to see more. This is essentially the method described above in Section 2.2 except that by using higher-order mobile code:

- arbitrary applications can be saved into documents by marshalling continuations—no special code needs to be written—and
- no prior or persistent knowledge is required on the server of what the user wants to run.

Web* [Almasi et al., 1995] allows scripts to be embedded inside World-Wide Web pages and executed by a server. When a page containing embedded scripts is requested from the server, a new page is formed for returning to the client by substituting each script with the output that results from executing it. This allows page elements, for example the current date, to be dynamically generated. Web* is similar to the Common Gateway Interface (CGI) [W3C, 1997], which allows scripted generation only of whole Web pages.

Web* also provides support for returning a script's state in Web pages. The state is then given to scripts embedded in pages subsequently retrieved by the user. However, Web* differs from the work described in this chapter in the following ways:

- Web* can only save variable values in Web pages. This means that a script's author must explicitly name global variables to be put into a Web page. The implementation described below in Section 5 provides a transparent mechanism to authors for saving state. It can save a program's execution state in a single call.
- Web* is page-oriented whereas the implementation described below is program-oriented. Web* applications have to be split into parts that are separately defined as scripts in Web pages. The implementation described in Section 5 allows programs to be written normally and provides a facility for saving their state simply in pages that they generate. User interaction via Web pages is implemented in a way that is integrated with program execution (see Section 6). Using Web*, programs

have to be adapted to fit the Web's model of interaction; the implementation described below adapts the Web to become part of program flow.

- Web* assumes that pages containing scripts already exist and continue to exist at the server. This chapter describes a technique that places no such burden on the server—no permanent knowledge of an application is held there.

The Network Computer [Oracle, 1996] with its thin clients and application servers is a vision of the future particularly relevant to the work described here. However, network computers have to be able to execute arbitrary code downloaded to them. This makes them more expensive than they would otherwise be and is not always necessary.

The rest of this paper discusses sending applications to the server when they need to execute and putting them dormant into the documents sent to the client when user input is required. First, we discuss the security implications of our approach and then how we have applied it to the World-Wide Web.

4. Security

The security both of application state that is transferred over the network and of servers that execute arbitrary application state must be ensured.

The fact that application state is placed alongside sensitive data leads to the concern that should untrusted parties breach security, they obtain not only the data but also code that can be used to interpret it. We argue that if someone is prepared to expend considerable effort to obtain some sensitive data then they have the wherewithal to make use of it, with or without the associated code. The necessity to assure the security of network traffic is not unique to our approach; data is vulnerable in any large-scale, ubiquitous environment used for personal or financial purposes.

The security of executing code delivered over a network is an important research area. In our environment, application state that is sent over the network to general processing servers must be denied access to any sensitive resources held there.

The prototype implementation described in Section 5.2 implements only very basic security, that is to isolate application state from sensitive resources by executing it in a restricted environment. This is done in user-space, which is inadequate for a production system. There should be operating system support for this [Zakinthios and Lee, 1997] to help avoid naive implementation in user space [McGraw and Felten, 1996], [Felten, 1997] and to provide a flexible system that is secure from the lowest level upwards. Current practice involves users having to trust software vendors that their code is safe; this applies as much to applications installed from CD-ROM as it does to code delivered over the network. Moving towards lower-level support for restricted environments of execution would allow more software to be run more securely [Ali-Reza et al., 1996].

Work on Java is addressing the security of executing code delivered over the network [Gong, 1997]. We can learn from this work, although there is still a debate about the merits of restricted environments of execution [Sun, 1996b], [Wulf et al., 1997] and research into Java security is ongoing [Gong et al, 1997]. Some recent work [Farmer et al., 1996a], [Farmer et al., 1996b] has addressed the notion of trust in mobile code systems. Other work [DARPA, 1997], [Cardelli and Gordon, 1997] has started to examine fundamentals of mobile code security.

We believe there are some things which help to mitigate the security concerns of the work described in this paper:

- Users' machines are dumb clients of servers. They simply display data and do not install any code from the network.
- Processing servers maintain no persistent application state nor have any requirement for a persistent store. Once a server has finished executing some (part of) an application, it can return to the same machine state it was in before starting. If necessary this machine state might be defined in read-only hardware, so that it cannot be corrupted by applications. This would isolate the effects of security breaches by individual applications from one another.

  To ensure that a server always returned to its 'clean' state, a time limit would have to be imposed on executing applications. This would be necessary anyway, to ensure that an application's use of processing time could be limited.

  A server able to restore and execute many application states in parallel on the same hardware would not be able to return to a 'clean' state after executing a particular application since there may be other executions in progress. In this case, support would be required to prevent applications from interfering with one another. As mentioned above, it is preferable that the operating system provides this support.
- Any resources such as databases or commonly used libraries of code can be deployed on different machines from the servers. Applications executed on the server communicate with these resources using traditional methods such as messaging or remote procedure call. This isolates the effects of security breaches on general processing servers from the rest of the system. We use these servers to provide only processing for applications when they need it and no more. For increased security, access to other resources can be made off-machine across the network.

We believe that the security requirements on single-tasking processing servers are comparable to those on storage servers across other architectures for ubiquitous systems.

From the user's perspective, in order to use a very lightweight client machine, he must trust a remote general processing server to execute the applications he wishes to use. We argue that users already put their trust in the applications they use and the programmers that write them. We also argue that running applications remotely over the network involves no higher level of trust than installing and running them locally on one's own computer.

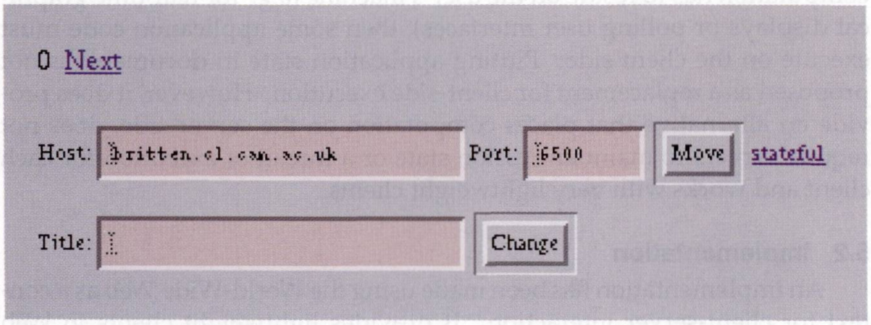


Figure 4: Counter in a stateful document

5. Application to the World-Wide Web

5.1 The Web as a testbed

The Web is suited for stateless servers and stateful documents because:

- A Web server potentially deals with requests from large numbers of diverse, widely distributed clients and maintaining their states in one place for long periods of time is burdensome. Their needs cannot be generalised and their physical connections may be unstable.
- The Hypertext Transfer Protocol (HTTP) [FGM+ 97] is stateless so servers do not by default maintain information about clients.
- Browsers are lightweight clients which simply display documents written in the HTML markup language.

In the system proposed here and detailed in the next section, servers provide general processing to browsers. They are written to treat requests for pages as programs to execute and send back to browsers the output of executing those programs. They forget about a program once it terminates. Browsers send applications to a server and then display their results as they are produced. If applications save their states in the results, they can be restarted at the server when the user provides input in the browser.

This allows for an application's state to be maintained whilst keeping both browsers and servers stateless. A server remembers nothing about the applications it executes when they finish running there and are sent back to the client. Browsers simply display the pages that servers return – they just need to render mark-up. Application state is contained in the links and forms contained in the pages. When the user provides input, the state is sent to the server and the application starts running again. It can then repeat the process by returning a new Web page containing new state.

Putting application state in Web pages increases the size of data sent between browser and server (see Section 7 for an indication of how much). Putting application state in documents is suited to fast networks. When a computation has to reside on the user's machine (e.g. for real-time graphical displays or polling user interfaces), then some application code must execute on the client side. Putting application state in documents is not proposed as a replacement for client-side execution. However, it does provide an alternative that places computation on the server side, does not require servers to maintain session state or a thread of execution for each client and works with very lightweight clients.

5.2 Implementation

An implementation has been made using the World-Wide Web as a context for client-server interaction. It provides lightweight clients in Web browsers (text or graphic) and a simple document mark-up language in HTML (that can be used for embedding application state).

Web browsers expect to communicate with servers that understand HTTP. An HTTP server has been written with the Tube higher-order mobile code system. It is installed simply by sending it to a Tube site.

Once installed, Web browsers can connect to this server. It treats any requests for pages it receives as (mobile) code to execute. The code is executed and any output produced sent back to the browser. This allows the Web to be used to provide convenient access to the Tube's compute servers.

The programs sent by browsers as page requests act as transitory Web servers; they run on the Tube Web server and disappear after producing their tasks' output as Web pages. They can be complete Web applications that implement an active and stateful user interface to generic services provided on the server side. They are novel because they are able to embed (part of) themselves into the documents they return to their browsers. They can do this because they are written as higher-order mobile code and can marshal arbitrary closures and continuations as plain text. The Tube mobile code system provides this facility.

The states are made part of links within HTML pages returned to browsers. Link addresses are set to point to the Tube HTTP server. After an application finishes sending its HTML document back to the browser, it terminates. The states it embedded in the document represent potential to execute again.

When the user clicks on one of the links in the browser, the embedded state becomes a request to the Tube HTTP server and the application starts running there again. It can then repeat the process by returning a new Web page with new states embedded in it. To summarise, an application runs on the HTTP server until interaction with the user through the client browser is required. At this point, it is removed from the server and its dormant state passed back with the interface presented to the user. After the user enters his input, it is sent back to the server with the application, which is restarted.

In the current implementation, a whole new HTML document is returned to the browser so that the user sees the whole page being updated. One might be able to achieve finer grain control over update by using Netscape's non-standard frame or layer enhancements to HTML.

A simple example is shown in Figure 4. It is a monotonically increasing counter. When started, it displays its initial value, zero. Every time the user clicks on the Next link, a new page is returned displaying the next value. This is because the next stage of the computation (that increments the counter and returns a new page) is stored as a marshalled continuation in the link. The size of the counter's marshalled state is around 10 kilobytes. However, the marshalling format is particularly amenable to compression—when compressed, the counter's state is just over 2 kilobytes in size. Section 7 contains further discussion of application state and the time it takes to compress and marshal it.

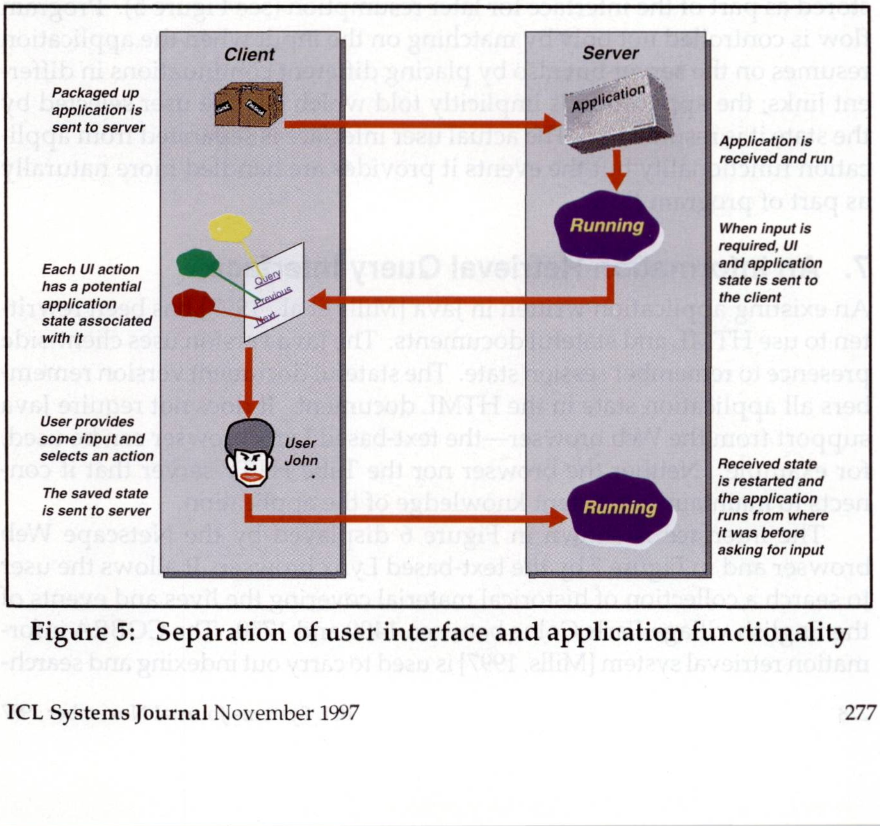

Client
Server
Packaged up application is sent to server
Application
Application is received and run
Running
When input is required, UI and application state is sent to the client
Each UI action has a potential application state associated with it
Query
Previous
Next
User provides some input and selects an action
User John
The saved state is sent to server
Running
Received state is restarted and the application runs from where it was before asking for input

Figure 5: Separation of user interface and application functionality

6. User Interface and Program Structure

The implementation benefits from the clean separation between user interface and application functionality that generating document mark-up promotes. That is, an application can proceed with its computation until it requires user input. It then generates from its data structures a fresh document to present to the user and waits for input. After the user provides input, the application can resume exactly in the same state as it was before waiting. The input can be processed and computation proceed accordingly.

This method of handling user input means that computation is driven by the application rather than by the user interface. Traditional call-back-based methods use the user interface to drive applications. The problem with using call-backs is that a side-effect must be used to note that a particular input has occurred. Subsequent call-backs can then tell what has happened before. The programmer must maintain some global state that is side-effected with the application's state each time something happens. Call-back driven applications can be converted into application-driven ones by using a continuation-passing style [Fuchs, 1996].

For client-server applications, a stateful server that maintains application threads can support application-driven user interaction by blocking after sending a user interface to the client until receiving input back from it. Stateless servers that are enabled by stateful documents achieve this too because while the user is providing input, the application is dormant and stored as part of the interface for later resumption (see Figure 5). Program flow is controlled not only by matching on the input when the application resumes on the server but also by placing different continuations in different links; the application is implicitly told which one the user selected by the state it is resumed in. The actual user interface is separated from application functionality but the events it provides are handled more naturally as part of program flow.

7. An Information Retrieval Query Interface

An existing application written in Java [Mills et al., 1997] has been rewritten to use HTML and stateful documents. The Java version uses client-side presence to remember session state. The stateful document version remembers all application state in the HTML document. It does not require Java support from the Web browser—the text-based Lynx browser can be used, for example. Neither the browser nor the Tube HTTP server that it connects to maintains persistent knowledge of the application.

The interface is shown in Figure 6 displayed by the Netscape Web browser and in Figure 7 by the text-based Lynx browser. It allows the user to search a collection of historical material covering the lives and events of the English village Earls Colne between 1400 and 1750. The COBRA information retrieval system [Mills, 1997] is used to carry out indexing and searching. The interface has to remember more state than a simple one such as AltaVista [Seltzer et al., 1996] because it supports relevance feedback. Relevance feedback allows the user to mark results that are relevant and present them as hints to the retrieval system for use in query refinement. A review of work on relevance feedback can be found in [Frakes and Baeza-Yates, 1992]. A user interface supporting relevance feedback must remember the complete context of a query; i.e. the user's search terms, search terms suggested by the system and any results marked as relevant to the query.

This state and the functionality of the interface must be held somewhere. The Java version holds them in the client. They could be held in the server if scripts were pre-installed, implementing the application's functionality, and space made available for persistent storage of state. The stateful document version remembers both the application's functionality and query context in the HTML documents displayed in the browser. At any one time, the state of the application is contained either in its execution at the server (in between user interaction), in a document sent to the browser for display or in a request sent from the browser to the server.

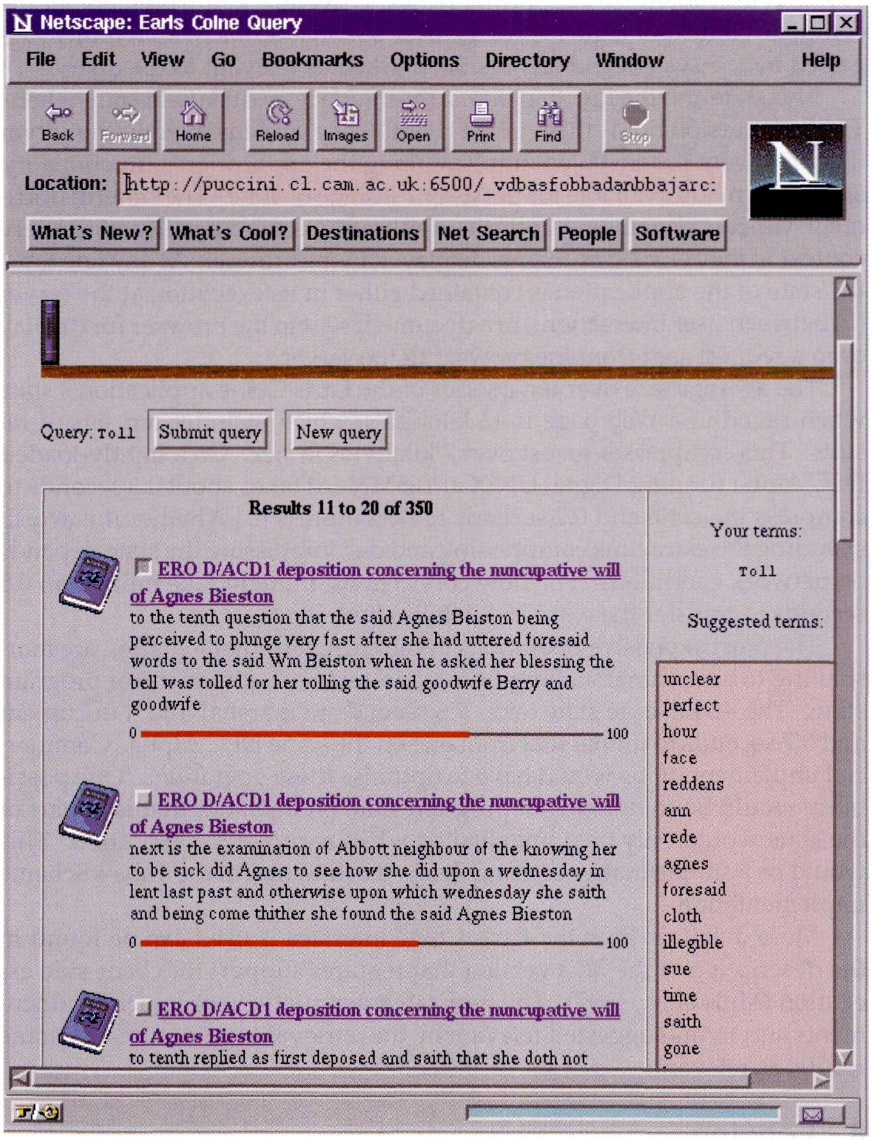
The average size over ten queries of the Earls Colne application's state when saved in a Web page is 42 kilobytes, which includes ten sets of results. This compresses to just over 7 kilobytes in size. On a lightly-loaded DEC Alpha running Digital UNIX at 166MHz, it takes about 0.3 seconds to compress the state and 0.2 seconds to decompress it. Whether it is worth spending the extra time compressing and decompressing the state depends on network conditions. For slow connections, it might take more than 0.5 seconds to transfer the extra 35 kilobytes both ways.

The most expensive operations in the current implementation are marshalling to and unmarshalling from plain-text representations of program state. The 42 kilobyte state takes 0.6 seconds to marshal into a document and 0.7 seconds to unmarshal from one on the same DEC Alpha. Commercial implementations would have to optimise these operations. One possibility would be to unmarshal program state on demand; that is, a part of the state would only be unmarshalled when required for execution. This would be a simple matter of using delayed evaluation in the Tube's Scheme implementation.

More detail on how the Earls Colne interface is used can be found in the description of the Java version that requires support for client-side execution [Mills et al., 1997]. The user can enter queries, see matching documents and terms suggested relevant by the retrieval engine and mark some results as relevant.

8. Logging

The ability to put all of an application's state inside a document is very useful for the Earls Colne query interface. By logging the documents returned to the browser, a history of the user's activity can be formed. One can then jump backwards in the log to a particular query and its context. For instance, a user might find that he has degraded results over time through questionable relevance feedback choices and wants to backtrack to a previous context. All he has to do is reload a previous document from


Netscape: Earls Colne Query
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://puccini.cl.cam.ac.uk:6500/_vdbasfobbadanbbajarc:
What's New? What's Cool? Destinations Net Search People Software
Query: Toll
Submit query
New query

Results 11 to 20 of 350

ERO D/ACD1 deposition concerning the nuncupative will of Agnes Bieston
to the tenth question that the said Agnes Beiston being perceived to plunge very fast after she had uttered foresaid words to the said Wm Beiston when he asked her blessing the bell was tolled for her tolling the said goodwife Berry and goodwife
0 — 100

ERO D/ACD1 deposition concerning the nuncupative will of Agnes Bieston
next is the examination of Abbott neighbour of the knowing her to be sick did Agnes to see how she did upon a wednesday in lent last past and otherwise upon which wednesday she saith and being come thither she found the said Agnes Bieston
0 — 100

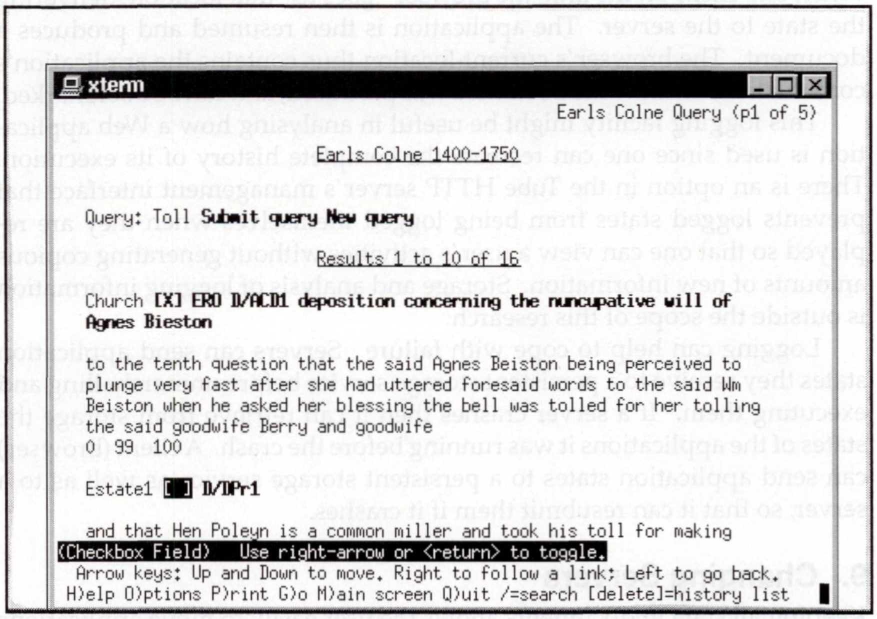
ERO D/ACD1 deposition concerning the nuncupative will of Agnes Bieston
to tenth replied as first deposed and saith that she doth not

Your terms:
Toll

Suggested terms:
unclear
perfect
hour
face
reddens
ann
rede
agnes
foresaid
cloth
illegible
sue
time
saith
gone

Figure 6: The stateful information retrieval interface displayed by Netscape

the log. A simple annotation facility is provided in case he wants to remember particular instances of the interface by name.



```
xterm
                                                    Earls Colne Query (p1 of 5)

                          Earls Colne 1400-1750

   Query: Toll Submit query New query

                          Results 1 to 10 of 16

   Church [X] ERO D/ACD1 deposition concerning the nuncupative will of
   Agnes Bieston

   to the tenth question that the said Agnes Beiston being perceived to
   plunge very fast after she had uttered foresaid words to the said Wm
   Beiston when he asked her blessing the bell was tolled for her tolling
   the said goodwife Berry and goodwife
   0 99 100

   Estate1 [ ] D/DPr1

   and that Hen Poleyn is a common miller and took his toll for making
(Checkbox Field)   Use right-arrow or <return> to toggle.
  Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
  H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

Figure 7: The stateful information retrieval interface displayed by Lynx

It should be emphasised that no special support is required here for logging. The application does not have to undo changes itself in order to backtrack to a previous state since each log entry is a complete and automatically-generated instance. When an entry is restored, it is the complete application that is restored, with the same context as when it was logged. It is not just the application's appearance that is restored, because potential new execution states are embedded as continuations within the user interface's hypertext links. Using stateful documents makes logging easy; using code resident in clients or on server machines requires bespoke solutions.

Logging can be carried out in the client (by the Web browser) or in the server (by the Tube HTTP server). If done on the server side, the states can be sent anywhere for storage—the client may tell the server to send them back to its local domain or may have contracted storage facilities from a third party. In the current implementation, the server saves states to disk and then makes them available for users to navigate via a management interface.

Web browsers can be made to log applications either by saving documents to disk or by bookmarking their addresses. The latter works because application state is contained in links within documents. These links also contain the address of the server on which the state should be restored. When one clicks on the link, the browser "goes to" that location, delivering the state to the server. The application is then resumed and produces a document. The browser's current location thus contains the application's continuation, i.e. how the document was produced, and can be bookmarked.

This logging facility might be useful in analysing how a Web application is used since one can recreate the complete history of its execution. There is an option in the Tube HTTP server's management interface that prevents logged states from being logged themselves when they are replayed so that one can view a user's activities without generating copious amounts of new information. Storage and analysis of logging information is outside the scope of this research.

Logging can help to cope with failure. Servers can send application states they receive to a persistent storage service before unmarshalling and executing them. If a server crashes then it can retrieve from storage the states of the applications it was running before the crash. A client (browser) can send application states to a persistent storage service as well as to a server, so that it can resubmit them if it crashes.

9. Changing Servers

Keeping all state in documents allows the user easily to move applications between servers, simply by changing the address of a link. Since servers keep no application-specific code, any one can be used, wherever it is located. This is useful if the user has to rent time on servers and a cheaper alternative is found, or if he moves and wants to use a server local to his new location. In this case, he can save the document to a file, change the link's address to the new server's location and simply load it into a browser at his new location. The server's address might be held in a well-known location, for instance in a trader.

Another use is for publishing programs. One could list stateful document Web applications in a Web page. Then either:

- The user saves that page, editing links to fill in his local server's address. This might be automated with support from the browser and involve a trader lookup. Or:
- The author might allow his server to be used for priming applications for use on a user's local server. The user would type in his server's address and a document containing links ready for launching the application in his domain would be returned.

Alternatively, the link containing the application's state might be e-mailed to the user, who would just fill in his local server's address and point his Web browser to it.

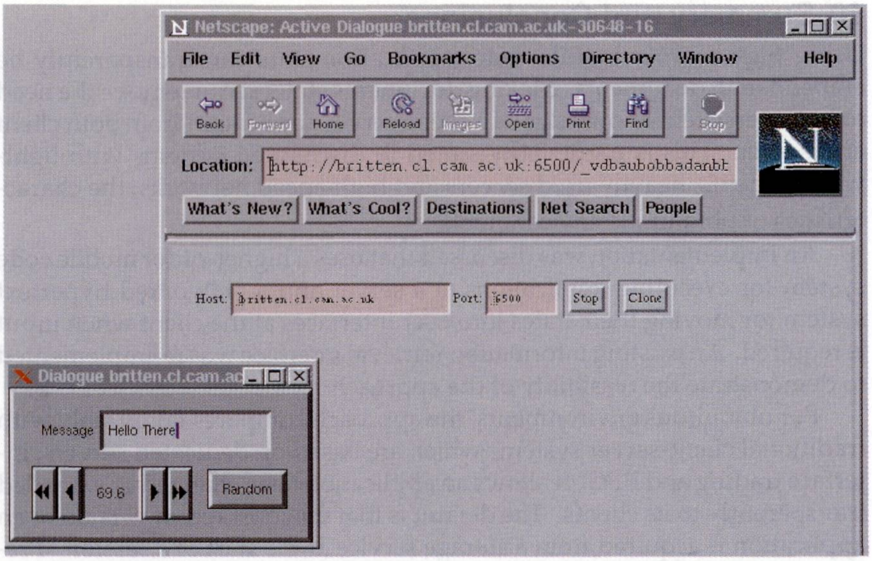

Netscape: Active Dialogue britten.cl.cam.ac.uk-30648-16
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://britten.cl.cam.ac.uk:6500/_vdbaubobbadanbt
What's New? What's Cool? Destinations Net Search People
Host: britten.cl.cam.ac.uk Port: 6500 Stop Clone
Dialogue britten.cl.cam.ac
Message Hello There
69.6
Random

Figure 8: Embedding external user interfaces in a stateful document

Since browsers download the initial states of applications from a central site, updating applications with new versions is simple. The new version of an application is installed at the download site and browsers use it for sending to a server for execution. This results in the decreased administration costs often cited for Network Computers [Oracle, 1996]—users cannot corrupt the installation of an application.

Finally, stateful documents have been extended to enable applications employing traditional user interface toolkits, such as Motif and XForms, to be launched and suspended from a Web browser. An example using the Tube's user interface state-saving facility is shown in Figure 8. It is launched by clicking on a link in which its code is embedded. The dialogue box is then displayed, which the user can manipulate by typing in a message or setting the counter.

At the same time, a page is returned to the Web browser which can be used to stop the dialogue box and place its state into another page returned to the browser. This page can then be used to restart the box in exactly the same state, with all of its components working. One can carry on saving and restoring its state indefinitely.

Just like stateless server Web applications, the dialogue box in this example can be moved between servers and saved to disk once it is captured inside a Web page. Its state as it is suspended and restarted can also be logged. One can also clone its state instead—as many copies of the dialogue box can then be made as required from the same page.

10. Summary and Conclusions

Using higher-order mobile code, application state can transparently be embedded in documents. Making documents stateful can remove the need to keep persistent knowledge of a client-server application from both client and server. This is particularly suited to distributed systems with light-weight clients, heavily used servers and high-speed networks; the characteristics of ubiquitous environments.

An implementation was discussed that uses a higher-order mobile code system for executing applications at a server and a networked hypertext system for moving their states into user interfaces at the client when input is required. An existing information retrieval interface was re-implemented to demonstrate the feasibility of the approach.

For ubiquitous environments, the approach compares favourably with traditional client-server systems which are based on dedicated servers, interface trading and RPC. It allows an application to be changed or extended transparently to its clients. The default is that the most recent version of an application is acquired from a storage service at the start of a session. Previous versions continue to exist for as long as the sessions which use them.

An application history may not only be logged conveniently but the history may be resumed from any point in the log if a user wishes to back-track to an earlier state or recover from failure.

Security concerns are comparable with those for alternative forms of ubiquitous environment, except that clients and servers hold no persistent state so are less vulnerable to penetration than dedicated servers and stateful clients. Storage services hold logs which contain sensitive data and must be protected. Encryption is expected to be available for documents in transit.

The stateful document can run at any server; if a user moves a new server can be used simply by rebinding to any local server in a processor bank. Automatic load balancing is facilitated by the ability to select any server to run a computation.

Most important, the client system need only be able to render the document interface, a desirable attribute for a ubiquitous environment.

Acknowledgements

David Halls gratefully acknowledges the support of EPSRC for his graduate studentship and both authors extend their thanks to ICL for general support of their research group.

Bibliography

ACORN (a), "The Acorn Network Computer," Acorn Computer Group, May 6, 1997. http://www.acorn.co.uk/acorn/products/nc/.

ACORN (b), "Acorn Online Media," Cambridge Interactive TV Trial, March 18, 1997. http://www.acorn.co.uk/acorn/news/releases/1997/march/trial.html.

ANDERSON, T.E., CULLER, D.E. and PATTERSON, D.A., "A case for networks of workstations," IEEE Micro, February 1995.

ARNOLD, K. and GOSLING, J., "The Java Programming Language," Addison-Wesley, 1996.

ALMASI, G., SUVAIALA, A., MUSLEA, I., CASCAVAL, C., DAVIS, T. and JAGANNATHAN, V., "Web*—a technology to make information available on the web," Fourth IEEE Workshop on Enabling Technology: Infrastructure for Collaborative Enterprises (WET ICE'95), 1995.

ADL-TABATABAI, A., LANGDALE, G., LUCCO, S. and WAHBE, R., "Efficient and language-independent mobile programs," Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation (PLDI), pages 127-136, May 1996.

BATES, J., HALLS, D. and BACON, J.M., "A framework to support mobile users of multimedia applications. ACM Mobile Networks and Nomadic Applications (NOMAD), 1(4), 1996.

BACON, J.M., LESLIE, I.M. and NEEDHAM, R.M., "Distributed computing with a processor bank," Proceedings of the Workshop on Distributed Computing, a European Update, number 433 in Lecture Notes in Computer Science, pages 147–161 (edited by Schroder-Preikschat and Zimmer), Springer-Verlag, Berlin, Germany, 1990.

BECKER, D.J., STERLING, T., SAVARESE, D., DORBAND, J.E., RANAWAK, U.A. and PACKER, C.V., "Beowulf: A parallel workstation for scientific computation," International Conference on Parallel Processing, 1995.

CARDELLI, L. and GORDON, A., "Abstractions for Mobile Computation," 1997. http://www.research.digital.com/SRC/personal/Luca_Cardelli/Ambit/Ambit.html.

CNR, Corporation for National Research Initiatives, Grail Home Page, March 12, 1997. http://monty.cnri.reston.va.us/grail/.

CHERITON, D.R. and ZWAENEPOEL, W., "The distributed v kernel and its performance for diskless workstations," Proceedings of the ACM Symposium on Operating System Principles, pages 129-140, October 1983.

DARPA, Foundations for Secure Mobile Code Workshop, Monterey, California, USA, March 26-28, 1997.
http://www.cs.nps.navy.mil/research/languages/wkshp.html.

FRAKES, W.B. and BAEZA-YATES, R. (editors), Information Retrieval Data Structures and Algorithms, Prentice Hall, 1992.

FELTEN, E.W., "Java Security: Frequently Asked Questions," Princeton University Secure Internet Programming Team, April 28, 1997. http://www.cs.princeton.edu/sip/java-faq.html.

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H. and BERNERS-LEE, T., "Hypertext transfer protocol _ HTTP/1.1," IETF Request for Comments 2068, January 1997.

FARMER, W.M., GUTTMAN, J.D. and SWARUP, V. (a), "Security for mobile agents: Issues and requirements," National Information Systems Security Conference, Baltimore, Maryland, USA, October 22-25, 1996.

FARMER, W.M., GUTTMAN, J.D. and SWARUP, V. (b), "Security for mobile agents: Authentication and state appraisal," Fourth European Symposium on Research in Computer Security (ESORICS '96), number 1146 in Lecture Notes in Computer Science, pages 118-130 (edited by Elisa Bertino, Helmut Kurth, Giancarlo Martella and Emilio Montolivo), Springer-Verlag, September, 1996.

FUCHS, M., "Escaping the event loop: an alternative control structure for multi-threaded GUIs," Engineering the Human Computer Interface (EHCI '95), (edited by Unger, C and Bass, L.J.), Chapman and Hall, 1996.

GONG, L., MUELLER, M., PRAFULLCHANDRA, H. and SCHEMERS, R., "Going beyond the sandbox: An overview of the new security architecture in the Java development kit 1.2," Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California, December, 1997.

GONG, L., "Java security: Present and near future," IEEE Micro, 17(3):14-19, May/June, 1997.

HALLS, D., "Applying Mobile Code to Distributed Systems," PhD thesis, University of Cambridge Computer Laboratory, June, 1997.

HALLS, D., BATES, J. and BACON, J.M., "Flexible distributed programming using mobile code," Proceedings of the Seventh ACM SIGOPS European Workshop, Connemara, Republic of Ireland, September, 1996.

HALLS, D. and ROONEY, S., "Controlling the tempest: Adaptive management in advanced atm control architectures," To appear in IEEE Journal on Selected Areas in Communication, 1998.

JOHANSEN, D., VAN RENESSE, R. and SCHNEIDER, F.B., "Supporting broad internet access to tacoma," Proceedings of the Seventh ACM SIGOPS European Workshop, Connemara, Republic of Ireland, September, 1996.

MCGRAW, G. and FELTEN, E., "Java Security: Hostile Applets, Holes and Antidotes," John Wiley and Sons, 1996.

MILLS, T.J., "Content Modelling in Multimedia Information Retrieval," PhD thesis (in preparation), University of Cambridge Computer Laboratory, 1997.

MILLS, T.J., MOODY, K. and RODDEN, K., "Providing world wide access to historical sources," Proceedings of the Sixth World-Wide Web Conference, Santa Clara, April, 1997.

ORACLE, "Network Computing Architecture White Paper," Oracle Corporation, September, 1996. http://www.oracle.com/nca/html/nca_wp.html.

ROUAIX, F., "A web navigator with applets in caml," Proceedings of the Fifth World-Wide Web Conference, INRIA, Paris, France, May, 1996.

SELTZER, R., RAY, E.J. and RAY, D.S., "The AltaVista Search Revolution: How to Find Anything on the Internet," Osborne McGraw-Hill, 1996.

SUN (a), "JavaStation—An Overview," Sun Microsystems, Inc., 1996. http://www.sun.com/javastation/whitepapers/javastation/javast_ch1.html.

SUN (b), "JavaSoft Security Forum," Sun Microsystems, Inc., 1996. http://java.sun.com/forum/securityForum.html.

SUN, "The Java Server Product Family," Sun Microsystems, Inc., 1997. http://jeeves.javasoft.com/.

SERRANO, M. and WEIS, P., "Bigloo: a portable and optimizing compiler for strict functional languages," Second Static Analysis Symposium, Lecture Notes in Computer Science, pages 366-381, Glasgow, Scotland, September, 1995.

W3C, "CGI—Common Gateway Interface," World Wide Web Consortium, 1997. http://www.w3.org/pub/WWW/CGI/.

WULF, W.A., PERI, R., TJADEN, B., WANG, C., KIENZLE, D., NAHAS, M. and COOPER, D., "Legion Worldwide Virtual Computer: Legion Security," 1997. http://www.cs.virginia.edu/~ legion/Security.html.

ZAKINTHIOS, A. and LEE, E.S., "A least privilege mechanism for user processes," Dependable Computing for Critical Applications (edited by Iyer, R.K., Morganti, M., Fuchs, W.K. and Gligor, V.), IEEE Computer Society, March, 1997.

ZHAO, T.C. and OVERMARS, M., XForms, 1997. http://bragg.phys.uwm.edu/xforms.

Biographies

Jean Bacon

Jean Bacon is a Lecturer in Computer Science at the University of Cambridge Computer Laboratory. She teaches in the systems area and is the author of books on computer architecture and concurrent systems. Her research interests are in distributed computing, focussing on system support for emerging applications which include multimedia.

David Halls

David Halls recently completed his PhD at the University of Cambridge Computer Laboratory. His research involved the investigation of the potential applications of mobile code. He now works for Persimmon, a Cambridge based IT company.

Middleware Support for Mobile Multimedia Applications

John Bates, David Halls and Jean Bacon

University of Cambridge Computer Laboratory, Pembroke Street, Cambridge, UK

Abstract

This paper describes a system we have developed to enable applications to follow mobile users as they move. If an application built using our system has made connections to distributed information resources, then these connections are re-established seamlessly after movement. It is thus possible to support mobility within applications as complex as multi-user distributed multimedia applications. The support system to achieve this is provided at the middleware level, i.e. between the environment (OS, network and system services) and the application levels. The interface can thus be system-independent and the system more easily extended than if it was implemented within the operating system. One system requirement is tracking the location of users and equipment and to this end a location-awareness service has been developed. Another requirement is the availability of components for constructing mobile applications and, for this, two types of mobile object are available. Firstly, mobile agents which allow complex state migration. Secondly, mobile media endpoints which provide lightweight migration for objects with a minimal amount of state and which benefit from minimum implementation overheads. A scheme for locating and binding to mobile objects is also outlined. We have implemented and experimented with the system described. The applications we have built include a mobile multimedia conferencing system in which the sessions of a mobile user can pop up on the nearest workstation so he/she can continue to communicate with colleagues whilst on-the-move.

1. Introduction

1.1 Motivation

The motivation for this work is to provide support for user mobility within distributed multimedia applications. Mobile telephony has indicated the attraction of transportable media endpoints. Mobile professionals who rely on computing resources can benefit from multimedia applications which are aware of user location and are themselves able to move within and between networks, remapping endpoints, such as cameras, microphones and user interfaces, on to current user locations. Mobile networking technology is not required for users to participate; computer deployment is increasing and users without networked portable computers will be able to make use of computers available in most sites they visit.

Some example applications which we can support are as follows:

- *Sending running programs to other users* – David can prepare and send a multimedia presentation about his latest project to John's current location, with an enclosed questionnaire. John can examine it, annotate it, fill in the questionnaire and send it back.
- *Location-triggered media presentations* – An application can be configured to monitor for when Jean and Ken are in the same room, go to that location and play a video clip from John.
- *Mobile multimedia* – Distributed media endpoints, such as software abstractions of cameras and microphones as well as user interface components such as video display windows, can be part of a mobile application. A simple example is tracking a user with video. When the user moves, the image switches to the nearest camera. Another example is for music to be relayed to different speakers as a person moves around a home.
- *Mobile co-operation* – An enhancement of mobile multimedia is to support mobility for a cooperating set of users. In a multimedia conference, if a user moves then the camera, microphone, speaker and user interface endpoints must be moved to the new location. Connections to the other users involved must then be re-established.

We envisage the most common requirement will be supporting user mobility within a single computing domain. Each domain provides a set of services to support this mobility. More infrequently, a user may move to another domain. Supporting this involves a handover from one set of services to another.

1.2 Related Work

In this section we discuss three projects which have targeted mobility support for application users. ORL's *Teleporting* project redirects the user interface of a mobile user to the user's current location. The *Total Mobility* environment provides support for users who move between machines and work over low bandwidth or disconnected network links. The work on *Migratory Applications* using Obliq employs a mobile agent system which supports a mobile user interface.

1.2.1 Teleporting

Teleporting [Richardson et al., 1994] is a technique to support user mobility by the user interface of an application following a mobile user. In order to *teleport*, a user clicks a button on their active badge (a type of electronic tag: see Section 2.5) while in proximity to a workstation. Their current session then pops up on that workstation.

The teleporting system is based on the technology of the X window system. A proxy X server is able to forward X protocol requests to another server thus providing a level of indirection. The display of X applications can thus move with a mobile user. A simple database provides a mapping between current location and workstation display.

Teleporting is a powerful technique, the main advantage being that it makes any existing X application mobile. This approach also has some disadvantages in that:

- It is tied to the X system. Only the display connections follow a user; there is no dynamic reconfiguration at an application level. If it is used with any medium that does not go through X, e.g. audio, then the approach fails.
- It is potentially inefficient as control messages have to go to the original site and then be forwarded. The system performance varies in accordance with the bandwidth of the network connection between the home site and the teleport site.
- The applications do not themselves move, are unaware of their user interface moving and are not built around the premise of mobile users. This means that factors explicitly related to mobility cannot be capitalised on; e.g. the application behaving differently depending on location.

The user mobility framework described in this paper allows applications as well as their user interfaces to migrate between computers, thus avoiding these disadvantages.

1.2.2 Total Mobility Environment

The Total Mobility environment (TM) [Wachowicz and Hild, 1996] views computers as "replaceable tools, which can be hired, used and given up once no longer needed". Like the work described in this paper, it aims to support the use of locally available computing resources whenever a user moves, to eliminate the need to carry equipment when it is not useful. It addresses the problem of providing users with access to their data even when using mobile devices that are not connected to a network.

TM provides location information about users and computer equipment and a data management system based on a disconnected file system architecture. Users register and de-register for use of a mobile computer in disconnected operation. Their files are replicated at the mobile devices when they register. A logging process is used to identify modifications that users make while they are on the move and disconnected. When the user de-registers with the mobile device, or network connectivity is available again, the main copies of the files are updated with the modifications. The logging process also helps to cope with conflicting modifications made by multiple users sharing the same set of files.

Our work differs from TM in that it concentrates on support for making (multimedia) applications mobile over broadly uniform networks rather than providing generic access to user files over low-bandwidth links. TM does not address migration of applications to follow users. However, combining the two would provide migratory, follow-me applications with support for working in low-bandwidth and disconnected environments.

1.2.3 Migratory Applications

A large number of mobile agent systems have been and continue to be developed. The applications of these systems include active documents [Arnold and Gosling,1996], managing hypermedia [DeRoure et al., 1996], network management [Halls and Rooney, 1997] and information gathering [Rus et al., 1996]. One mobile agent system which has specifically linked user and application mobility is the Migratory Applications work using Obliq [Bharat and Cardelli, 1995]. This work employs the Obliq mobile agent system so that applications can move from one user to another or can be configured to follow users from machine to machine. Support for state saving in Visual Obliq allows applications to take their user interfaces with them when they move.

The Migratory Applications work is similar to that described in this paper in that it employs mobile agents and knowledge of user location. However, it produces monolithic applications for single users in which the mobile agent contains all control functionality and the user interface. The following sections describe a framework for user mobility which subsumes this functionality but also has the advantage of being able to support mobility for distributed multimedia applications involving more than one user.

1.2.4. Conclusions

The main limitation of these projects is the lack of mobility support for distributed, multimedia and cooperative applications. Teleporting moves only the X user interface and does not consider reconfiguration of multimedia streams. TM is concerned with uniformity of access regardless of location but does not address real-time issues. With Migratory Applications, only what is encoded within the application moves; there is no notion of reestablishing existing communication channels after movement.

1.3 Overview

We elected to build the framework described in this paper at the middleware level; i.e. between the environment (operating system, network and system services) and the application. This is so that our system is not dependent on any particular operating system. Also, since it is not within the operating system, new components can more easily be added over time. Figure 1 outlines the main components in our framework.

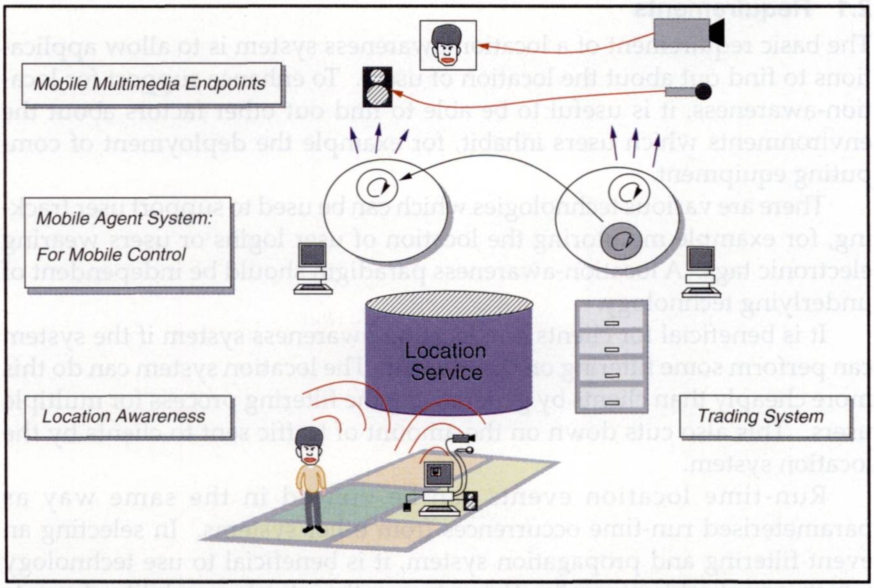


Figure 1: Components in our mobile applications framework

The first requirement of our system is to allow applications to monitor the locations of users. To this end we have designed and built a location-awareness system, described in Section 2.

It must be possible to move applications or components of applications in response to changes in user location. One technology we deployed to this end is a *location-aware mobile agent system*, described in Section 3. Agents can detect user movements and respond by moving to the user's nearest computer or taking other action.

Our experiments showed that mobile agents were too heavyweight for use as media components in mobile multimedia applications, e.g. as video sources, processors or sinks. To address this problem we developed lightweight moveable media objects, described in Section 4.

It must be possible to locate and bind to objects, such as mobile agents or multimedia endpoints, before communication is possible. In Section 5 we describe how trading technology can be adapted for naming and locating mobile objects.

The integration of location-awareness and mobile objects into a mobile application support framework is described in Section 6. The deployment of and experimentation with this framework is described in Section 6. Section 8 summarises and concludes the paper.

2. Supporting Location-Awareness

2.1 Requirements

The basic requirement of a location-awareness system is to allow applications to find out about the location of users. To enhance support for location-awareness, it is useful to be able to find out other factors about the environments which users inhabit, for example the deployment of computing equipment.

There are various technologies which can be used to support user tracking, for example monitoring the location of user logins or users wearing electronic tags. A location-awareness paradigm should be independent of underlying technology.

It is beneficial for clients of a location-awareness system if the system can perform some filtering on their behalf. The location system can do this more cheaply than clients by generalising the filtering process for multiple users. This also cuts down on the amount of traffic sent to clients by the location system.

Run-time location events can be viewed in the same way as parameterised run-time occurrences from other systems. In selecting an event filtering and propagation system, it is beneficial to use technology which is applicable generally. This enhances the interchangeability of events between different systems, encouraging interoperability.

2.2 A Model for Active Location

We have developed a mechanism of event-based programming which we use to support location-awareness [Bacon et al., 1995]. The motivation behind event-based programming is that many applications, including those associated with mobility, are *active* in nature; i.e. run-time actions are triggered in response to asynchronous occurrences (events). This situation is complicated within an open distributed context since applications can be composed of components running on separate machines, each of which is a potential source of events. It is desirable to avoid providing interfacing code to handle occurrences from every potential source.

To address these circumstances we have developed a generalized model in which events are transmitted as parameterised run-time occurrences. An event service is defined as any network service which publishes a specification of classes of occurrence it can inform a client of. Clients register interest in the set of occurrences they wish to be informed of. Services notify clients asynchronously if an occurrence matching the registration criteria is detected. The registration paradigm enables scalable and flexible construction of active applications and avoids communication saturation by filtering at source and/or intermediate nodes.

The event approach aims to augment rather than replace current distributed programming methods (such as CORBA [OMG, 1991]). We have therefore extended an existing Interface Definition Language (IDL) to handle events. This enables a server to declare, in a strongly-typed way, the events it can notify. It also allows a client to see the server's specification of events and to select those of interest.

In the case of location-awareness, applications can register interest with a location service (see Figure 2). The service can advertise location monitoring facilities, declared in an IDL, for example:

```
LocEvent: EVENTCLASS [ person : USER,location : PLACE ];
```

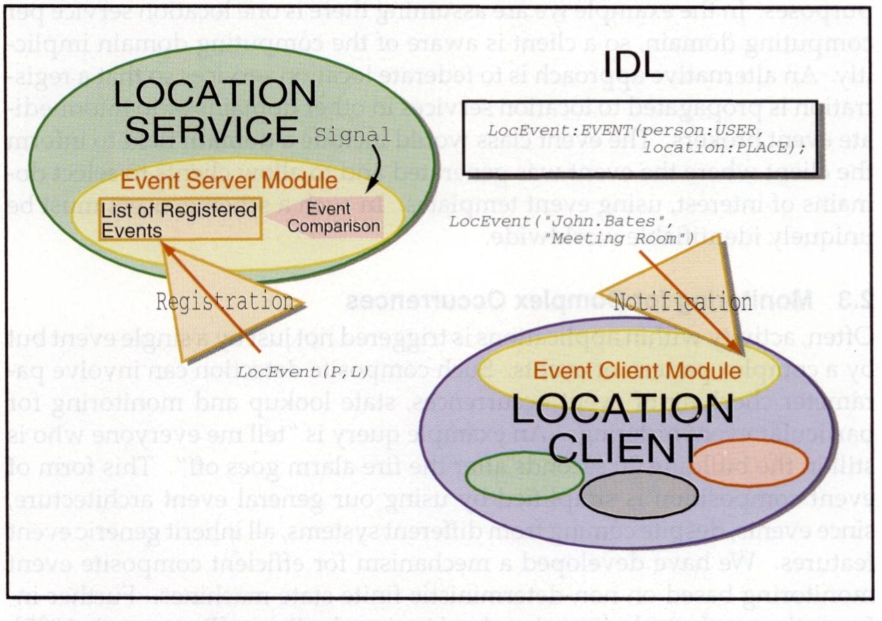


Figure 2: Event-based location awareness

A location service inherits the properties of an event service; i.e. generic support for handling and storing registered events, for comparing signalled instances with those registered and for notifying any matches. A mechanism for querying the specific location technology is contained within the location service. Related work is currently underway which involves the correlated use of multiple location technologies to increase the accuracy and flexibility of a location service [Nelson, 1997].

An event client inherits facilities for registration and for handling notifications. Structural information generated by processing an event interface can be used by both client and server for the transmission of event instances. A client provides an *event template*; the parameters of the event are given as values which must be matched, or variables which are wildcards. Examples of location event templates are:

- **`LocEvent("John.Bates", L)`** — Report wherever John is seen
- **`LocEvent(P, "Meeting Room")`** — Report when anyone is seen in the meeting room
- **`LocEvent(P, L)`** — Report when anyone is seen anywhere.

When event instances are notified to clients, all parameters are given; e.g. the instance, **`LocEvent("John.Bates", "Meeting Room")`**, means John has been seen in the meeting room.

In the above example we have simplified the event class for illustrative purposes. In the example we are assuming there is one location service per computing domain, so a client is aware of the computing domain implicitly. An alternative approach is to federate location services so that a registration is propagated to location services in other domains or to intermediate event brokers. The event class would include a **domain** field to inform the client where the event was generated and to allow clients to select domains of interest, using event templates. In such a scheme, users must be uniquely identifiable worldwide.

2.3 Monitoring for Complex Occurrences

Often, activity within applications is triggered not just by a single event but by a complex pattern of events. Such composite detection can involve parameter checking of event occurrences, state lookup and monitoring for particular event orderings. An example query is "tell me everyone who is still in the building 20 seconds after the fire alarm goes off". This form of event composition is simplified by using our general event architecture, since events, despite coming from different systems, all inherit generic event features. We have developed a mechanism for efficient composite event monitoring based on non-deterministic finite state machines. Further information on the techniques involved is given by Bacon [Bacon et al., 1995].

2.4 Environment Location Information

It is beneficial if other information is available to enhance location-awareness, for example the following types:

- The names of rooms and their geography; i.e. how they interconnect. This is a useful measure of proximity. For example, monitoring can be extended to detect when two or more users are on the same floor of the building
- The equipment stored in each room. This information can either be static or equipment can be electronically tagged. Using this informa-

tion it is possible to determine whether a specific room or part of a room offers a certain facility, e.g. live video

- The names and capabilities of pieces of equipment. For example, the architecture, operating system, hardware facilities, attached peripherals and software configuration of a workstation.

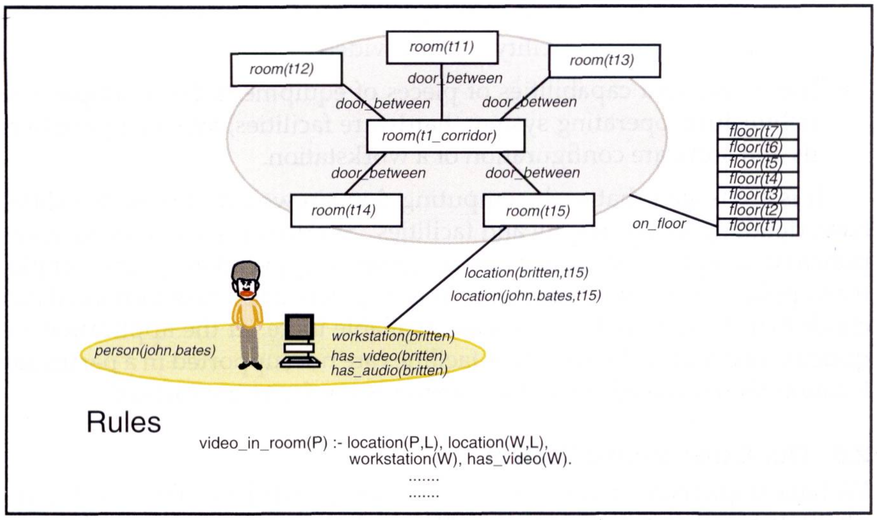
It is envisaged that each computing domain would offer such a database, detailing its geography and facilities. Such a service is a useful component of adaptive location-aware multimedia applications. For example, if an application is following a user or is dispatched to a new location it can check that the required facilities are available there. If the application requires video and audio and these facilities are not supported in a particular location then it can adapt and use text-only conferencing instead.

2.5 Our Experimental System

We have implemented our event architecture as part of an ODP-ANSA compliant distributed programming platform [Bates, 1996] and are working on integrating it with a CORBA platform. Within this work we have enhanced IDL processing tools to generate stubs for event code. We have also developed server and client libraries to provide generic event operations, such as registration and notification.

Using this technology we have built a location service and a location information database. The location technology employed by the location service is the active badge system [Harter and Hopper, 1994] deployed in our laboratory. Each user wears a badge which periodically transmits a unique infra-red signal. A network of detectors allows the location of the user to be pinpointed. Our location service collects badge events and then passes them to the event server library using the operation **Signal**. If an event matches any registered templates, the event system will notify the appropriate client(s).

The location information database contains data about the organisation of rooms and the names and capabilities of equipment stored in each room. Our implementation was based around an entity-relationship data model, which we implemented in Prolog. An example from the model is shown graphically in Figure 3. This illustrates how the interconnected geography of a building, containment of items in rooms and item properties can be described. The database is given an initial set of data and is updated using events describing the latest locations of people and equipment. It is possible to add new rules as well as to submit queries to the database. Examples are, "is there a workstation with video capability at John's current location?" **`(video in room(john.bates`** — see Figure 3) and, "is there a route out of the building avoiding my supervisor?!"



```
video_in_room(P) :- location(P,L), location(W,L),
                    workstation(W), has_video(W).
                          .......
                          .......
```

Figure 3: Entity-relationship model from location database

3. Location-Aware Mobile Agents

3.1 Mobile Agents

Mobile agent systems facilitate mobile applications. Application code and execution state, sometimes including the user interface, can be migrated from one host to another. Each host within the system must run a service which can send, receive and execute agents. Mobile agent systems differ from process migration systems in that they are implemented in user space, allowing system independence. Many useful applications of such systems have been documented (see Section 1.2.3).

In this work we introduce a new range of applications of mobile agents. By making location information available to mobile agents, the integration of user mobility monitoring and mobile applications is facilitated. This enables mobile agents to be sent to users rather than to a named host. By using location information, the host nearest to the user's current location can be determined. Many existing applications involve mobile agents traversing a number of hosts. Location information enables agents to traverse a number of user locations. Most interestingly, it is possible for mobile agents to follow users by triggering movement in response to location events. This is beneficial since much state is set up in the course of using an application and it is a waste of time and resources if this state has to be recreated manually at various hosts. If the user becomes mobile for an extended period of time, it is possible to persistently store agents on disk or transportable smart card.

The alternative to using mobile agents is for an application to create and control an object remotely. This method involves factories and protocols.

Factories are distributed services which can create objects of certain classes. Protocols which are understood by both client and server are used to manage the process of object creation and control. Mobile agents have various advantages over using such an arrangement. Firstly, it is quick to prototype a new application, since no compilation or service setup is required. Creating and controlling an application remotely involves network traffic, whereas a mobile agent for this purpose can be sent to an execution point near to the resource required. Also, mobile agent platforms are system-independent, so there is no requirement to comprehend an system-dependent API.

There are two main ways in which mobile agents can be used for location-oriented applications. Firstly, the mobile agent can be the entire application, containing all functionality including the user interface. Secondly, the mobile agent can be used for high-level mobile control. In this case the agent is responsible for setting up and controlling distributed connections using other services. It is possible that these services are distributed around the environment, however, if they are not location-specific they can be linked dynamically to a mobile agent. This enables a wider range of applications than coding the entire application into one monolithic agent. For example, as will be described later, distributed multimedia applications involving multiple users can be supported. Using mobile agents for control also aids the management of heterogeneity. By binding dynamically, mobile agents can access domain-specific functions. Mobile agents are appropriate for this high-level management role since it can be prototyped rapidly and moved flexibly.

3.2 The Tube: Our Mobile Agent System

We have gained experience from building and deploying our own mobile agent system, the Tube [Halls, 1997]. This section describes features of the Tube, including many which we found essential for its use in an active location-aware environment. The integration of the Tube with other environment features to support mobile multimedia applications is described later in Section 6.

For a host to be able to send and receive mobile agents it must run an instance of a Tube site. Agents are written in the Scheme language (a dialect of Lisp); they are transmitted over a network as marshalled expressions. If tagged as executable, they are passed to an interpreter for execution. The marshalling respects duplicate objects and cyclic references. We are able to transmit the full range of expressions, even closures[1] and continuations[2]. This means that these agents can create arbitrary functions

[1]A closure is a function together with its defining scope.

[2]A continuation is a closure that represents the current state of execution. Calling a continuation results in the computation resuming from where the continuation was captured.

and send them elsewhere, with their closing environments.

Through being able to transmit continuations, an agent can be stopped, moved and restarted elsewhere in a single call. The Tube does much more than just execute byte-code compiled scripts remotely — agents can modify themselves, create and dispatch other agents and treat their functions and state as first class, transmissible data. Agents execute in a safe interpreter and are only allowed to access those functions that the platform's owner makes available.

Tube sites are multi-threaded. Full access to a POSIX threads interface provided by the underlying operating system is given to agents. A noticeboard is provided, which an agent can post messages to or read messages from. Data values can thus be left by an agent for others that may arrive later to use. The noticeboard can be divided into different areas; an access control list is associated with each.

Tube sites can dynamically load at run-time compiled libraries of code and call functions contained in them. This allows facilities to be added without having to stop and recompile.

A user interface toolkit has been integrated with the Tube. Functions to create and manipulate widgets are available to agents. We have also enhanced the toolkit with the ability to return the state of any user interface, along with any callbacks registered on widgets, as a series of bytes. A corollary function takes a saved user interface and recreates it in a visible form. This allows an agent to create a user interface on one machine and retain it as embedded state. We have also written a Netscape plug-in that allows the Tube to be used for writing applets embedded in World-Wide Web pages.

The core state-saving functionality of the Tube is written entirely in Scheme. It is portable across Scheme interpreters and compilers, and thus also across operating systems. Code to interface with other systems, such as POSIX functions or a user interface toolkit, is not portable between operating systems. Currently, the implementation is Unix-based. We use the Bigloo [Serrano, 1994] Scheme compiler/interpreter and the XForms [Zhao and Overmars, 1996] user interface toolkit. Versions are running on DEC Alphas under Digital Unix, SUN SPARCs under Solaris, Intel Pentiums under Linux and HP 9000s under HP-UX.

4. Mobile Multimedia Endpoints

4.1 Requirements

The use of distributed multimedia introduces new problems for mobile applications. In this discussion we are assuming a system is available which offers an object-based abstraction of media source and sink endpoints, such as cameras, microphones, video displays and speakers. These objects can be created on distributed hosts and connected together flexibly using stream abstractions, e.g. a video stream connecting a live video source to a video window. This is reasonable since such systems have been developed successfully [Wray et al, 1994].

To move one or more objects involves transfer of object communication state. For example, assume David is looking at a video window of Kerry which is connected to a live video capture object at Kerry's location. If Kerry moves then we must remap the video source onto the nearest camera and reconnect to David's window. Such re-routeing of streams is known as *handoff*. This is commonly provided by the network layer (see for example [Rajagopalan, 1995] or [Porter and Hopper, 1994]), though application support for *handoff* has been proposed [Pope, 1996]. Our implementation of *handoff*, described below, is done entirely in user-space and is specialised for multimedia objects which can tolerate data loss. We plan to capitalize on research which is being carried out into allowing application-specific policy for mobility to be closely bound at runtime with network control functionality [Rooney, 1997].

Media endpoints can be implemented as mobile agents. It is more likely that they would be built using more traditional distributed programming techniques (e.g. an enhanced CORBA platform) and created dynamically at run-time using distributed factories. The assumptions behind this reasoning are as follows:

- that an application involving media connections will have some form of intelligent controlling program, responsible for setting up the connections. In the case of a multi-user program, each user may have such a program. Therefore this program can be responsible for location-awareness and subsequent moving of connections
- media objects generally perform simple memory-less functions and therefore do not need the advanced state-saving of a mobile agent system such as the Tube
- media functions generally involve the manipulation of large amounts of data and thus should be as fast as possible. Mobile agent systems involve more heavyweight execution overheads due to code interpretation and migration mechanisms. It is therefore often desirable to use compiled code for media objects.

4.2 A Media Object Implementation

In our implementation we support mobility of media endpoints in user-space, by saving the communication state of media objects and using this information to re-instantiate a copy in the new location (see Figures 4–6). Media objects are themselves responsible for re-establishing connections to the objects they were previously connected to.

We have developed media endpoints as active objects, autonomous entities which can send and receive messages and process data concurrently.

Each object class is developed to support distribution, with an interface that can include invokable methods, events that the object can notify (see Section 2.2) and connectable stream endpoints. Stream endpoints are named and typed; for example `VidSource` and `VidSink` of type `VideoStream`.

In order to configure a multimedia session, first the appropriate objects must be created on the appropriate hosts and then they must be connected together. Media objects are created by making a creation request of an object factory on the appropriate machine, by spawning an object as a new process or by linking dynamically to the object code and instantiating a new instance. Factories are services at which objects of certain classes can be created. For instance, on every workstation with live video capture facilities there will be a factory capable of creating live video source objects.

If an object must be created remotely or a connection to an existing object is required then an interface reference for the appropriate object/ factory must be located. A trader is used to this end. Object references are stored in the trader using attributes to aid retrieval (see Section 5.3). For example, a live video source factory can be annotated with the `location` attribute set to `Meeting room`. If multiple cameras are available in one location then they can be distinguished with an attribute describing their proximity to computing equipment or active badge sensors. When objects are created they are allocated unique attributes to distinguish them from other objects of their class.

In order to support stream operations we have extended a distributed programming platform. Our implementation is available on DEC Mips running Ultrix and Alpha platforms running Digital Unix. Enhanced stub generation tools build functions to read and write stream data and generate interfaces to allow third parties to set up and connect stream endpoints. Third parties use a library, written to manage this process. Once interface references have been obtained, the third-party client invokes a `Connect` operation with the two objects in question as parameters. The underlying implementation is that the source object is sent a message detailing the sink object. The source and sink objects make a stream connection, handshake and exchange their unique naming information. Data can then be sent from source to sink using stubs generated by processing the object interface. Our implementation of the stream library uses sockets. Appropriate protocols can be selected to satisfy particular application requirements.

The stream library supports a mechanism to allow a third party to access the communication state of an object. The object stream library supports using this information to re-instantiate the streams when required. The trader from the domain in which the other object(s) existed is contacted (see Section 5.3) to get the reference to its stream interface. A stream connection is then re-established.

4.3 Implemented Mobile Media Objects

We have developed various media object classes using our enhanced distributed programming platform. These include sources for live and stored video and audio, audio players, video windows and shared drawing and text components. There is also a component which enhances a Web browser for mobility. It queries the URL of the page currently being accessed by the Netscape browser on a particular host, using Netscape's socket API. Using the API on another host it can instruct the browser running there to display the saved URL.

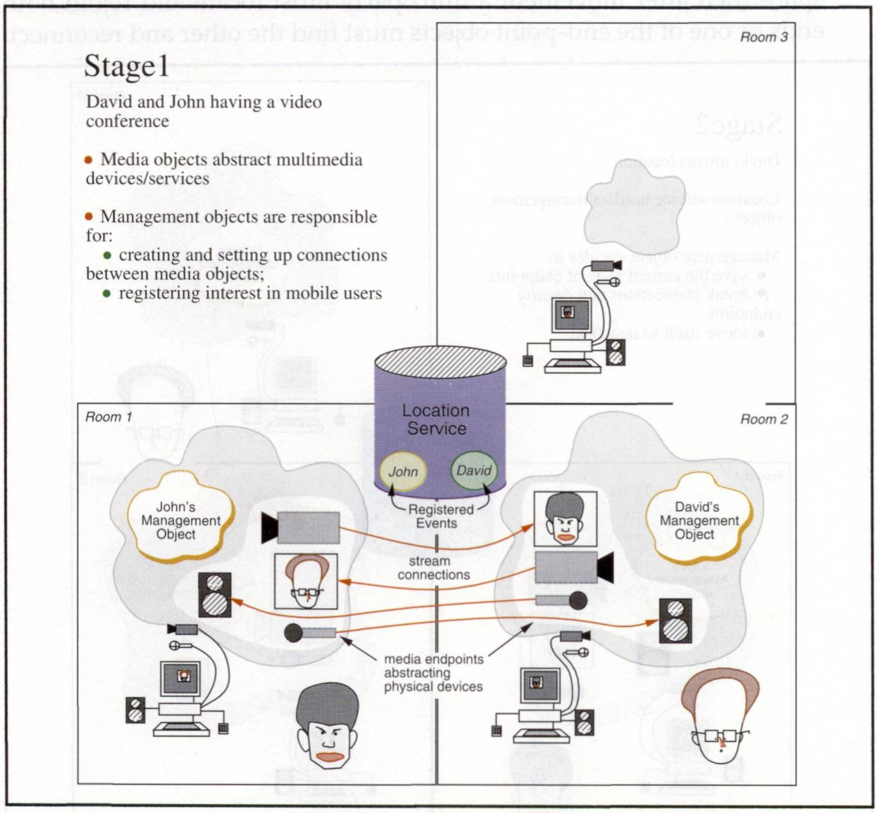

Room 3
Stage1
David and John having a video conference
• Media objects abstract multimedia devices/services
• Management objects are responsible for:
• creating and setting up connections between media objects;
• registering interest in mobile users
Location Service
John
David
Room 1
Room 2
John's Management Object
David's Management Object
Registered Events
stream connections
media endpoints abstracting physical devices

Figure 4: Stage 1 in moving an application instance

5. Naming and Location of Mobile Objects

5.1 Requirements

We have introduced two types of mobile object: mobile agents and mobile multimedia endpoints. For objects to communicate, the remote object must first be located (in physical terms a mapping to host and port) and the two objects bound. There are several circumstances in which location of remote objects for communication must be determined, for example:

- If a specific class of multimedia endpoint is required on a specific host, then the factory on this host must be located
- For a new user to bootstrap themselves into a distributed cooperative application, it may be required to bind to another user's instance and run a join protocol
- If two multimedia endpoints are joined using a stream and one endpoint moves, then the stream must be re-established. If this is done in user space then after movement a third party must locate and rejoin both ends or one of the end-point objects must find the other and reconnect.

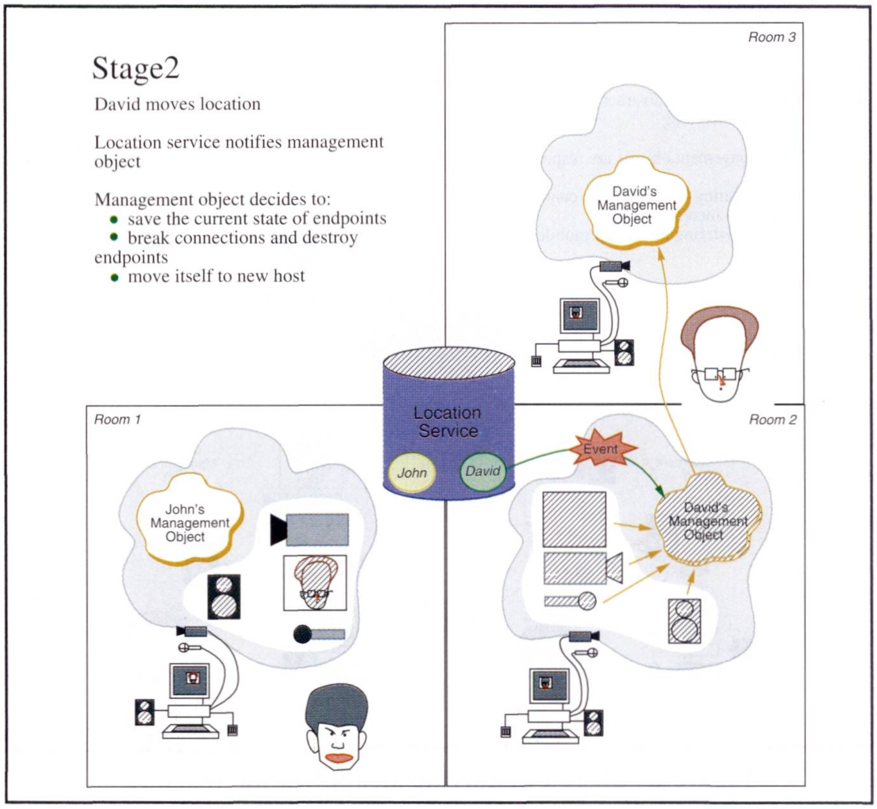

Stage2
David moves location
Location service notifies management object
Management object decides to:
• save the current state of endpoints
• break connections and destroy endpoints
• move itself to new host
Room 3
David's Management Object
Location Service
John
David
Event
Room 1
John's Management Object
Room 2
David's Management Object

Figure 5: Stage 2 in moving an application instance

5.2 Trading Services

The traditional way of locating an object is to use a trading service. An object's interface reference is required to bind with the object for communication purposes. Interface references are registered in a trader's namespace, indexed by the object class and other naming information. If objects move, interface references are of no use, since they encode physical location information. Regular access to trading information is thus a requirement in a mobile object system. An object naming scheme should allow the advertisement of specific characteristics to assist in locating a specific instance, for example name, class, owner and the host on which it runs.

If an object moves then a mechanism to alert applications which may try to contact it is required. During movement, the object can be marked in a trader as *mobile* or the object can remain alive to inform clients of its status. If the object moves to another domain it must be possible to locate it. One method is to federate the namespaces of trading services. Another is to leave *tombstones* (forwarding information) in a trader to enable objects to be located in their new domain. Updating references is at the discretion of moving objects. Additionally, this method introduces a problem of garbage collection. A further mechanism is to associate a home trader with each object and keep it up-to-date with the object's current location.

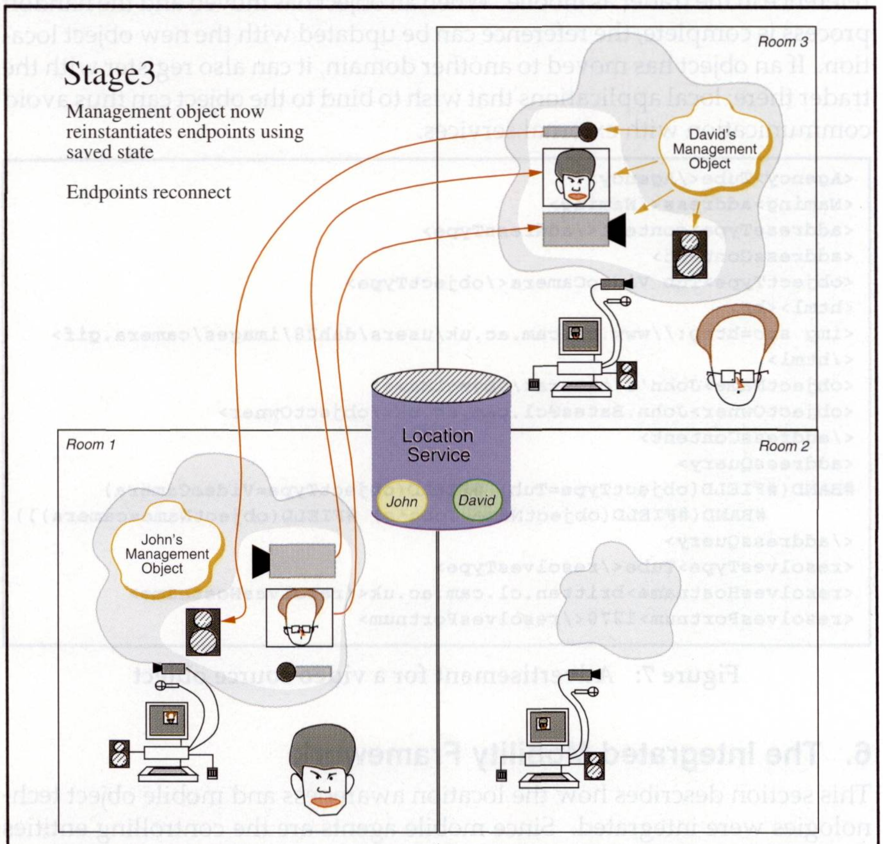


Figure 6: Stage 3 in moving an application instance

5.3 An Example Implementation

We have built a trading service which supports a rich object naming scheme, based on SGML tags. An object advertises its functions with a trading service, e.g. the video source object in Figure 7. Clients of the trader can query objects associatively, for example a search, **`objectName contains "John", objectType == "Tub VideoCamera"`** would return the object from Figure 7 and any others matching the search criteria. Tags are used at the discretion of applications. For example, the **`html`** tag allows arbitrary HTML markup to be inserted for use by Web browsers. We have implemented a graphical interface to the trading service, using the Web (see Figure 8). By using information from traders to update large information repositories (analogous to Web search sites like **`http://altavista.digital.com`**) it is possible to search and locate active services worldwide, based on their characteristics.

In our implementation, an object in the process of moving can mark its reference in the trader as mobile. When an object has moved and the handoff process is complete, the reference can be updated with the new object location. If an object has moved to another domain, it can also register with the trader there; local applications that wish to bind to the object can thus avoid communication with external services.

```
<Agency>Tube</Agency>
<Naming>address</Naming>
<addressType>content</addressType>
<addressContent>
<objectType>Tub VideoCamera</objectType>
<html><br>
<img src=http://www.cl.cam.ac.uk/users/dah28/images/camera.gif>
</html>
<objectName>John's camera</objectName>
<objectOwner>John.Bates@cl.cam.ac.uk</objectOwner>
</addressContent>
<addressQuery>
#BAND(#FIELD(objectType=Tub) #FIELD(objectType=VideoCamera)
       #BAND(#FIELD(objectName=John's) #FIELD(objectName=camera)))
</addressQuery>
<resolvesType>Tube</resolvesType>
<resolvesHostname>britten.cl.cam.ac.uk</resolvesNostname>
<resolvesPortnum>1270</resolvesPortnum>
```

Figure 7: Advertisement for a video source object

6. The Integrated Mobility Framework

This section describes how the location awareness and mobile object technologies were integrated. Since mobile agents are the controlling entities in a mobile application, integration work was centred on the enhancement of our mobile agent system.

6.1 Enhancing Mobile Agents for Management

The Tube has been enhanced to make it location-aware and able to configure distributed multimedia objects. In our integrated architecture, the Tube is a client of both the location service and the multimedia framework. To enable this we linked the relevant client code into the Tube.

Mobile agents are able to register interest with the location service and, due to the multi-threaded implementation of the Tube, handlers can be set up in scripts to respond to asynchronous notification of user movements.

Scripts can also create objects on remote hosts, connect stream endpoints together, invoke operations on media objects and save communication state. Other versions of media objects have been developed which enable dynamic linking. These are more tightly coupled to a mobile agent as they run in the same address space. It is also possible to write user interface components using the Tube directly since it contains the state-savable XForms library.

6.2 Communication Systems

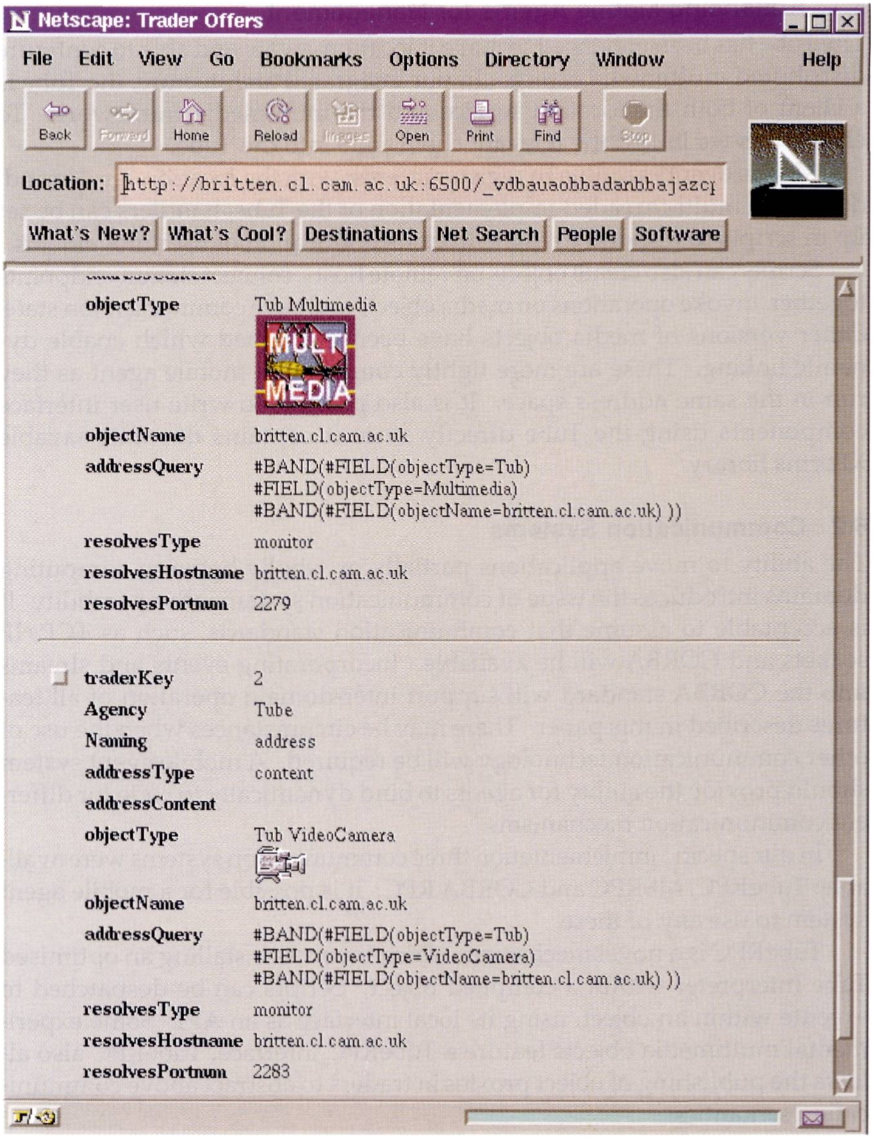
The ability to move applications partially or wholly between computing domains introduces the issue of communication system interoperability. It is acceptable to assume that communication standards, such as TCP/IP sockets and CORBA will be available. Incorporating events and streams into the CORBA standard will support inter-domain operation of all features described in this paper. There may be circumstances when the use of other communication technology will be required. A mobile agent system should provide the ability for agents to bind dynamically to code for different communication mechanisms.

In our specific implementation three communication systems were available: TubeRPC, MSRPC and CORBA RPC. It is possible for a mobile agent system to use any of these.

TubeRPC is a novel mechanism which involves installing an optimised Tube interpreter within a compiled object. Scripts can be despatched to execute within an object, using its local interface as an API. Some experimental multimedia objects feature a TubeRPC interface. TubeRPC also allows the publishing of object proxies in traders to abstract above communication semantics.

7. Experiments

Multimedia workstations on our local area network run instances of multimedia object factories and Tube sites. We deployed individual instances of our trading and location services. We wrote a number of experimental applications using this apparatus.


Netscape: Trader Offers
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
What's New? What's Cool? Destinations Net Search People Software
MULTI MEDIA

Location: http://britten.cl.cam.ac.uk:6500/_vdbauaobbadanbbajazc

objectType	Tub Multimedia
objectName	britten.cl.cam.ac.uk
addressQuery	#BAND(#FIELD(objectType=Tub) #FIELD(objectType=Multimedia) #BAND(#FIELD(objectName=britten.cl.cam.ac.uk)))
resolvesType	monitor
resolvesHostname	britten.cl.cam.ac.uk
resolvesPortnum	2279

traderKey	2
Agency	Tube
Naming	address
addressType	content
addressContent	
objectType	Tub VideoCamera
objectName	britten.cl.cam.ac.uk
addressQuery	#BAND(#FIELD(objectType=Tub) #FIELD(objectType=VideoCamera) #BAND(#FIELD(objectName=britten.cl.cam.ac.uk)))
resolvesType	monitor
resolvesHostname	britten.cl.cam.ac.uk
resolvesPortnum	2283

Figure 8: Trader advertisements in a Web browser: a multimedia object factory and a video camera

7.1 Experimental Schedule

The first phase of experiment involved building a number of small applications to illustrate rapid prototyping using Scheme scripts. We illustrated the principle of specifying host machines by their proximity to users rather than their name. Tube mobile agents have the ability to go directly to a particular Tube site or perform a traversal of a number of sites. Using the location-awareness information, it is possible instead to traverse the hosts nearest to one or a number of users. For example, a questionnaire about the time of a meeting can be sent to all proposed participants, one after another. The next iteration is signalled by clicking on a button.

For the second phase of experiments we carried out various multimedia experiments. One example is *follow-me* video and audio. Follow-me video is useful if one wants to monitor a particular person continuously, e.g. a child-minding application. The video display window follows the application user, and the live video source follows the person being monitored. The follow-me audio application allows music to follow a user around their environment. Additonally, we have developed a follow-me Netscape Web browser, which pops up showing the last used Web page whenever a user moves.

The third phase of experiments involved developing a complex illustration of our architecture in action: the mobile multimedia conferencing application, shown in Figures 4–6. We built two versions, centralised and decentralised, in order to study their comparative behaviour and performance. In the decentralised version, each user has a mobile agent controlling their source and sink endpoints. The agent registers interest in the movement of its own user. When the user moves, the agent receives an event detailing his/her new location, saves the state of its objects, moves itself to the new location and recreates the objects using the saved state information. The objects are responsible for reconnecting themselves to the media endpoints of other users.

Instead of using mobile agents, the centralised conference, known as *Mobocop* (see Figure 9), holds all information about connections in an agent managed by the chairman. The Mobocop agent only moves to follow the chairman. This version has the advantage that it is much simpler to author and user movement is managed slightly more rapidly (see below). The decentralised version is more elegant and scalable.

7.2 Measurements

Timings were taken using the conference applications described in the previous section. An audio and video conference was set up between three people with access to three DEC Alpha computers, each in a different room, equipped with a video camera, microphone and speakers, connected to a local-area ATM network and running a Tube site under Digital Unix. One of the users was designated chairman and the Mobocop application was able to follow him as he moved.

Each user was able to see and hear the others as they moved between rooms. All the timings given below include a constant overhead of 2 seconds for the active badge system to report badge sightings to Tube sites.

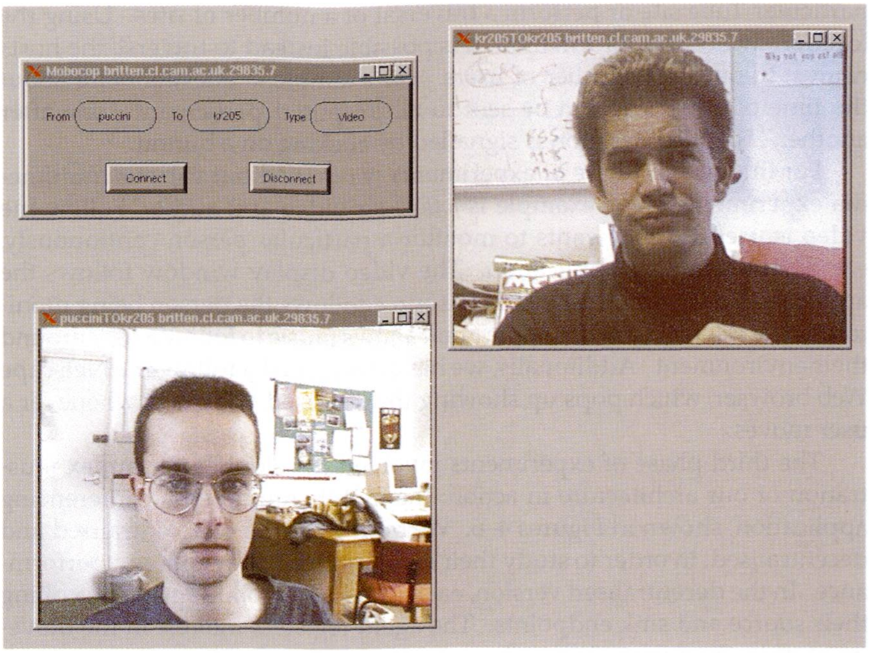

Mobocop britten.cl.cam.ac.uk.29835.7
From puccini
To kr205
Type Video
Connect
Disconnect
kr205TOkr205 britten.cl.cam.ac.uk.29835.7
pucciniTOkr205 britten.cl.cam.ac.uk.29835.7

Figure 9: The Mobocop video conference manager

The average time taken for a user's multimedia objects to appear at his destination when he moved between rooms was 8.9 seconds (815 milliseconds to close the old connections, 1.614 seconds to destroy the old objects, 2.934 seconds to create the new objects and 1.509 seconds to recreate the connections between the new objects). The time taken between him arriving at a new room and the other users seeing him there was 9.2 seconds on average. Movement of the controlling agent adds an additional 500 milliseconds to this total. This is experienced in the centralised version only when the chairman moves and in the decentralised version when anyone moves.

These times proved adequate for conferences with only sporadic movement. They are also respectable given the heavy load placed on the computers. Each user had outgoing audio and video connections to the other users and to himself. That is, each user had six outgoing connections and there were eighteen connections in total. The video objects had to send and receive 1.33 megabytes of data over each connection per second (there were 12 frames per second, each of which was 384 by 288 pixels in 8-bit colour). The audio objects had to send and receive 22 kilobytes of data over each connection per second (it was sampled at 11kHz, using 16 bits per sample).

The DEC Alpha computers were overloaded by this amount of data, so that the playing of audio became intermittent and the display of video jerky.

Handling of user mobility had to compete with the processing of this multimedia data, without help from the Digital Unix scheduler which cannot impose quality of service guarantees. Use of an operating system designed to provide such guarantees would have helped here [Leslie et al., 1996]. Performing stream handoff at a lower level, instead of tearing down and re-building connections, would have reduced the amount of software involved in re-routeing data (see Section 4.1). Using a separate device for sending data from video cameras to the network [FORE, 1997] would have reduced the load placed on the computers.

Note that the related projects cited in Section 1.2 give no performance measurements. Our system is experimental and we plan to work on improving these times significantly.

8. Summary and Conclusions

We have investigated general-purpose support allowing applications to follow mobile users, for example by hopping to the nearest computer. We have built and deployed a framework to test our ideas.

A location monitoring system is required to track users and to provide information about resources available in various locations. Providing filtering support in a service so a client is only informed about location events of interest is realistic and cuts down on network traffic and client processing. Clients should be informed asynchronously when events of interest occur.

A mobile agent system allows script-level programs to move from site to site whilst retaining user interface and execution state. Multimedia endpoints can be made mobile by providing support for the saving of stream connections and their re-establishment after movement. A mechanism for locating mobile objects such as agents and multimedia endpoints is required. Standard trading technology can be used here, assuming an appropriate naming scheme and a policy of indicating objects which have moved to another domain.

Our integrated architecture places mobile agents at the centre of application coordination. We have found this paradigm works well since scripts are easily prototyped and agent support works flexibly between heterogeneous domains. We have also found that compiled media endpoints, built using distributed programming mechanisms, such as CORBA, are appropriate as mobile media endpoints. Our initial experiments are encouraging; we have found we can support complex applications, such as a multimedia conference in which users can move between hosts and connections to other users are re-established automatically.

Acknowledgements

David Halls gratefully acknowledges the support of EPSRC for his graduate studentship and John Bates would like to thank Michael and Morven Heller and St Catharine's College, Cambridge, for supporting his research. All three authors extend their thanks to ICL for general support of their research group.

Bibliography

K. ARNOLD and J. GOSLING, "The Java Programming Language," Addison-Wesley, 1996.

J. BATES, "A framework to support large-scale active applications," SIGOPS European Workshop, ACM, 1996.

J. BATES and J. BACON, "Supporting interactive presentation for distributed multimedia applications," Multimedia Tools and Applications, 1(1), 47–78, March 1995.

J. BACON, J. BATES, R. HAYTON and K. MOODY, "Using events to build distributed applications," 2nd International Workshop on Services for Distributed and Networked Environments, IEEE, 1995.

K. BHARAT and L. CARDELLI, "Migratory Applications," Proc. ACM Symposium on User Interfaces Software and Technology, Pittsburgh, USA, November, 1995.

D. DEROURE, W. HALL, H. DAVIS and J. DALE, "Agents for Distributed Information Management," Presented at Practical Applications of Intelligent Agents and Multi-Agents (PAAM '96), London, UK, 1996.

FORE Systems, Inc., "FORE ATM Video Solutions, 1997," URL
http://www.nemesys.co.uk/products/video/index.html.

OBJECT MANAGEMENT GROUP, "The Common Object Request Broker: Architecture and specification," Technical Report 91.9.1, Object Management Group, December, 1991.

DAVID HALLS, "Applying Mobile Code to Distributed Systems," PhD thesis, University of Cambridge Computer Laboratory, June, 1997.

A. HARTER and A. HOPPER, "A distributed location system for the active office," IEEE Network, 8(1), 1994.

D. HALLS and S. ROONEY, "Controlling The Tempest: Adaptive management in advanced ATM control architectures," Position paper, January, 1997.

I. LESLIE et al., "The design and implementation of an operating system to support distributed multimedia applications," IEEE Journal on Selected

Areas in Communication, September, 1996.

G. NELSON, "System Support for Location Aware Computing," PhD thesis, University of Cambridge Computer Laboratory, 1997 (in preparation).

J. PORTER and A. HOPPER, "An ATM Based Protocol for Wireless LANs," Technical Report 94-2, Oracle Olivetti Research Limited, Cambridge, 1994.

S. POPE, "Application Support for Mobile Computing," PhD thesis, Jesus College, University of Cambridge, October, 1996.

B. RAJAGOPALAN, "Mobility Management in Integrated Wireless ATM Networks," Proceedings of Mobicom, IEEE, Berkeley, 1995.

T. RICHARDSON, F. BENNETT, G. MAPP and A. HOPPER, "Teleporting in an X Window System Environment," IEEE Personal Communications, August, 1994.

D. RUS, R. GRAY and D. KOTZ, "Autonomous and adaptive agents that gather information," Proceedings of the AAAI 96 Workshop on Intelligent Adaptive Agents, 1996.

S. ROONEY, "The Structure of Advanced ATM Control Architectures," PhD thesis, University of Cambridge Computer Laboratory, 1997 (in preparation).

M. SERRANO, "Bigloo User's Manual," INRIA, B.P. 105, Rocquencourt, 78153 Le Chesnay Cedex, France, 1.7 edition, 1994.

S. WRAY et al., "The Medusa applications environment," 1st IEEE International Conference on Multimedia Computing and Systems, 1994.

M. WACHOWICZ and S. G. HILD, "Combining location and data management in an environment for total mobility," Proceedings of the International Workshop on Information Visualization and Mobile Computing, Rostock, Germany, February 1996.

T. ZHAO and M. OVERMARS, "XForms,"

URL http://bragg.phys.uwm.edu/xforms, 1996.

Biographies

John Bates

John Bates holds the Heller Senior Research Fellowship in Object-Oriented Computing, at St Catharine's College, Cambridge and his research is based at the University of Cambridge Computer Laboratory.

The main thrust of his work is in applying object and event technologies, to support end-user requirements of distributed computer systems. Areas of application to date include multimedia, CSCW and support for mobility.

David Halls

David Halls recently completed his PhD at the University of Cambridge Computer Laboratory. His research involved the investigation of the potential applications of mobile code. He now works for Persimmon, a Cambridge-based IT company.

Jean Bacon

Jean Bacon is a Lecturer in Computer Science at the University of Cambridge Computer Laboratory. She teaches in the systems area and is the author of books on computer architecture and concurrent systems. Her research interests are in distributed computing, focussing on system support for emerging applications which include multimedia.

INDEPOL Client—A 'facelift' for mature software

S. B. Southerden

ICL Enterprises, ICL, Bracknell, UK

Abstract

The purpose of this paper is to describe the lessons learnt through the development of a client component of a mature software product which runs in a high performance server environment. The original product, INDEPOL, (**IN**telligence, **DE**fence and **POL**ice) was developed in the early 1980's by ICL Defence Systems in collaboration with the Ministry of Defence. INDEPOL, which provides a toolkit for generating applications and runs on VME systems under TPMS, was designed to exploit to the full the CAFS Information Search Processor.

1. Introduction

Research in ICL, during the late 1970s and early 1980s, had demonstrated that the standard relational model, although satisfactory for many business applications, was semantically weak in those situations where both data types and structures were complex [Crockford, 1982]. This led to the development of a prototype system based on binary relations and a directed graph representation of the database schema, in which both fixed format data, loosely structured text and free text could be handled in a homogeneous manner [Maller, 1996]. INDEPOL, a fourth generation system, was based on this research and was designed specifically to meet the requirements of both the military and the police intelligence market.

The pairing of INDEPOL and CAFS provided a powerful capability to retrieve information, containing a mix of structured data and/or free text, with very short response times. The product was first launched in 1986 and is now considered 'mature'.

2. INDEPOL Background

It may be helpful to the reader if a brief description is given of the type of applications for which INDEPOL is used. The key feature of the application is that it will be investigative; i.e. the primary purpose is to investigate the data and particularly relationships within the data. Secondary features are likely to include:

- the value of information over time, leading to the building up of large libraries of data
- data comprising a mix of numeric information and text (either may be used) singly or in combination for retrieving records
- data of variable structure and variable length
- the improbability of being able to specify in advance the nature of the users queries and the need therefore to iterate through queries
- the need to support effective searching with fuzzy matching (wildcard) and 'near miss' (quorum searching) selection
- the need for fast response times
- the need to be able to specify the content and format of the results of enquiries
- in some cases the need to provide a data entry capability for users to add new records
- in other cases, data is imported from some other source (including external to VME systems)
- special security applied to field and content level.

Real applications for which INDEPOL has been used include:

- crime reporting and criminal intelligence (police)
- share fraud investigation (police and financial services)
- management information of corporate data such as personnel, logistics and events (defence, commercial including ICL)
- audit trail analysis (government)
- fraud investigation (police, government)
- Royal Commissions (government).

Significantly, several of the current applications may fairly be described as data warehouses: users have taken snapshots of operational logistics databases and used INDEPOL applications to provide a management information source for their enterprise. This demonstrates how many of the features, originally specified for INDEPOL to satisfy a specialized set of users, have been delivered in a form permitting generic applicability.

Other lessons which were learned from exploitation of INDEPOL by customers included:

- the benefits of rapid prototyping and incremental system development
- the ability to support a multi-user interactive query system on very large volumes of data

- the resistance of some non-technical users to creating free format queries (and in other cases the reluctance of IT departments to give them this capability)
- and consequently, in some instances, the under exploitation of INDEPOL's features by some customers.

Sales of INDEPOL were made between 1986–1993 and the last release of INDEPOL (310) was made in December 1992. This was declared to be the last version which would be produced and the development team was disbanded and the product passed into support. The marketing strategy for further enhancements has been to build 'surround' products which could add value to the types of applications for which INDEPOL was being used without change to the core, hence providing added value while retaining stability and improved quality for the customers. Examples of such products and utilities include Intelligence Analyst Workbench (IAW) [Southerden, 1992], High Speed Data Matching (HSDM) and INDEPOL Data Update and Conversion (IDUC), the last two being developed by Peter Sweetser, then of ICL Australia.

Prior to the decision to make 310 the final version, the ICL Product Authority commissioned market research, which was carried out over a six month period in 1991, to identify potential future requirements for the product as well as to test the hypothesis that a business case existed for a UNIX version to be developed. This research became known as INDEPOL-X and took cognizance of the developments then being undertaken within ICL into SCAFS [Martin, 1994]. While the market review captured valuable and supportive information, both from many of the INDEPOL customers and the sales channels within ICL, the commercial decision did not support the investment required to build a new product.

The story then moves forward to 1995, by which time INDEPOL 310 had been released to customers and, more importantly, installed by all of them. It has to be recognized that most, if not all, of the customers were running INDEPOL as part of a suite of services and the introduction of the new version had to take the appropriate priority alongside their other business activities. It was therefore not until mid-1994 that all sites had moved their operational applications over to the new version.

Other technologies had advanced over the period since the INDEPOL-X study. In particular, the proliferation of PCs was leading to a desire for those facilities, which were becoming available in PC software, to be provided within the INDEPOL environment. This had to be considered, however, in the context that some organizations were still using 'dumb' terminals and for security reasons did <u>*not*</u> want PC type facilities to be added for their users.

While some steps were taken to provide a PC type interface, typically through the provision of a 7561 terminal emulator, neither 'tight' integration, nor a true Client component had been developed.

At the end of 1994 the INDEPOL-X report was reconsidered by the Product Authority and further investigations were made to determine users' key requirements. The results of this study established that the following facilities should now be added:

- an improved interface including a windows environment with drop down menus and icons, integrated business graphics, seamless interface into word processing and spreadsheets, cut and paste capability between INDEPOL data and other applications, integration with images, photographs and fingerprints
- new development tools including painting tools for screens and reports, easy to develop macros
- provision of searching capabilities, including ability to save 'hit' files, and the performance of joins to be raised to, at least, the standard of relational databases
- new features to be made available to applications, including the provision of improved text entry, word wrap, ability to sort data once retrieved and graphical representation of data.

While this list is not an exhaustive description of all the points made, many respondents emphasized requirements, already met by INDEPOL, in order to highlight essential features which any new development to replace INDEPOL must possess.

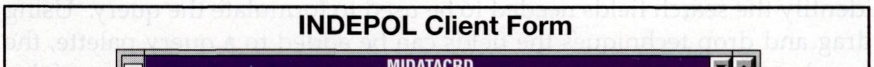
The 'business drivers' generating these requirements included the need to reduce technical support in the formulation of complex queries and exporting data, the desire by users not to have to 'double handle' data after retrieving it in order to put it into a more understandable display format, and increasing recognition of the value of the accrued data, if only it could be exploited by more users who were increasingly resistant to the monochrome screen display and the command query language.

Other work was being carried out within ICL to provide similar kinds of capabilities to other VME based and non-ICL databases. One of these [Thompson and Robertson, 1994] describes the background and development of the Dialogue Manager product and Beer [Beer, 1994] provides a description of a similar project with one customer and also identifies some of the technical issues which have to be considered at a generic level. Contact was also made with a development group who had provided a similar capability on two ICL internal systems, SAMIS and Configurer, and in particular had addressed two problems: firstly how to keep PC based (client) software updated when the VME (server) application changes and, secondly, how to provide a maintainable and replicable communications infrastructure for connecting Client applications to the server. As is so often the case, solutions are potentially available from a variety of sources but real world issues such as the releasing of those resources or the provision of appropriate levels of skills transfer can preclude immediate benefits from being realized. Fortunately on this project a collaboration was possible.

A case was formulated to develop INDEPOL Client and for this to be a true client component of the INDEPOL 310 product, with requirements to deliver a targeted set of the enhancements identified in the INDEPOL-X study. Approval to proceed was given in mid-1995 and a development team set up, comprising one newly joined graduate, a part time team leader and some pre-existing software and technical consultancy from a collaborating unit within ICL.

In support of the business case, a prototype of some of the key features was developed in Visual Basic. These features included drop down menus and several of the 'standard' offerings of Windows based clients. Features which particularly stimulated discussion and interest included a visual representation of the data model, which we called the data map, exploitation of business graphics and integration with a set of geographic tools.

The first implementation with a customer focused on data entry, not an area where a Windows GUI would obviously be of benefit as data entry staff tend to have fast keyboard skills and mouse controls are likely to slow down input. The implementation demonstrated the advantages of validating the data in the Client as the application did not permit backward navigation between templates. The client solution allowed several pages of data to be available with tabs allowing any one to be brought to the front as required.



Mikey	B01G 1GA31 3920251014488	Nato Stock Number	3920251014488	Proc-NSN	3920251014488		
Stats-Pd	01	Stats-Year	95/96	Cat-Store	01	Division	B
Prov-Grp	G	PSC	1G	DMC	A31	ICC	3
Depot	B	Dep-Loc-Group	B5BE024C	SGC	B	Prime-Location	5BE024C
Building	5B	Location	BE024C	RSAC	E		
Stg-Classn-Code	B	SS-Code	70	Stg-Res-Code	2	Pack-Code	A
Stktg-Group	B0860	Stktg-Gp-Ref-no	B0860				
Stktg-Ser-No	086	Stktg-Code	B	Stktg-Type-Code	0	Pre-Iss-Insp-Code	0
Stk-Serv-Code	0	Disp-Res-Code	C				
It-Buy-Type	1	Sh-Life	00	Hz-Cat	00	Short-Item-Name	BELT,WAIST
D-Of-Q	EA	Basic-Price					
Price-Val-Code	8.702	NSC-Average-Price	B	Contr-Skel-Ind	48.292	Scld-Item-Ind	0
EIN-Ind	0	SSN	N				

Figure 1: An illustration of multi-page data

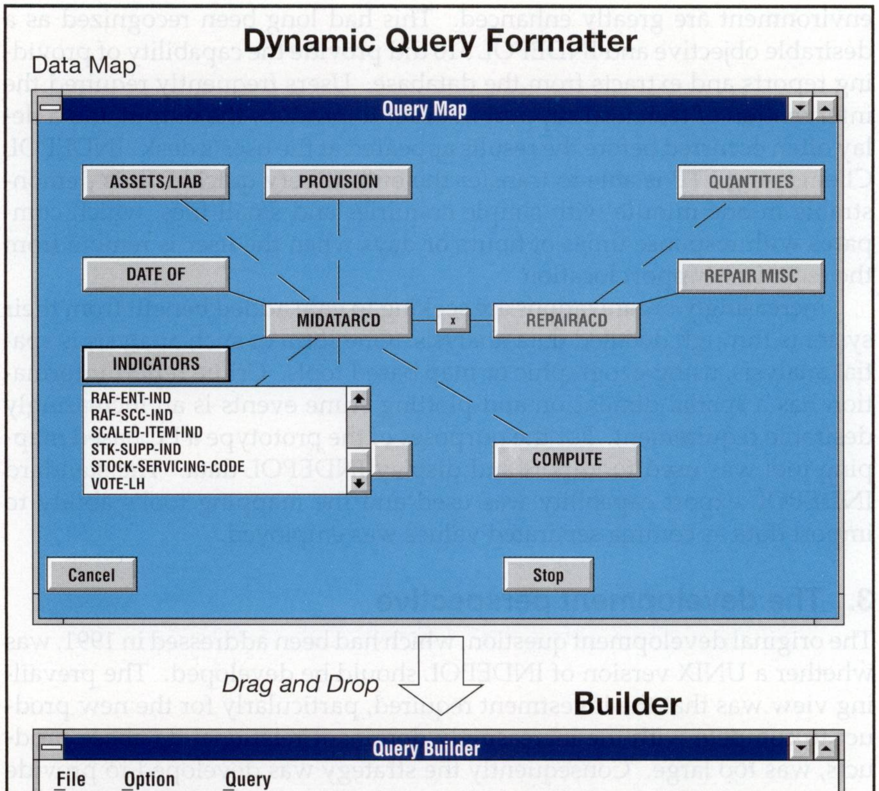
The INDEPOL application already carried out validation at the field level, as well as cross field validation. The rules for this validation are held within the data model and so are easily incorporated within the Client. The extent to which further cross field validation might be carried out is a trade off between a more complex product, in which all data is entered through INDEPOL Client, and the benefits which may be achieved by reducing the size of the data model. There are advantages affecting the supportability of the INDEPOL application if the data model remains the source of such validation tables.

The prototype suggested how improved information could be made available to users to assist them in searching the database. Applications represent real world objects and may include files such as people, addresses, vehicles, telephones and documents. A police application with this information may be required to support a query to find all people aged 25 – 40 years, over six feet tall, who own or are known to be connected with a red Ford saloon car having 'N' as a prefix in the registration mark and who live in Kent. Such a query would require INDEPOL to perform a join across three files (people, vehicles and addresses) and is likely to be carried out using the query language rather than by completing a template. While the query language of INDEPOL is 'English-like' the precise syntax which has to be followed can be prone to error, particularly by inexperienced or infrequent users.

The INDEPOL Client data map provides a graphical schema of the data which, together with drop down menus for each file, allows the user to identify the search fields needed to be used to formulate the query. Using drag and drop techniques the fields can be added to a query palette, the search values added, and the Client will then translate the contents of this palette into a syntactically correct INDEPOL query. A safety check could potentially be constructed at this point, inhibiting searches which require more than, say, three file joins. It is not unknown in some user sites, where there are many tens of files, for queries to be asked which take many hours to run because of the many file joins required.

The prototype and an early customer implementation demonstrated the extraction of data into an Excel spreadsheet. Excel provides various tools to turn data into business graphics and this working example showed how commonplace software products can be used for providing management information in a readily understandable form. Without INDEPOL Client, applications are limited to producing the output as tables or histograms. The power of office tools from any supplier, is enhanced by the capability of importing data direct from the source and presenting the analysis in colour, in graphics, and in a document.

While graphical display is the desired end product, the analytical capabilities which can be performed by moving data into a different software



OP	Query	Value
	MIDATARCD.DMC	a31
	MIDATARCD.BASIC-PRICE	>100

Display: MIDATARCD.DMC, MIDATARCD.NSN

Result Selection: [✓ All] Sample Count CSV

A/PAGE 9 OF 9 LAST

Figure 2: Dynamic Query Formatter

environment are greatly enhanced. This had long been recognized as a desirable objective and INDEPOL 310 did provide the capability of providing reports and extracts from the database. Users frequently required the intervention of technical support in order to produce the output and a delay often occurred before the results appeared at the user's desk. INDEPOL Client using FTP is able to transfer the output very quickly. It is demonstrable in one minute with simple enquiries and small files, which compares with response times of hours or days when the user is remote from the technical support location.

Increasingly organizations are seeking to gain added benefit from their systems through detailed data analysis. One form of such analysis is spatial analysis, using geographic or map based tools. Crime report information has a spatial dimension and plotting crime events is an increasingly desirable requirement. For the purposes of the prototype a PC based mapping tool was used to import and display INDEPOL data. The standard INDEPOL export capability was used and the mapping tool's ability to import data as comma separated values was employed.

3. The development perspective

The original development question, which had been addressed in 1991, was whether a UNIX version of INDEPOL should be developed. The prevailing view was that the investment required, particularly for the new product to compete with the increasingly dominant relational database products, was too large. Consequently the strategy was developed to provide 'surround' tools and utilities which would add value for INDEPOL applications while the core product remained unchanged and therefore stable.

Discussions with existing customers showed them to be generally happy and loyal to the product. Predictions were made which forecast that there would be a dramatically reduced INDEPOL base by 1995, brought about by migration to relational products and general strategies of down-sizing. In reality this did not happen. So when the 1991 documents were re-examined in 1995, the adopted strategy identified the benefits of developing an improved interface and offering additional features, as well as exploiting a client-server architecture. Existing products were investigated which might have provided a basis for exploitation or collaboration, in particular, Dialogue Manager, and the view was formed that a major part of the requirement was to decode INDEPOL templates. INDEPOL is a macro based language which can display 'fixed' or pre-set templates, or dynamically form them on the fly from parameters entered by a user as an output requirement to a query statement. Both approaches are supported with equal efficiency. Picking out fields from a dynamically formatted template requires flexibility in specifying where a field is. Often this is done by matching the prefix or suffix string, but the logic behind the provision of such facilities comes from an understanding of the INDEPOL template mechanisms.

To validate this approach an exercise was constructed to take the research and map new functionality on to a Client Server architecture. This identified four core sets of users:

1. those who primarily used INDEPOL templates either for data entry or enquiry
2. those who predominantly used the data for simple investigation, research or analytical purposes using the free format facilities
3. experts, who typically built queries on behalf of others and may be experts in category 2 or technical support staff
4. technical developers and support staff of the applications.

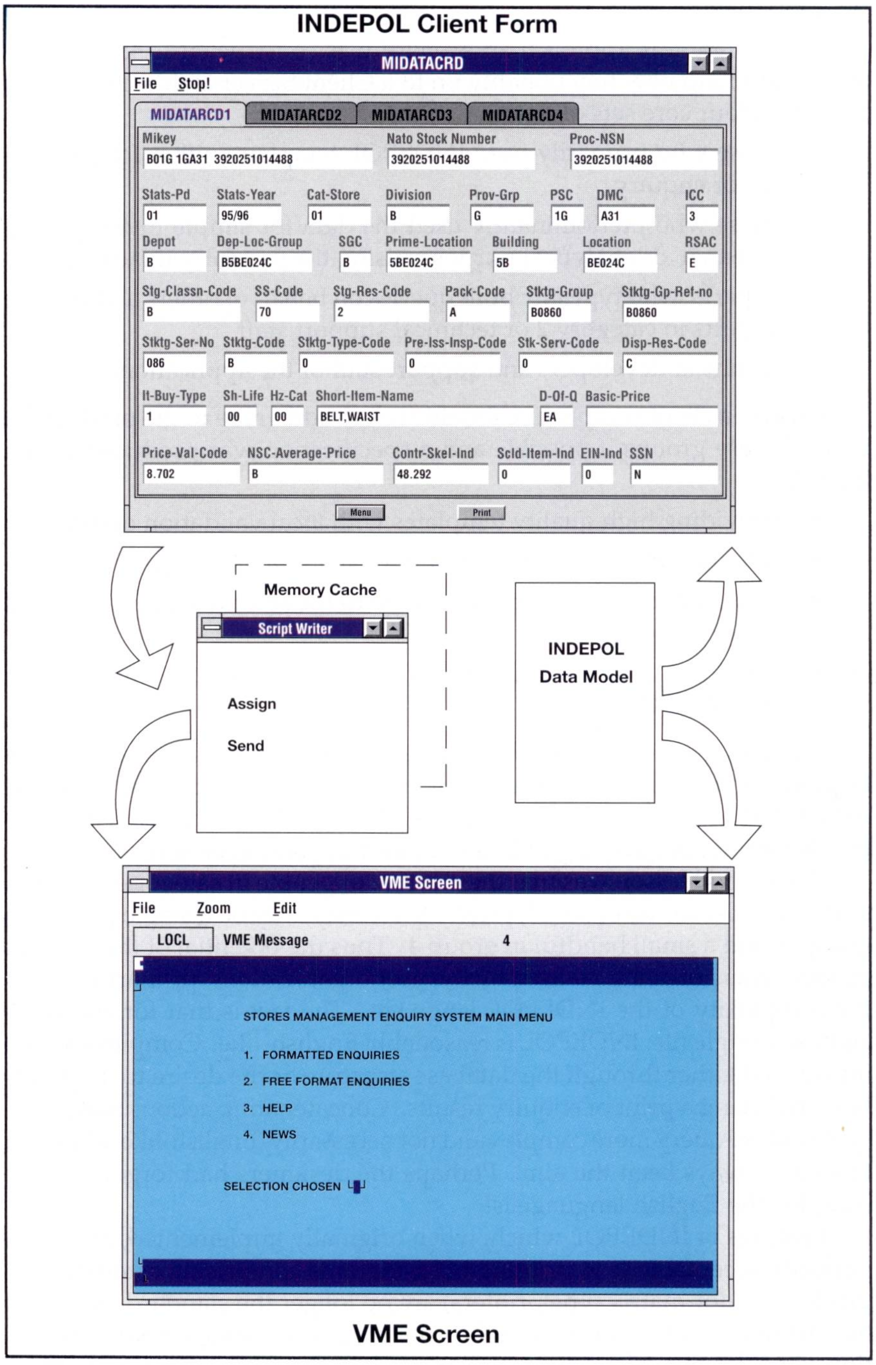
Clearly a client component should seek to add value to the product for each of these groups if possible and a specification was produced which included:

- providing high quality templates with local validation of data
- retaining the characteristics of the INDEPOL language
- ensuring that all facilities available to the second group continued to be supported
- providing tools for support staff actually to create the application on INDEPOL Client.

Two lessons were learned from this exercise. Firstly, a system could not be produced that replicated all existing templates or, at least, doing so would have led to a closed application and overall loss of functionality. The problem, therefore, of querying INDEPOL 'head-on' had to be solved.

The second lesson was that the number of people in categories 2–4 was small on a site to site basis. Typically there may only be a single person in group 3, and a small handful in group 4. Thus the potential of the original product was not being realised by the user organizations, primarily due to the complexity of the INDEPOL language. The fact is that for enquiries against a single file, INDEPOL is reasonably English-like. Complexities are introduced either through the database structure or the desire to do things like producing a print of enquiry results. Concatenating actions makes the syntax of the query more complex and not necessarily 'English-like' although this had always been the aim. Perhaps the designers had forgotten how complex the English language is!

Features of INDEPOL which, when originally implemented, had been perfectly acceptable (e.g. printing of histograms comprising asterisks and printed via dot-matrix type printers) are no longer the standards required by customers in the late 1990s when the output of management information via PC spreadsheet tools provides attractive, 3-dimensional, colour charts, perhaps incorporating the logo of the department or enterprise pro-



Mikey	B01G 1GA31 3920251014488	Nato Stock Number	3920251014488	Proc-NSN	3920251014488		
Stats-Pd	01	Stats-Year	95/96	Cat-Store	01	Division	B
Prov-Grp	G	PSC	1G	DMC	A31	ICC	3
Depot	B	Dep-Loc-Group	B5BE024C	SGC	B	Prime-Location	5BE024C
Building	5B	Location	BE024C	RSAC	E		
Stg-Classn-Code	B	SS-Code	70	Stg-Res-Code	2	Pack-Code	A
Stktg-Group	B0860	Stktg-Gp-Ref-no	B0860				
Stktg-Ser-No	086	Stktg-Code	B	Stktg-Type-Code	0	Pre-Iss-Insp-Code	0
Stk-Serv-Code	0	Disp-Res-Code	C				
It-Buy-Type	1	Sh-Life	00	Hz-Cat	00	Short-Item-Name	BELT,WAIST
D-Of-Q	EA	Basic-Price					
Price-Val-Code	8.702	NSC-Average-Price	B	Contr-Skel-Ind	48.292	Scld-Item-Ind	0
EIN-Ind	0	SSN	N				

```
STORES MANAGEMENT ENQUIRY SYSTEM MAIN MENU

1.  FORMATTED ENQUIRIES
2.  FREE FORMAT ENQUIRIES
3.  HELP
4.  NEWS

SELECTION CHOSEN
```

Figure 3:

ducing the information. Similar quality output was required to support INDEPOL reports and highlighted the need for the INDEPOL data to interwork with PC applications. The prototyping phase of the development confirmed the feasibility of providing this level of integration through demonstrations of interworking with a spreadsheet and a mapping application. The philosophy behind the design was not to reinvent wheels but to exploit tools already available in the PC environment.

Experience within the development team of interacting with legacy systems led INDEPOL Client to be designed around three concepts:

1. the definition of templates
2. a window painter
3. a scripting language to implement interactions between 1 and 2.

Collectively these facilities became known as the 'personality' for a specific implementation.

A debate occurred on whether a script language was needed or whether Visual Basic should be used. Visual Basic has the advantage of being both quicker and more flexible, but it could not be constrained and therefore the benefits of using the INDEPOL system model to achieve consistent validation would have been lost. Using Visual Basic a simple working system could have been produced within an estimated three weeks, but it would not have gone further than displaying the VME screen and would not have delivered the value added features that were desired.

A further essential component was the INDEPOL system model interrogator, which retrieves information about fields in the application, their names, size and screen positions.

It was apparent that to support the appropriate levels of de-bugging which might be required a mechanism was needed to move one step at a time through scripts and to be able to examine variables. To facilitate this, a screen capture mechanism was developed which also allowed off-site development (i.e. away from the INDEPOL application) and supported a free standing demonstration capability.

To address the query component of INDEPOL, a 'drag and drop' approach was produced which allowed the user to drag field names on to a query formatter which ensured that the correct INDEPOL syntax was constructed.

Finally, to retain the spirit of the Server application, the ability to store INDEPOL queries, and subsequently to re-load, modify and re-launch those queries was provided.

To support these features a visual representation of the INDEPOL database can be incorporated which allows users to look up fields, perform 'drag and drop' query building and to understand the complexity of the search they may be seeking to perform, with the consequential impact on response times for their query. A significant improvement was offered through this approach, as the validation of a query could be performed by the Client before launching it to INDEPOL. This ensures that only valid queries get to the Server and that the number of searches which fail are considerably reduced.

The results of a query are displayed to a user in a table. This was set up so that if a table entry contained a unique key then the INDEPOL Client window showing the full details of the appropriate entry was displayed. This mechanism is crucial in bridging the gap between template users and free format query users. The need to type lists of data items has been removed.

4. Customer experiences.

The development of INDEPOL Client was assisted by the first customer organization to commit itself to the product. Their desire was to provide a mechanism which, in addition to providing an improved data capture/ entry mechanism, also allowed other data which was available at the same time, to be keyed in and then directed to a second (non-INDEPOL) system. The original decision to include the INDEPOL data model interrogator paid off handsomely and underpinned the swift painting of screen templates. It also helped to improve the legibility and consistency of the application. The development of scripts was also relatively quick and this enabled the user's navigation of the application to become visible at an early stage.

The limitations of Visual Basic 3 in its screen handling abilities with large numbers of fields were learnt to our cost. The physical limits and performance, which were documented, had seemed acceptable for the size of INDEPOL applications, but the data capture requirements of the first customer exceeded these design limits.

The second customer implementation was carried out by way of a pilot exercise to install up to ten copies of INDEPOL Client with a full 'personality' to enable all the facilities of the customer application to be used. It was agreed that the pilot period would be six weeks, divided equally into three weeks of implementation and three weeks of testing and feedback by the users. Prior to the use of INDEPOL Client the customer's system would be searched to obtain a sub-set of data which would be displayed at a terminal. There was an expectation that further manipulation of that data would be done using a spreadsheet package. This would have caused the user particular problems:

1. if the user wanted to count Totals, all the detail, potentially many hundreds of records, would have to be imported into the spreadsheet, yet the totalling facility already existed within INDEPOL.
2. some of the user's macros produced computed fields which were

sufficiently important to the users that it was unacceptable to make it a two stage process by using a spreadsheet. For example, the concept of stock value exists which can be derived in a number of ways according to circumstances. The write off value of the stock may be used, or alternatively a cost value. Dependent upon the values used, a complex series of calculations can be performed across a number of the data attributes. These could be replicated in a spreadsheet but a better quality result is obtained by performing these functions in INDEPOL as:

- a centrally controlled set of macros is used (providing consistency)
- no additional programming effort in the spreadsheet was required.

So the capability to exploit these macros had to be incorporated in the personality of the application.

To bring the story up to date, INDEPOL Client has now been 'ported' to a Windows NT environment. A conversion to Visual Basic 5 (via VB4) was carried out to give 16 and 32-bit versions. The previously employed communications interface was replaced by a HLAPPI interface to insulate the product from communications differences and by delegating all communications functionality to the VME Client product component on which the product now relies.

The positive reaction of users of INDEPOL has been expressed by an acceptance of some of the evidence recorded in this paper, including the under-utilisation of some of INDEPOL'S features and a serious intention to make further use of the tools through application enhancement and the use of the Client component.

5. Conclusions

With the speed of change in technology, businesses have to be increasingly selective in where they make their investments, not just because of the cost but with pressures on resources to implement change, the opportunity cost may be high. The value to businesses is in the information they have, not necessarily the technology which stores and processes the information. The advent of Client Server technology provides capabilities for adding value to the enterprise information through improved analytical capabilities which were not available in the server component, hence we have seen the development of the data warehouse. The author [Southerden, 1992] explained the intelligence process as employed in law enforcement, but the model is generic to government and business sectors. Intelligence is the process of adding value to information through the processes of collection, collation, evaluation, analysis and dissemination: our customers are seeking to obtain similar benefits, although without necessarily using these terms.

INDEPOL with CAFS provides a unique solution for effective information storage and retrieval, PC based tools provide commonly available tools which support complex analysis. INDEPOL Client now provides the 'glue' between the two which enables intelligence to be developed from the original information.

Acknowledgements

At various times during the development phase we had involvement and advice from INDEPOL experts, both from within ICL, and externally. In particular the suggestions made by members of TCMS Ltd. (who were conveniently located close to us in Slough) and by Tony Colville led to improvements in the usability of the product and we offer them our grateful thanks.

I would also like to record my thanks to Torstein Muldal (ICL Systems) who conceived the original idea of INDEPOL Client and who, as Technical Design Authority, developed and breathed life into the product, and into this paper. Any errors of fact or opinion however, are mine alone.

Bibliography

CROCKFORD, L.E., "Associative Data Management System," ICL Technical Journal, Vol. 3, Issue 1, pp 82–96, May, 1982.

MALLER, V.A.J., "*Criminal investigation systems: the growing dependence on advanced computer systems,*" IEE Computing & Control Engineering Journal, April, 1996.

SOUTHERDEN, S.B., "*Information technology: support for law enforcement investigations and intelligence,*" ICL Technical Journal Vol. 8, Issue 2, pp 302–315, November, 1992.

MARTIN, M.W., "*The ICL search accelerator, SCAFS: functionality and benefits,*" Ingenuity, The ICL Technical Journal, Vol. 9, Issue 2, pp 325–340, November, 1994.

THOMPSON, R and ROBERTSON, I., "*Dialogue Manager: Integrating disparate services in client server environments,*" Ingenuity, The ICL Technical Journal, Vol. 9, Issue 1, pp 138-150, May, 1994.

BEER, A., "*From a Frog to a handsome Prince: Enhancing existing character based mainframe applications,*" Ingenuity, The ICL Technical Journal, Vol. 9, Issue 1, pp 81-101, May, 1994.

Biography

Stuart Blair Southerden

Blair Southerden joined the Kent County Constabulary in 1967 and served as a police officer for sixteen years. As a mature student at the University of Kent he was introduced to computing studies, and on return to the force he worked for four years on a computer project team.

He left the police in the rank of inspector and worked for three years in a systems house. He joined ICL in 1987 as an industry consultant and supported the marketing work leading to the establishment of a focused line of business within ICL Australia intending to exploit the INDEPOL product. He became Product Authority for INDEPOL in 1994.

Using the ECL^{i}PS^e Interval Domain Library in CAD

T.M. Yakhno, V.Z. Zilberfaine and E.S. Petrov

Institute of Informatics Systems, Russian Academy of Science, Novosibirsk, Russia

Abstract

This paper describes the Interval Domain Library, a new logic programming tool, developed at the Institute of Informatics Systems of the Russian Academy of Sciences, for solving non-linear constraints over sets of real numbers. It accepts equations and inequalities written in standard mathematical form and produces as output variables ranging over continuous intervals. Such constraint propagation is regarded as an efficient way of processing incomplete information in what the authors have called sub-definite models. The paper also discusses some technical aspects of the integration of the library into ECL^{i}PS^e and how the resulting system may be used to solve a practical problem. The project was carried out in collaboration with IC Parc (Imperial College, London) and ICL (Bracknell).

1. Introduction

Solving non-linear constraints over real numbers is necessary in economics, mechanics, engineering, and many other areas. Due to the computational complexity and the numerical issues involved, finding all the solutions of a system of non-linear constraints with high precision is a hard problem.

Methods and tools for solving non-linear constraints can be classified into first, classic techniques, e.g. Newton's method, which is built into such systems of computer mathematics as Mathematica, MathCad, etc. and, second, interval techniques. The latter class may be split into the following groups.

The first group consists of libraries which include interval extensions of real functions [Knueppel, 1994]. Such a library is conventionally used in an imperative environment for performing explicit computations, in which users are free (or forced) to design their own algorithms for solving non-linear problems.

The second group bridges the gap between interval libraries and non-linear applications by using *constraint programming* (CP) languages as in BNR-Prolog [Older & Vellino, 1990], CLP(BNR) [Benhamou & Older, 1995], Newton, Helios [van Hentenryck, 1996]. Usually, CP systems of this kind hide the process of solution from the users.

Although a well-known CP system, such as ECL^{i}PS^e (ECRC Common Logic Programming System), provides a user with a number of libraries, such as the Finite Domains (FD) or Finite Sets library [ECRC, 1995], to help to solve discrete problems, there are no corresponding tools for solving non-linear constraints over real numbers. The Interval Domain (ID) library was designed to overcome this difficulty. It has been developed as a new logic programming tool specifically to solve non-linear constraints over real numbers. It accepts equations and inequalities written in normal mathematical form and, for each variable, it produces results in the form of continuous intervals.

All the CP systems, although they may differ in detail, do the same thing; they propagate constraints. The utility of some systems is often limited because they are free-standing. The benefit of the library approach is that it brings the power of CP methods into conventional programming languages . Another merit is that having ECL^{i}PS^e as the base language gives the user the possibility of combining the ID library with the other facilities available under ECL^{i}PS^e.

The paper is structured as follows: Section 2 briefly introduces major notions of Prolog and its dialect SEPIA, which is used in ECL^{i}PS^e; Section 3 demonstrates some of the capabilities of the ID library[1]; Section 4 focuses on some important features in the implementation of the ID library; Section 5 describes a detailed example of an application of the library and, finally, Section 6 outlines the directions of future work.

2. Prolog and ECL^{i}PS^e

Prolog is a logic programming language. It provides a means for specifying both application knowledge and declarative queries. The application knowledge is put into a program consisting of facts and rules. A Prolog interpreter infers the answer to a user's query by resolving the logical expression represented by the program in terms of the available facts. This will involve algorithms for backtracking and unification. An interpreter for Prolog always processes the program "from top down and from left to right", i.e., it tests facts and rules in their textual firing order.

The following simple program expresses parent-ancestor relations in a family:

```
parent(peter,john).
parent(peter,ivan).
parent(john,james).
```

[1] see also ftp://ai200.iis.nsk.su/pub/papers/ID/id_description.ps.gz

```
ancestor(X,Y):- parent(X,Y).
ancestor(X,Y):- parent(X,Z), ancestor(Z,Y).
```

The first three are assertions are facts and the rest are rules. The statements **ancestor** and **parent** are predicates or relations; **peter**, **john**, **ivan**, **james** are atoms and **X**, **Y**, **Z** are variables. A fact of the form **parent(X,Y)** states that **X** is a parent of **Y**. The relation **ancestor** states that (1) **X** is an ancestor of **Y**, if **X** is a parent of **Y** (the first rule) and (2) **X** is an ancestor of **Y**, if **X** is a parent of **Z** and **Z** is an ancestor of **Y** (the second rule). To find all the descendants of **peter**, it suffices to give the Prolog interpreter the following query:

```
?- ancestor(peter,Y).
```

The interpreter will then instantiate the variable **Y** to give the first answer:

```
Y = john          More? (;)
```

Upon receiving ';' from the user, the interpreter backtracks and the next answer is retrieved. This process can be continued until no more answers are found.

ECLⁱPS^e is a system based on SEPIA Prolog. Its aim is to serve as a platform for integrating various logic programming extensions [ECRC, 1995]. Its "glass box" architecture makes it possible to design extensions directly at the ECLⁱPS^e level and achieve efficiency by using various built-in predicates.

ECLⁱPS^e has a number of powerful capabilities, unavailable in pure Prolog, two of which are of major importance in the current context:

- handling delayed goals
- manipulating meta-terms

Usually, ECLⁱPS^e delays the execution of any goal with a built-in predicate, if neither failure, nor success of this goal is logically sound. For example, ECLⁱPS^e delays the goal **X =< Y** ("**X** is equal or less than **Y**" when both **X** and **Y** are not instantiated.

```
[eclipse 1]: X =< Y.
X = X
Y = Y
Delayed goals:
=<(X,Y)
yes.
```

Modification of the state of some variables (e.g., instantiation) may enable normal execution of some delayed goal, for example:

```
[eclipse 2]: X =< Y, X = 9.0, Y = 3.0.
no (more) solution.
```

Users are able to program the handling of delayed goals directly in

ECLⁱPS^e. For example, the function **sum** defined below computes the sum of a list of numbers. Obviously, it should be delayed, if the list contains uninstantiated variables, i.e., is nonground. This is done as follows using the in-line compilation facility of ECLⁱPS^e:

```
[eclipse 3]:[user].
  sum([X|Xs],Sum):- number(X), Sum is X +
sum(Xs).
  sum([],0).
 delay sum(Xs,_) if nonground(Xs).
 ^D
 user compiled traceable 676 bytes in 0.0
second(s).
yes.
[eclipse 4]: sum([X,1,3,7],S).
X = X
S = S
Delayed goals:
sum([X, 1, 3, 7], S)
yes.
```

Meta-terms assist in attaching additional information to standard Prolog variables. For example:

```
V{enum([red, white, green, blue])}
```

is a meta-term which associates a Prolog term (say a domain) with the variable **V**. Involving this information in unification makes unification stronger and helps reveal failures. The way unification works for meta-terms is defined by the unification handler. For example, the handler may check whether the value of the meta-term as instantiated belongs to the domain:

```
[eclipse 5]: V{domain([red, white, green,
blue])} = red.
V = red
yes.
[eclipse 6]: V{domain([red, white, green,
blue])} = black.
no (more) solutions.
```

Handling delayed goals and manipulating meta-terms together allows users easily to programme various constraint programming techniques, e.g., most notably constraint propagation.

In the ID library, delayed goals behave as agents; i.e., a goal, after inferring some information and letting other goals know about it, re-delays itself until a further inference is enabled by the activity of some other goal. Meta-terms are used to organise all these information flows between delayed goals.

3. The Interval Domain Library

The Interval Domain library is a new extension to ECL^{i}PS^e. The library is based on the use of sub-definite models [Narinyani, 1983] and implemented in the UniCalc system [Semenov et al., 1993]. It significantly simplifies the process of solving complicated non-linear problems.

3.1 Functions & Predicates

The syntax of terms which may appear in constraint predicates follows SEPIA Prolog conventions. The list of available functions defined in the library includes comparison operations along with extra logical functions such as basic arithmetic and trigonometric functions. In addition to these there are special predicates:

`Vars ** Domain` associates the real interval domain `Domain` with the variable or list of variables `Vars`. The domain is written as `Min..Max`, where `Min` and `Max` are real or integer numbers. If `Vars` is undefined, it becomes a domain variable. If `Vars` is already a domain variable, its new domain becomes the intersection of its old domain and that specified by the interval domain. If no initial interval can be specified, the keyword `undefined` can be used instead, which designates the interval ($-\infty$, $+\infty$).

`range(Var, Min, Max)` is used to retrieve the current domain of a variable. It returns the minimum and maximum values in the domain.

`set_accuracy(Epsilon)` sets the accuracy. The value of `Epsilon` determines the residual range of the output intervals. The solution process terminates, when all domains become smaller than `Epsilon`.

`locate(Vars)` starts procedures for locating the roots of a system of constraints. The predicate uses a branch-and-prune algorithm. Variables are taken from the list `Vars` one at a time and their domains are split. The list is treated as if it were circular. Domains smaller than the current accuracy are not split.

`assign(Var, NewMin, NewMax)` destructively sets the domain of the variable `Var` equal to `NewMin..NewMax`. The new domain may not overlap with the previous one.

3.2 Defining a New Function

The library can handle user defined functions, for example, the *hyperbolic sine*:

$$sinh(x) = \frac{1}{2}\left(e^{x} - e^{-x}\right)$$

can be defined by:

```
sinh(X, Res):- Res *== (exp(X) - exp(-X)) / 2.
```

A function of more than one argument (for example, *harmonic mean*) is defined in the same way:

```
hm(X, Y, Res) :- Res *== 2/( 1/X + 1/Y ).
```

A function can accept an arbitrary number of parameters, which may be either Prolog terms or domain variables. For example, the harmonic mean of a list of domain variables may be defined as follows:

```
hm(L, Res):-
  sum_inv(L, Sum),
  N is length(L),
  Res *== N / Sum.

sum_inv([X], Res):- !, Res *== 1/X.
sum_inv([X|Rest], Tot):-
   sum_inv(Rest, SubTot),
   Tot *== SubTot + 1/X.
```

This function can now be used in the following constraint:

```
hm([X1,X2,X3,X4]) + hm([S1,S2,S3]) *==
hm([Z1,Z2]).
```

3.3 Using the Library

All variables used in non-linear constraints, must be initialized first. The syntax for initialisation is:

```
<Variable> ** <MinValue> .. <MaxValue>
<List of variables> ** <MinValue> .. <MaxValue>
```

e.g. `V ** 0.5..9.2` turns `V` into a meta-term with the range `0.5..9.2`. `V` can then be used in a constraint expression, e.g. `sqr(V) *=< 2`, and such constraints may be arbitrarily complex.

3.4 Some Capabilities of the Library

The process can be illustrated by finding the roots of the equation:

To solve the problem, the following program is enterd at the ECLⁱPS^e prompt:

```
X ** -1000..1000, X^3  - 3*X + 1 *== 0
```

The computation results in a range for X

```
X = [-1.8859, 1.5632]
```

which is guaranteed to contain all the real roots of the equation.

To separate, say, the negative roots, the inequality `X *=< 0` has to be added. The computation then produces a much narrower interval which contains the negative root:

```
X = [-1.8793, -1.8794].
```

A more precise interval that still contains the root can be obtained by changing the accuracy with `set_accuracy/1` and repeating the computation. Setting the accuracy to 10^{-8}, the following values are obtained:

```
[eclipse 1] :
set_accuracy(1e-8),
X ** -1000..1000, X^3 -3*X + 1 *== 0,
X *=< 0.

X = X_{[-1.87938526, -1.8793852]}
```

To locate all roots, other intervals need to be examined. For example, adding the condition `X*>=0, X*=<1` leads to the second root `X=[0.3472, 0.3473]` and adding the further condition `X*=>1` will lead to the third root `X=[1.532, 1.5322]`. If an inconsistent restriction is applied (say, `X*>=10`), then ECL^{i}PS^e issues the standard message `no (more) solutions`.

3.5 Locating Roots with locate/1

Another method of locating real roots employs a binary splitting. This is activated by calling the `locate/1` predicate:

```
X ** -1000..1000,
X^3 - 3*X + 1 *== 0,
locate([X]).
```

which automatically searches for roots of the equation from left to right. On backtracking, all multiple roots are returned.

4. Features of the Implementation

4.1 Constraint Propagation

The interval domains of the variables are narrowed by constraint propagation. The library decomposes each equation and inequality into *primitive constraints*, i.e., those binding two or three variables, such as `A+B*==C`. A primitive constraint essentially is a set of *calculation functions*, each being evaluated for a particular variable. For example, `A+B*==C` consists of:

```
C:= A+B (1)

B:= C-A (2)

A:= C-B (3)
```

Whenever the domain of some variable is changed, every other function that also references that variable is executed. This may in turn change the domains of yet other variables and thus trigger execution of further functions and so on.

From the point of view of implementation, each function is a delayed goal. The list of functions to be executed on changing the domain of a variable is stored in its meta-attribute **wait_any**. In the example we have:

```
A{wait_any:[1, 2]}

B{wait_any:[1, 3]}

C{wait_any:[2, 3]}
```

4.2 Using Dynamic Priorities

Experiments with the library show that over 50% of the goals are checked for consistency. The ID library therefore applies constraint propagation with *dynamic priorities* so that each time a goal fails to reduce any interval, its priority is lowered. Although this minor innovation does not extend the functionality of the library, it improves its performance significantly.

4.3 Locating Roots

Two distinct algorithms for reducing the domains of variables are implemented in the library. The first uses conventional constraint propagation. Whenever the domain of a variable is changed, the constraint handler checks all the constraints which reference that variable. This will propagate domain reduction activity into other constraints to the point where no further reductions are possible or because the variables involved now have domains within the required accuracy limits. At this stage the intervals are usually very narrow, but not always. (Some problems are not amenable to solution by constraint propagation or may yield more than one solution). Once all *a priori* constraints have been propagated, a further stage in solving the problem can be adopted, such as branch-and-prune. In operation, the constraint handler splits a variable's domain in two symmetric halves and tests whether each still satisfies the constraints. This is practicable for a single variable but gives rise to problems if many variables need to be split since performance can be strongly order dependent. Since this choice cannot be automated, it is up to the user to make a decision. A further problem of using the **locate/1** predicate is that, when the problem is large, the library may fail to find a single interval containing the actual root and will yield a number (up to ten or so) of intervals. This situation is common to a number other constraint solving systems. To reduce this undesirable behaviour, a **locate/2** predicate is available, which performs some additional checking after each split. This procedure involves splitting the domain into two parts of unequal size and testing only the smaller subdomain for a solution—for small intervals this is a relatively efficient process. If a solution exists, the part of the interval represented by the subdomain is pruned from the domain of the variable. This process of testing small regions at the ends of the interval for solutions and pruning them is solutions are found, a process sometimes known as shaving, is continued until a subdomain is encountered which does not give a solution—this is retained. The process can then be applied to the remaining variables. This process can reduce the amount of splitting actually required significantly but again at the expense of performance. The value of this technique over a symmetric split is apparent where the original interval is wide:

```
A ** 1 .. 4, Y ** 4..9, A*A *== Y.
```

Where the interval for A needs to be reduced as much as possible, local propagation cannot reduce the domain for A, nor can symmetric splitting. Although the roots for A can be found in sequence, this may not be what is wanted. If instead only the lesser parts of the split are tested then the required interval for A can be quickly reached. The predicate `locate/2` can be used for this purpose in the form:

```
locate([<List of variables>], [<List of vari-
ables>])
```

The elements of the first list are the variables to which the standard symmetric split is to be applied, those of the second list are the variables for which an asymmetric split is more appropriate. The program fragment:

```
A ** 1 .. 4, Y ** 4 .. 9, A*A *== Y,
locate([],[A])
```

will give

```
A=A_{[1.887, 3.177]}.
```

The reason this is not `A=A_{[2.0, 3.0]}` is that a condition is enforced that the lesser part of an interval should be at least 10% of the entire interval length, otherwise the procedure would be extremely inefficient. Experiments have shown, that with the right initial choice of variables for splitting, no variables need to be put into in the second list. Usually this can be achieved after a few trials. Where there are many variables, it is more effective to put some of them in the second list as symmetric splitting will sometimes not lead to a reliable result at the required precision leading to number of very narrow intervals around the actual root.

4.4 Advanced Example

A predicate for minimising a function is given as a worked example. This is based on the well-known branch-and-bound technique. The arguments of the predicate are the variable to minimise and the variables upon which it depends. The domains of the latter will be split.

```
minimise(+Var, +ListOfVariables, Minimum)
```

A sample definition of the predicate is given below. This definition is not complete, since in practice the predicate may produce neither success nor failure. This problem can be easily solved by introducing a block to infinite branching.

```
minimise(V, List, Min):-
   range(V, T1, T2),
   assign(Min, T1, T2),
   V *=< Min,
   split(V, List, List, Min).

split(_, [], [], _):- !.
split(V, [], L, Min):- split(V, L, L, Min).
split(V, [Var|Tail], L, Min):-
   range(Var, T1, T2),
   Mid is (T1+T2)/2,
   (Var *=< Mid; Var *> Mid),
   range(V, _, V2),
   range(Min, M1, M2),
   (V2 < M2 -> assign(Min, M1, V2); true),
   split(V, Tail, L, Min).
```

5. Wiring Application

An important stage in the design of an aircraft is its electrical wiring. The problem, which arises when all the mechanisms and sensors are being placed on an aircraft, is how to connect these devices into a number of systems. It is helpful to assign a different colour to each sub-system and its associated cables. The objective is to minimise the total costs while satisfying some constraints put on the parameters of the different sub-systems. A number of applications have been developed to cope with this, and related, problems, e.g.[Kuper & Wallace, 1995], [Mezhoud et al, 1996].

The above is a very general statement of the wiring problem. It neither specifies whether some interconnections require pre-determined positions nor what other constraints need to be considered, and does not define "total costs". The next sections will show how the ID library can be applied to wiring problems with constraints placed on the position of interconnections.

5.1 Formal Model

Because all the interconnections are already fixed, the topology of the aircraft is adequately represented by the graph $G = (V, E)$, where the set of vertices V includes both devices and interconnections and the set of edges E lists all pairs of vertices that can be directly connected by a cable. The cost of connecting two vertices is given by the weight function $W : E \to R^+$

All the devices of the system coloured i form the set $V_i \subseteq V$, the i-th set of required vertices. Some systems are not noise proof (e.g. sensors and gauges). Their wires must therefore be kept away from others which can induce that noise (e.g., power cables). Constraints of this kind produce a binary relation C which is true of colours i and j if and only if cables coloured i match cables coloured j.

The requirements concerning co-routing of wires depend on their location in the aircraft. This information is summarised by the function $T : E \to N$ which evaluates to the maximum number of cables wired between two vertices.

The cost function can express the real costs of the solution, or its quality, or particular preferences, etc. As many parameters are usually involved, it can be quite complex. The example will only deal with the simplest one which expresses the total length of wired cables where if the wiring cost is minimum, then necessarily each system is connected by a tree. (with the simple case of minimising the total length of wired cables, which implies that each system is necessarily interconnected by a tree.

The problem can be expressed as follows: find trees T_i such that

1. T_i connects the required vertices V_i
2. for each j, $\neg C(i,j) \Rightarrow T_i \cap T_j = \varnothing$
3. for each $e \in E$, the inequality $\sum_i I(e \in T_i) \le T(e)$ holds, and
4. the $\sum_i \sum_{e \in T_i} W(e)$ is a minimum.

Here $I(e \in T_i)$ is 1, if $e \in T_i$, and 0 otherwise. When T_i occurs inside a formula, it is treated as a set of edges.

5.2 Interval Domain Constraints

The above four items can be re-written in terms of constraints and a number of interval domain variables. For each colour i and each vertex v (edge e), the domain variable `V_i_v` (`E_i_e`) indicates that the vertex (edge) is in the tree T_i. For each required vertex v in V_i, `V_i_v` is bound to 1.

For each colour i, the graph T_i should be a tree; i.e., a connected graph with n edges and $n+1$ vertices. The latter property prompts the following constraint:

```
V_i_u+...+V_i_w *== 1+E_i_d+...+E_i_f
```

where $V = \{u, \ldots, w\}$ and $E = \{d, \ldots, f\}$.

Connectivity is define as follows. A graph is connected if and only if all its vertices are reachable from any vertex. To express connectivity of T_i, it is sufficient to select some vertex c_i in T_i, compute the distances between c_i and the other vertices of T_i, and ensure that these distances are "less than infinity". c_i is termed the centre of T_i. The centre can be selected from the set V_i.

Computing distances between c_i and the other vertices of the tree T_i causes problems, because initially T_i is not known. Instead the distances between c_i and the other vertices of the entire graph *G* are computed. These distances are computed with respect to T_i; i.e. if a path between c_i and some vertex goes through a vertex or an edge not in T_i, then its length is "infinity".

For each vertex *v* coloured *i*, the domain variable `D_i_v` designates the distance between c_i and v. Let vertices v_1, v_2, etc. be connected to $v \neq c_i$ in *G* by edges e_1, e_2, etc. Let `Infty` be a constant that is larger than the length of any acyclic path in *G* (the "infinity"). The variables `D_i_v`, `D_i_v_1`, `D_i_v_2`, etc., and `E_i_e_1`, `E_i_e_2`, etc. must satisfy the following constraint:

```
D_i_v *== min [(1+D_i_v_1)*E_i_e_1+(1-E_i_e_1) *Infty,
                   (1+D_i_v_2)*E_i_e_2+(1-E_i_e_2)*Infty, ... ]
```

which says that to reach *v* from c_i in `D_i_v` steps is equivalent to reaching its neighbour in `D_i_v-1` steps and then making the last step along some edge in T_i. The constraint can then be reduced to

```
New_D_i_v*== 1+min[New_D_i_v_1*E_i_e_1,
                   New_D_i_v_2*E_i_e_2, ... ]
```

by substituting `New_D_i_v` for `D_i_v-Infty+1`.

The connectivity constraint itself is equivalent to the following conjunction:

```
D_i_c_i *== 0,
D_i_u * V_i_u *=< Infty-1,
...
D_i_w * V_i_w *=< Infty-1.
```

The lower the value of `Infty`, the tighter the constraints will be. However, `Infty` should be chosen carefully, because if `Infty` is too small, the optimum solution may occasionally be pruned. Stating other constraints is easy, e.g. $T_i \cap T_j = \varnothing$ is equivalent to `E_i_d *E_j_d +...+ E_i_f * E_j_f *== 0` etc..

5.3 Analysis, Experiments and Comments

It can be easily proven that conditions 1, 2 and 3 of the formal model for a solution are equivalent to the constraints shown in the preceding section. However, interval techniques are not sufficient and do not infer some logical consequences of these three conditions from the constraints. Therefore, some redundant constraints can be added, e.g., the constraint which asserts that every vertex in a tree should be incident to some edge:

```
V_i_v *== max [E_i_e_1, E_i_e_2, ... ].
```

Here the edges e_1, e_2, etc. are incident to the vertex v in the graph G, and `E_i_e_1`, `E_i_e_2`, etc. and `V_i_v` are their respective interval domain variables.

In dealing with problems of combinatorial optimization, the use of branch-and-bound mechanisms is almost universal. However, constraints can express some properties of optimum solutions; e.g. if some leaf v of the T_i is not in V_i, then T_i cannot be an optimum solution (because is is possible to remove v from T_i and get a cheaper tree). This consideration is implied by the following constraint:

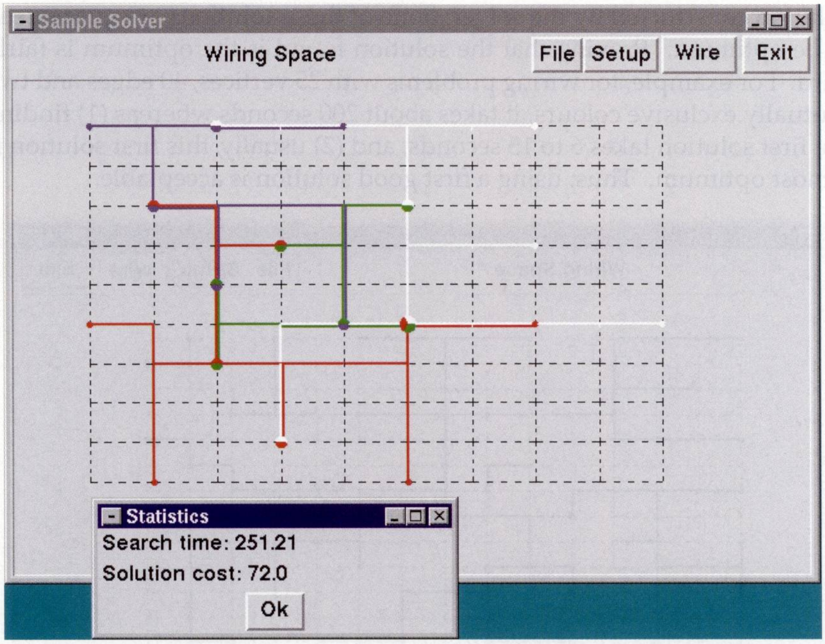
```
2 * V_i_v *=< E_i_e_1 + E_i_e_2 + ... ,
```

where the edges e_1, e_2, etc. are incident to the vertex $v \notin V_i$, and `E_i_e_1`, `E_i_e_2`, etc., and `V_i_v` are their respective interval domain variables.

The wiring problem can be reduced to a set of non-linear constraints which involve the min, max and arithmetic operations included in the ID library. The only further addition to the library was the `int` constraint which enforces an interval domain to have integer bounds. This constraint is applied to the variables `D_i_v`, `V_i_v` and `E_i_e` for each vertex v, edge e and colour i.

Wiring cables of the same colour is equivalent to the Steiner tree problem studied in graph theory [Christofides, 1975]. If two, or three, or all vertices of the graph are required, then the Steiner tree problem is provably solvable in polynomial time. However, in the general case, the problem is NP-complete.

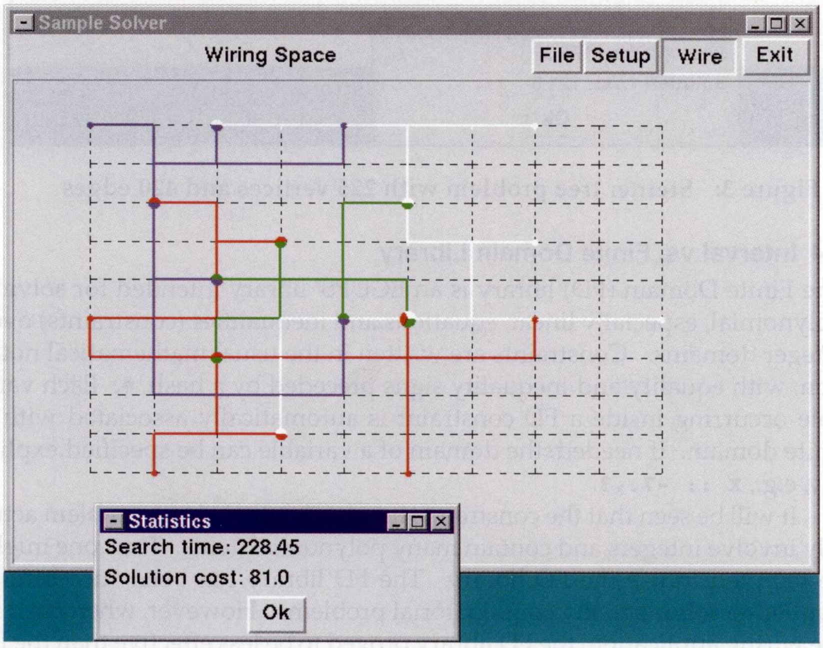
The Interval Domain Library has been tested on a series of problems with four colours and a wiring space having approximately 400 edges and 200 vertices. A typical problem involves about 800 domain variables and about 800 equations and inequalities. Figures 1, 2 and 3 show screen snapshots taken of the solver devised for the wiring application. The tests were run on a Pentium©-66 processor under Linux©. Each figure shows the *first*



Search time: 251.21

Solution cost: 72.0

Figure 1: Differently coloured cables are mutually exclusive

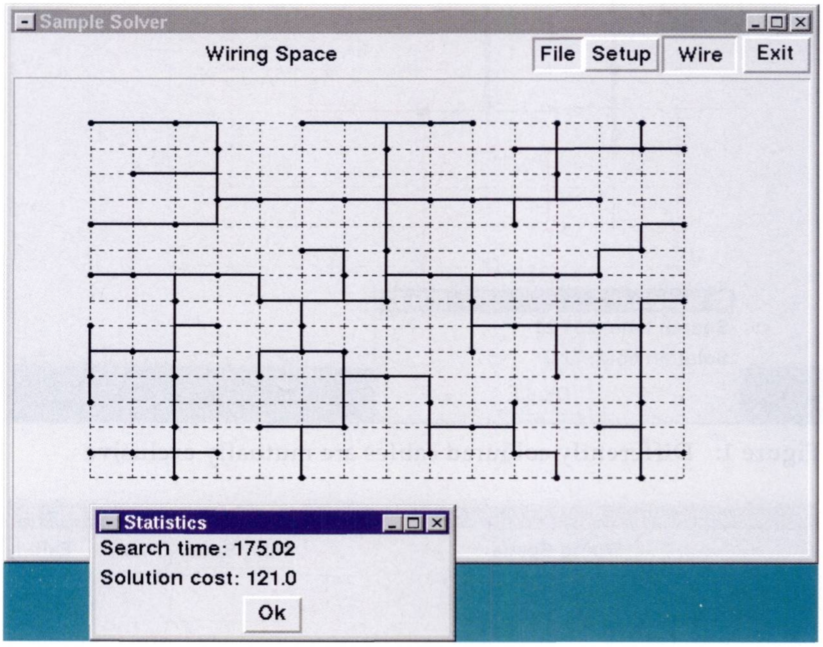


Search time: 228.45

Solution cost: 81.0

Figure 2: Similar to Figure 1, but any colour matches any other

solution constructed by the solver; none of these solutions are guaranteed to be optimum. Proving that the solution found is the optimum is fairly hard. For example, for wiring problems with 25 vertices, 40 edges and two mutually exclusive colours, it takes about 200 seconds whereas (1) finding the first solution takes 6 to 15 seconds, and (2) usually, this first solution is almost optimum. Thus, using a first good solution is acceptable.



Search time: 175.02

Solution cost: 121.0

Figure 3: Steiner tree problem with 225 vertices and 420 edges

5.4 Interval vs. Finite Domain Library

The Finite Domain (FD) library is an ECL^{i}PS^e library intended for solving polynomial, especially linear, equations and inequalities (constraints) over integer domains. Constraints are written in the usual mathematical notation, with equality and inequality signs preceded by a hash, `#`. Each variable occurring inside a FD constraint is automatically associated with a finite domain. If needed, the domain of a variable can be specified explicitly, e.g., `X :: -7..3`.

It will be seen that the constraints involved in the wiring problem actually involve integers and contain many polynomial items. Thus one might be tempted to use the FD library. The FD library has been successfully applied to solving many combinatorial problems. However, when used in the wiring application, the FD library proved to be less effective than the ID library. This is mainly because the evaluation of polynomial constraints is delayed by the FD library until they become linear but in the case of the ID library are processed from the very start of the solution process.

Both libraries were tested on the same set of examples. In the interests if fairness, the **min** and **max** constraints were changed so as to access FD variables as well. As more information is inferred from constraints, the non-determinacy is reduced. The number of non-deterministic choices made before reaching a solution was therefore taken as a measure of the ineffectiveness of each library.

Some results were quite unexpected. For example, in the version of the FD library, multiplication with negative numbers gave faulty results. For example,

```
[eclipse 1]: lib(fd), X :: -10 .. 0, Y :: -7 .. -5, X
* Y #= 20.
X=X{[-4..-2]}
Y=Y{[-7..-5]}
Delayed goals
   X{[-4..-2]} * Y{[-7..-5]}#=20
   X{[-4..-2]} *` Y{[-7..-5]}#=20
yes.
```

whereas

```
[eclipse 2]: X :: 0 .. 10, Y :: 5 .. 7, X * Y #= 20.
X=4
Y=5
yes.
```

and even worse

```
[eclipse 3]: X :: 0 .. 1, Y :: -5 .. -3, X * Y #= 0.
no (more) solution.
```

whereas

```
[eclipse 3]: X :: 0 .. 1, Y :: 3 .. 5, X * Y #= 0.
X=0
Y=Y{[3..5]}
yes.
```

However, processing negative integers is vital to the wiring application because **New_D_i_v= D_i_v–Infty+1** (recalling that **D_i_v** is the distance in T_i between its centre c_i and the vertex v) is usually negative. Fortunately, by declaring the connectivity constraint "as is", without **New_D_i_v**, causes no problem to the FD library.

The major drawback to using the FD library is the low sensitivity to the value of **Infty** and, as a consequence, to the connectivity constraint. Supposing that white, red and green cables were to be wired in pairs, then, with white and red cables are already laid, the wiring space remaining for the green cable usually is very constricted. Thus connectivity becomes the major requirement.

In wiring problems with 100 vertices, 10 of which are required, 180 edges, and one colour, values obtained for the ineffectiveness(FD)/ ineffectiveness(ID) ratio ranged from 1 to 1.5; with two colours, the range was from 1 to 5. With three colours, the very first test gave a value of ineffectiveness(FD)/ineffectiveness(ID) greater than 20 (with "greater than" meaning that with the help of the FD library no solution was found within sixty minutes on a Pentium-66 processor under Linux). However, with the ID library, the search times were within three minutes on the same platform.

Conclusions

The Interval Domain library is a new logic programming tool for solving non-linear constraints over real numbers. It accepts equations and inequalities written in normal mathematical form and, for each variable, it produces as output the residual continuous intervals over which the variables range.

The library falls naturally into a set of calculation functions and a set of Prolog predicates primarily needed to perform constraint propagation. Originally, the set of calculation functions was designed to handle incompletely specified numerical data in sub-definite models. Constraint propagation was chosen as the most adequate algorithm for sub-definite computations. In fact, sub-definite computations are an advanced variant of constraint propagation. To model constraint propagation in ECL^{i}PS^e is easy due to such built in mechanisms as goal suspension and meta-term manipulation.

The Interval Domain Library has been tested on many benchmarks for non-linear constraint solvers [Hillstrom et al., 1981], and at present the library is being used to develop a number of applications for resource planning and non-linear optimisation.

Each improvement to the library (or any other software) will influence either its effectiveness, or its ease of use. In terms of effectiveness of the ID library, it is obvious that the smaller the number of variables and operations in the equations to be solved, the tighter will be the output intervals obtained with constraint propagation. Currently a symbolic pre-processor is under design which will automate that elimination of "extra" variables, thereby increasing the effectiveness of the library.

Acknowledgements

The integration of the ID library into ECL^{i}PS^e has been completed at IC Parc (Imperial College, London) and ICL (Bracknell). Working there was really pleasant and fruitful and we extend our special thanks to W. O'Riordan, M.J. Rigg and M. Wallace for their help and kind support. We wish to thank O.V.D. Evans, in particular, for his great effort in correcting errors and improving the readability of the paper. We would like to thank also our home institutions, the Institute of Informatics Systems of the Russian Academy of Sciences and the Russian Research Institute of Artificial Intelligence.

Bibliography

BENHAMOU, F. and OLDER, W., "Applying Interval Arithmetic to Real, Integer and Boolean Constraints," Journal of Logic Programming, 1995.

CHRISTOFIDES, N., "Graph theory: an algorithmic approach," Academic Press Management Science, 1995.

ECRC, ECLⁱPS^e 3.5, ECRC Common Logic Programming System, ECRC, 1995.

MORE, J., GARBOW, B. and HILLSTROM, K., "Testing Unconstrained Optimization Software," ACM Transactions on Mathematical Software, Vol. 7, 1, pp 17—41, 1981.

KNUEPPEL, V., "PROFIL/BIAS – A Fast Interval Library," Computing, Vol. 53, 3–4, pp 323–335, 1994.

KUPER, G. and WALLACE, M., "Constraint Databases and Applications Lecture," Notes in Computer Science, Vol. 1034, Springer-Verlag, 1995.

MEZHOUD, A., DUFOURD, J.C. and DARBEL, N., "Performance-Driven Interconnection Optimization based on Constraint Programming," Proceedings of PACT'96, London, pp 179-191, 1996.

NARINYANI, A.S., "Subdefiniteness and Basic Means of Knowledge Representation," Computers and Artificial Intelligence, 2(5), 443–452, 1983.

OLDER, W. and VELLINO, "Extending Prolog with constraint arithmetics on real intervals," Proceedings of Canadian Conference on Computer and Electrical Engineering, Ottawa, Canada, 1990.

SEMENOV, A., BABICHEV, A., KASHEVAROVA, T. and LESCHENKO, A., "UniCalc, a Novel Approach to Solving Systems of Algebraic Equations," Proceedings of the International Conference on Numerical Analysis with Automatic Result Verification, Lafayette, La, USA, pp 29–47, Feb, 1993.

VAN HENTENRYCK, P., "Helios: A Modeling Language for Global Optimization," Proceedings of PACT'96, London, pp 317-335, 1996.

Biographies

Tatyana Yakhno

Tatyana Yakhno currently is the head of the Laboratory of Artificial Intelligence at the Institute of Informatics System (IIS) of the Russian Academy of Sciences and an associate professor of Novosibirsk State University. She received her PhD in Computer Science in 1987. Her research interests include knowledge base technology, constraint programming and distributed AI.

Slava Zilberfaine

Slava Zilberfaine graduated from Novosibirsk State University in 1996 with an MSc degree in Computer Science. His major scientific interests are computer graphics, logic programming and genetic algorithms. Currently he is studying for aPhD at IIS.

Evgueni Petrov

Evgueni Petrov is a PhD student currently enrolled at IIS as a member of the Laboratory of Artificial Intelligence. He graduated from Novosibirsk State University in 1996 with an MSc degree in Computer Science. The topic of his PhD work is combining methods of discrete mathematics with constraint logic programming.

ICL & University of Newcastle Conference on Teaching of Computer Science 1–5 Sept 1997

A Personal Review

Michael H. Kay, ICL Fellow

ICL, Bracknell, UK

This annual conference, sponsored by ICL and chaired by Brian Randall at the University of Newcastle, brings together by invitation leading academic computer scientists and industrialists to consider the developments in a field of computer science that is expected to have significant impact on the future teaching of the subject.

The theme of the 1997 conference was "The Web" and, as one might expect, it steered well clear of the mundane to focus on those current developments that have real strategic significance. In the sections below I report very much from a personal perspective what I learnt from each of the speakers.

These summaries are personal impressions and they have not been endorsed in any way by the speakers or by the conference organizers.

The Future of Telecommunications and Networking

Professor David Farber, University of Pennsylvania

Farber's first talk concentrated on the technical aspects of the future of networking, while his second talk concentrated on the political influences.

- Wide-area bandwidth will continue to grow inexorably, the question is how we take advantage of it.
- Distributed systems haven't really happened yet in any real sense. They will, but we still have software engineering challenges to build them.
- The increased bandwidth on the network will cause a change in processor architecture. We cannot afford to lose all the bandwidth by moving the messages through multiple layers of application software. Specialist video server machines are leading the way. The future is for the network to reach the processor chip directly, eventually with direct optical connection to the chip.
- We must stop treating computers and networks as separate subjects.

Rights and their management

Rober Weber, Senior Vice-President, InterTrust Technologies Corporation

- Weber's basic premise is that electronic commerce should mirror the trust and rights relationships that have evolved in commerce over the centuries.
- The aim of the InterTrust technology is to allow the application developer to create *any* rights structure, rather than imposing the supplier's own ethical judgements. Different societies have different views on issues such as censorship, freedom-of-speech, rights of employers to monitor communications, rights of governments to impose a tax on transactions, and so on. A technology supplier should support all these models (even if it means supporting some models that we consider repressive).
- InterTrust is a system for defining such rights and enforcing them. The idea is that in a value chain each participant can assert his own rights but cannot violate the rights claimed by others.
- InterTrust does not provide the administrative apparatus for collecting royalties, etc.: this has to come from a third party.
- There is potentially a problem that many information rights (e.g. fair dealing in copyright law) are very fuzzy in their interpretation. Codifying them more precisely is not necessarily in any party's interest.

Steganography

Ross Anderson, Cambridge University

- Steganography means "hidden writing", it is the technique of putting hidden information in an apparently clear message. It is an ancient science!
- There are applications in watermarking and fingerprinting. A watermark is like a hidden copyright notice: an early example being printed logarithm tables which contained the occasional deliberate error, to enable copying to be detected and proved. A fingerprint is like a hidden serial number, used to identify the individual copy so that the copyright violator can be traced.
- Like other security techniques, steganography can be used equally by good guys and bad guys. (And of course, we can have our own views as to who the good guys and bad guys are).
- There are many techniques that work very well if you aren't too worried about the hidden message being spotted, intercepted, or destroyed

(and thus, for the many situations where no one suspects you are using the technique). Secure techniques that will withstand attack by a "capable motivated opponent" are much more difficult and tend to be very low bandwidth.

- Naïve fingerprinting techniques can always be defeated by buying several legitimate copies and comparing them.
- Ross talked about "breaking" a steganographic message without always saying whether this involved detecting, reading, or destroying the hidden message. I would have liked to have seen this explained more clearly.
- The details vary by information type (image, audio, and text being the most common) but the underlying principles are similar. Examples:
  - use the least-significant bits of an image or audio stream
  - similar, but using a secure key to determine which bits are affected
  - exploit the sensory redundancy in the data stream; e.g. trailing spaces in text, high frequency sound, colour details in image
  - with text, exploit semantic redundancy; e.g. spelling variations or synonyms. Very hard to automate effectively.
- To create a fingerprint, e.g. a serial number on each copy of a software or information product, one desirable idea is to have a single ciphertext with a different key for each user, such that each user gets a different plaintext copy. There are algorithms that achieve this quite well, unless lots of legitimate users collaborate to produce a clean copy. (The history of satellite decoders shows that such collaboration is quite likely!)

The use of the web for access to information: The Stanford University Libraries program

Jim Coleman, Stanford University Library

This talk discussed the changing role of the university library (and the scholarly publisher) in the age of the web. There is a need to distinguish the traditional roles of the library and to ensure none of these is lost: repository, archive, place of study, hub of information access, broker of data and metadata resources, content creator/publisher. Stanford University Information Services (=Library + Computer Service) employs 500 people to serve 20,000 students and staff, and has a correspondingly large budget.

I felt the speaker was too complacent about the need for organizational change. The value chain (via publisher and librarian) is bound to change, and the cross-institution relationships will also need to change. Currently everything Stanford puts on the web is available to students at every other institution in the world, including small institutions that cannot match Stanford's resources but are competing for the same students and research grants. In my view, this traditional approach to collaboration in the library community will come under increasing strain as resources become accessible worldwide.

Having said that, the demonstration—a classic example of the World Wide Wait—proved merely that for a Newcastle University student to rely on a library in Stanford is not yet a practical proposition!

Document Formats for the Web

Vincent Quint, INRA/W3C

This talk was mainly an update on progress on HTML 4.0 and XML.

Quint also talked about MathML, an application of XML for defining mathematical formulae, either at layout level (subscripts, superscripts) or at semantic level (sum from 0 to n of ...). There was some scepticism in the audience: why hadn't they just put XML tags around a LaTeX mathematical formula?

The speaker was most enthusiastic about, but didn't spend much time on, DOM, the Document Object Model. This is an API allowing scripts to manipulate documents, including their logical structure, presentation, and content. It is designed to be platform and language independent and to work with HTML and XML. It is event based (e.g. allowing actions to be defined on mouse clicks) and related to the hierarchic structure of the document, so events not taken at one level are passed up to the next level.

URL's Considered Harmful (Information Management for the Web)

Wendy Hall, University of Southampton

This was mainly a talk about hypermedia: the speaker apologized for the catchy but misleading title.

Hall pointed out that there are two kinds of links in a typical web site, the structural links and the semantic links. Generating HTML from a database helps you to automate the structural linking but the semantic linking is still ad-hoc.

She reminded us of the original vision of hypermedia (e.g. Tom Nelson, Vannevar Bush, Berners-Lee) and pointed out how much of this was absent from today's Web:

- Bush: information pioneers or trailblazers find a way through the maze of information and leave a trail of links for others less expert to follow
- Nelson: the potential interconnectedness of everything

- Berners-Lee: the Web as an interactive medium; every browser is an editor.

The biggest difference in a true Open Hypermedia System is that the links are separate from the base content. This means you can view the same content via different sets of links. This allows great flexibility in customization, personalization, etc.

Hall talked about Southampton's experience with the Mountbatten Archive (250K documents, 50K photos, some film, video, and audio). Because of the archival nature of the material the links must be separate from the archived content. There can also be copyright considerations (apparently in the case of the Churchill papers, the UK government has purchased the physical materials but not the copyright!)

The Microcosm model, implemented in the Southampton system, is as follows:

- layer 1, tools such as WP, spreadsheet, query
- layer 2, presentation services
- layer 3, link services
- layer 4, document management
- layer 5, text / image / video / audio content.

The Southampton system implemented the link services in a Web proxy server.

Networking Futures - the politics

Professor David Farber, University of Pennsylvania

Prof. Farber talked specifically about US activities to take the internet forward. There are two separate activities, which are often confused but shouldn't be: Next Generation Internet, which is primarily government driven, and Internet II, which is an academic initiative. Both are about increasing bandwidth and reducing congestion. Individual communities (including the academic community) want to improve their own quality of service by sharing fewer resources with other communities.

There is some tension between doing networking research and just providing a better service. How to achieve Quality of Service without the enormous overheads of a fine-grained charging model is still a contentious issue.

Societal issues (freedom of information, governmental / international control, the cryptography debate) are slowing progress.

Distributed Middleware Tools

Rober Weber, Senior Vice-President, InterTrust Technologies Corporation

I felt this talk was a little too commercial for this event, with lots of claims ("We have invented a fundamental new form of security architecture") that were not substantiated—a dangerous tactic with a distinguished academic audience, even if the claims were true.

InterTrust use the concept of a DigiBox (which at first sight seems not too dissimilar from IBM's cryptolope) as a secure container for access rights; the DigiBox is programmable so you can define any rights model you like. It is separate from the (encrypted) content which might arrive later, e.g. you can get rights to watch a boxing match that hasn't been shown yet. The box can only be opened at an "Interrights Point" which is a small secure piece of software that enforces the rules.

Weber talked about the model of "superdistribution" of content: any purchaser becomes a potential reseller, with the rights owners collecting micro-payments from each reader.

Some of the academics found the whole thing culturally an anathema: their Utopia is a world where the taxpayer finances the creation of information and everyone then has free access to it. Weber had to take the defensive, arguing that his technology was neutral to the rights policy you chose to adopt.

Global Searching

Udi Manber, University of Arizona

Udi is the author of the Glimpse and Harvest products. He claims that all his co-workers have become millionaires but he preferred to stay in research!

There are two types of search site, spider-based search engines (AltaVista etc.) and manually created catalogues (Yahoo). Spiders can't cope any more, because the Web is too big. They now sample each site rather than looking at all its pages, and visit some sites more frequently than others. Also, an increasing amount of information is hidden behind registration forms or database queries making it inaccessible to the spiders.

The search sites are starting to see themselves as content venues financed by advertising. (An interesting side-effect is that they want you to be at the site a long time—improvements in search speed are not necessarily in their commercial interest!)

Web site creators are using crazy techniques to increase their rankings in search results (the cyberspace equivalent of calling your company "1A Taxis") but the search engines are fighting back. One vendor (OpenText) started accepting money from Web sites to increase their profile but this failed.

Manber has a new tool called the Search Broker which tries to mix manual categorization and search: you select a category first, then search.

Virtual Education

Peter Cochrane, BT Labs

A stimulating if chaotic talk, with little structure, but lots of interesting and challenging ideas about the future impact of technology on society in general and education in particular.

He got a pretty rough ride from the academics for using unfounded assertions and statistics (interjection from the floor: "90% of all statistics are made up on the spur of the moment") and for talking about exciting concepts like software breeding without saying in technical terms what he meant by them. Probably the real reason he aroused their wrath, though, was that if his predictions about the changes in society over the next twenty years are true, then universities as we know them are dead in the water.

Highwire Press, the Internet Imprint of Stanford University Libraries

Jim Coleman, Stanford University Library

Stanford has set up a publishing operation called Highwire. Its aim is to change the economics of the STM (Scientific, Technical and Medical) publishing process, and to create a more direct link between the writers and readers of scholarly journals. In practice, however, I find it hard to see how it differs from "yet another STM publisher".

It has 17 journals on-line at http://highwire.standford.edu, and claims these are very high-use journals, mainly published by professional societies. He says they have another 50 signed up. The service looks very similar to (say) the Academic Press IDEAL service designed and operated by ICL. They rely on the original publisher to do subscription management: they just produce the on-line version of the journal.

The academics present all seemed to share a universal dislike of the STM publishers, viewing them as entirely parasitical with no real added value. (Most of them haven't grasped the fact that the main added-value is in fact in deciding which papers to reject).

One thing they have been working hard on is citation-following. They are working on automatic linking protocols and industry standards to facilitate this. They offer as a special bonus "toll-free linking": if you subscribe to an article in journal A and it has a link to an article in journal B, you are allowed to follow that link even if you haven't paid for access to B. (I don't know if this applies recursively!)

Tools for Intercreativity on the Web

Vincent Quint, INRIA

The original concept of the Web was that every user should be able to read and write. It was never intended to create the strong division that now exists between publishers and consumers.

Quint talked about a project called Amaya that is trying to design a collaborative authoring environment: see http://www.w3.org/Amaya for details. Amaya is the W3C testbed client software. On the authoring side it is really just a syntax-directed editor. It doesn't attempt to solve the server-side issues in collaborative working, such as version control, checkin/checkout, change notification, etc. It just uses the standard (but not always implemented) PUT method in HTTP 1.1.

Future Web Search techniques

Udi Manber, University of Arizona

Manber described a number of interesting projects, mainly, but not exclusively, his own.

- A category-based searching tool. Basically he has (single-handedly and with no librarianship training) carved out about 400 "databases" on the Web (many of them true databases, some just collections of pages/sites) and provided a uniform keyword search interface to all of them. So you say something like "fly London Paris" and it searches its favourite database of airline flights for you.
- webGlimpse is a tool to search the neighbourhood of any page. When a page is displayed he adds a keyword search form at the bottom; if you enter a query on this form it searches all the pages reachable in less than *n* links for the keywords supplied.
- John Kleinberg and Cornell has prototyped a promising way of using the citation structure of the web to find key pages. For example, if you do a search for "relational database" on AltaVista you will get thousands of hits. If you analyse all these hit pages looking for inward and outward links, you will eventually determine some useful "hub" and "authority" pages. *Hubs* are pages that contain the greatest number of useful outward links, *authorities* are pages that everyone likes to link to. It seems that this technique might enable the search engine to identify a small number of really useful hit pages for some otherwise very difficult queries.

Dependable Web Services

Santosh Shrivastava, University of Newcastle

Most of what Shrivastava described was uncannily close to the techniques we use in the ICL COMMANDS product (replication., clustering, load sharing, and so on). He described it as a box of tricks rather than a coherent architecture, but said it was the best you can currently do. He thinks there is potential to make the DNS (Domain Name Server) much more intelligent (or perhaps to add another level of indirection?). He said he hadn't really got any answer to the problem of wide-area load balancing (nor have we). Arguably, since load sharing is only necessary when the workload is high, allocating work to servers at random is as good a technique as any.

For transactional services such as on-line banking Shrivastava sees the CORBA object transaction service as the way forward. This has a Java interface called JTS. He also believes distributed workflow technology is the right way to script the application logic.

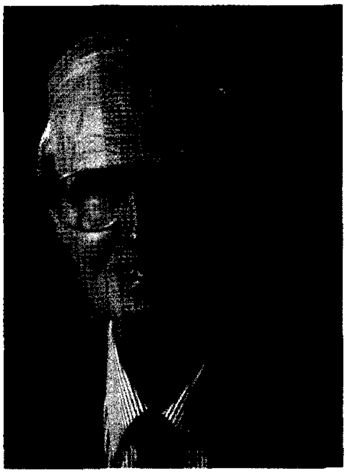
Safety and Privacy Issues for Clinical Information Systems

Ross Anderson, University of Cambridge

Anderson had been engaged as a consultant by the British Medical Association which was building a model to inform the debate on security and privacy in the health service.

This was a talk about the politics and ethics of security, not about the technology or the mathematics. The journalist who wants to find out whether a public figure is being treated for depression does not hack into the computer system, he phones the hospital receptionist, says he is a relative, and asks whether he can visit that afternoon. Anderson argued, therefore, that a great deal of the investment in security in the health service was based on an incorrect threat model. In particular, there was a tendency within the health service to centralize medical information and make it available to every health professional in the service (about a million of them), whereas the only possible way of retaining any patient confidentiality, in some cases, was to allow the GP to keep the record, on paper, in the bottom drawer of his desk. The clinical benefits of making information more widely available, he argued, had not been scientifically researched or proven.

Obituary



Dr John Maurice McLean Pinkerton, Chief Engineer of the project that built the world's first routinely working electronic business computer, LEO, died suddenly at his home on 22 December, 1997, in his 79th year. His career spanned the era of electronic computing from the very beginning to the present day. He joined J. Lyons & Co in 1949 to develop LEO and retired from ICL in 1984. He remained very active during his retirement pursuing a broad portfolio of activities, one of which was editing the ICL Technical Journal between 1990 and 1996.

Colleagues of John Pinkerton, who had known him at different periods during his career, offered the following contributions to this obituary, for which the Editor is extremely grateful.

Maurice Wilkes, FRS, writes:

I first met Pinkerton in 1939, when Watson-Watt and Cockcroft organized a scheme whereby some ninety physicists from various universities, but mostly from Cambridge, would spend a period of five weeks on a coastal radar station in order that they might acquire experience that would be useful if war came. Pinkerton was a member of a group led by Ratcliffe and attached to the CH station at Dover. I was in charge of a similar group attached to a station near Canterbury and my acquaintanceship with Pinkerton dated from this time. Pinkerton had just taken his BA degree and it seems likely that he had made arrangements to start research under Ratcliffe, but the outbreak of war meant that he went straight to a war appointment at TRE.

After war service, John Pinkerton returned to Cambridge and began research in the Cavendish Laboratory under J.A. Ratcliffe, who had been my own PhD supervisor when I was a research student, and it was during this time that I became well acquainted with him. Pinkerton's subject of research was the propagation of ultrasonic waves in liquids. Whether this topic was his own choice, or whether it was suggested to him by Ratcliffe, I do not know. At all events, it was of interest to me since I was then head of the mathematical laboratory, where we were engaged on the design and construction of an electronic computer based on a memory with mercury ultrasonic delay lines.

When Pinkerton had written his thesis and was about to look for a job, Lyons were ready to go ahead with their project to build a computer based on our work and were looking for a chief engineer. Accordingly, I pointed Pinkerton in their direction. They recognised his quality at once and he joined their staff. He was ideally qualified for the assignment and it was very largely due to him that the project was such an outstanding success. The Lyons team was also very strong on the application and business side but without Pinkerton's brilliant engineering skills this would have counted for little.

After the success of LEO I, Pinkerton went on to design other computers and, by the early 1960s, Leo Computers had the best offerings for business applications of any British computer company. Unfortunately, Lyons could not conveniently provide the continued injection of capital that was required and they arranged a forced marriage with the English Electric computer interests. English Electric was not only the dominant partner but its management style and background were as different as could be from those of Leo. The effect was to bring the development of the Leo range of computers to an end. If Leo Computers had remained an independent company with Pinkerton as chief engineer, it would, in my view, have had a much better chance of survival than any of its British competitors.

The merger with English Electric was followed by other mergers until the identity of Leo was completely lost in ICL. It is a sad fact that, although Pinkerton remained active in ICL at a senior level, he never found a role that in any way matched his track record or gave full scope for his abilities.

David Caminer (former Director of LEO Computers) writes:

John joined J. Lyons and Co. in January, 1949 to implement the decision of the company, then Britain's best known caterer, to build a computer of its own to cope with its mass of paperwork. There was no alternative supplier. Indeed, there was no other stored program electronic computer in operation anywhere at that time. John was recruited from Cambridge University where he was undertaking research after graduating in Natural Sciences and spending the war years on vital radar work at the Telecommunications Research Establishment, Swanage and then at Malvern.

John's first task was to create an engineered version of the scientific EDSAC computer that was being completed at Cambridge. This he accomplished with a very small team enabling the world's first regular, routine, time-critical business application to start its ongoing life in November 1951. This was followed at the end of 1953 by the full LEO 1, which with multiple inlets and outlets, concurrent computing and automatic conversion, provided the facilities required for full-scale, integrated office work. In February 1954, the first such application anywhere in the world, the Cadby Hall Bakeries payroll began its weekly operations.

John Pinkerton, still with his tiny team by present standards, went on to design LEO II, embodying the technological advances over time and the experience that had been gained of intensive live-running. In 1959 he was appointed a Director of LEO Computers Ltd, which had been formed by Lyons to market the system. Versions were installed on the premises of several leading organisations, among them the Ford motor company.

In his LEO years, John Pinkerton's culminating success was with the LEO III range that followed. The system that he designed, working closely as always with the 'users', the applications and programming team, incorporated both timesharing and microprogramming as well as being faster and smaller than previous models. One of the first to be sent out into the field went to goldmining interests in South Africa. This was in 1962, two years before the *announcement* of the IBM 360 series. John followed this with the LEO 326, much faster still, which, in competition with every system in the market, won from the Post Office the largest order ever placed with any European computer company up to that time.

Derek Hemy (a colleague at J. Lyons & Co.) writes:

I knew John Pinkerton during the period I worked with him at Lyons from 1949 to 1954. Thereafter, I met him only occasionally, although I did have a number of telephone conversations with him during the last few years.

I first met him in early 1949, when he had just come to Lyons. I found him very easy to talk to, though his manner struck me as quite distant. I soon realised that this was very understandable—the result of his starting to take charge of an engineering project that must have seemed pretty far fetched: to design and produce a working business computer in a firm with no background in electronic engineering. He had, moreover, to report to a Director of Lyons who, although enthusiastic, had had no engineering experience.

In fact his job was made more difficult by the fact that he was a newcomer in a company that prided itself on an intimate and family atmosphere and where the existing members of the LEO team were already established. However, he soon settled in without any particular difficulty, which I think was because we all found that he was a man with whom it was very difficult, if not impossible, to fall out.

He had a most disarming way of listening intently to what others said and was never afraid to admit ignorance or doubt. This was not due to any lack of assurance: he was able to make up his mind quickly and express his opinions very firmly but always quietly and reasonably. His leadership was not of the extrovert, demonstrative sort but he was not in the least afraid of 'calling the shots' and he was always unfailingly polite and shunned exaggeration.

In those early years of computing, a sense of humour was needed and in that John was not lacking, though his was a quiet dry humour. In particular, I remember a meeting chaired by John's director in Lyons to review progress and to try to solve some recent problems. The director ended the meeting by stating that he wanted to avoid unexpected problems in future and told John to give him a detailed note of them! After a pause, as we all tried to envisage such a note defining the unknown, John remarked, "Surely you are asking me to make a list of all the towns in China that I do not know;" a statement to which there was no possible reply.

John's engineering competence I can leave to others for comment, but to me his outstanding achievement was the way he handled the senior management. He usually got his way, not by any Machiavellian deviousness but by patience, good sense and an almost simple minded honesty.

I, certainly, am very grateful to have worked with John during those years.

Jack Howlett (founder Editor of the ICL Technical Journal) writes:

I had known John Pinkerton well by name in the early days of the digital computer, as one of the key people in the LEO project; it's possible that we met, neither knowing the other, at one or more of the meetings in the classic series organized by Maurice Wilkes in the Mathematical Laboratory at Cambridge in the early 1950s. What became a close association began in 1978, in connexion with the ICL Technical Journal. The company had agreed to a proposal to publish such a journal and Peter Hall, the Director responsible, had asked me to edit it and had strongly supported the idea of an independent Editorial Board with both ICL and non-ICL members. As a consequence of the series of mergers that had led to the creation of ICL in 1968, John was then an ICL employee (I should say here that I had a Consultancy attachment to ICL) and was a very obvious choice for membership of this Board: on the grounds of sheer scientific and intellectual strength, wide experience of the computer world and, by no means unimportant, experience with the production and publication of scientific literature gained from his involvement with the Institution of Electrical Engineers, of which he was a Fellow.

From the start John took a keen and active interest in the journal. He was always a very positive contributor to the discussions at the Board meetings, always with plenty of ideas on any of a great range of subjects and, invaluably, a fine, critical but constructive intelligence brought to bear on any paper or proposal. The journal gained a great deal from him; I found him an excellent colleague from whom I learnt a great deal, and we worked together increasingly closely in this relationship for a dozen years.

Around 1990 I began to feel that it was time for a change of Editorhip; John was the clearest choice, and accepted the offer. I wrote a valedictory

Editorial for my last issue in November 1989 and John what might be called an inaugural one for his first issue in May 1990. He asked me to continue not only as a Board member but also to look after the mechanics of the production and distribution of the Journal: each of the twice-yearly issues was of at least 200 pages, and we printed and distributed about 7000 copies of each, world-wide. I was very hapy to do this, and we continued this active and enjoyable collaboration until John in turn retired in 1996.

It scarcely needs to be said that John took the task of editing the journal with great seriousness, energy and enthusiasm, and spared no effort in ensuring that the papers for each issue—on subjects agreed, at a greater or lesser level of detail, at Board meetings—were produced on time and were of a standard that met his exacting standards for content, presentation and written English. He was very good indeed at discussing the content and form of a possible paper with a potential author and, with an inexperienced author, was especially good at helping to sort out the essential ideas and put them in the right logical order. His considerable scientific knowledge and wide experience were of great value here.

Altogether, I worked with John for about 20 years. He had as keen a critical intelligence as anyone I've ever met—you'd never get a phony argument past him—an enviable ability to handle detail and, as I've said several times in what I've written, was a true scientist. If ever I wanted to know something about physics or electronics I'd ask John if I could, knowing that I could rely on whatever he would tell me. But there were plenty of other sides to his character: he knew a lot about music, for example, and about English literature and had a lively appreciation of good food. It was my good fortune to have had this association with him.

John Aris (former Director of the National Computer Centre) writes:

I knew John Pinkerton for nearly forty years. When we first met he was already covered in glory (although he seemed quite unconscious of this) and I was the newest trainee. His friendliness and kindness to the likes of me, not in the least *de haut en bas,* were immediately striking—and of course typical.

But I would like to write about a much more recent period. John was one of the original Court members, from 1988, of the City of London's newly founded Worshipful Company of Information Technologists, and was the mainspring of their Apprenticeship Scheme. This is an imaginative adaptation of the mediaeval concept of apprenticeship to the modern world and to a modern craft. It is aimed at selecting, training and encouraging a group of able teenagers new to the world of work. Setting it up was far from easy. It needed creativity, energy, persistence, authority, ability to enlist and enthuse the talents of others, and meticulous care over detail—a rare combination—but an exact fit for John. Some twenty-five young people who have already benefited will remember him with gratitude and affection, and many more in the future will have cause to do so.

Anonymous Contributions:

Other former colleagues wrote to the Editor but, due to what they felt were the limited nature of their comments, they requested that their material be unattributed. Three additional aspects of John's career, not mentioned by the named contributors, follow.

- When English Electric Computers and LEO Computers merged both companies were heavily engaged in the development and introduction of important new computer systems. Their architectures, technology and forward strategy had nothing in common, a situation which could have led to prolonged and costly internal conflict. John Pinkerton played an important part in focusing the technical teams on shared objectives—perhaps thereby easing the future merger between English Electric Computers and ICT to form ICL.

- ECMA, the European Computer Manufacturers' Association, was formed to meet a need for rapid development of standards such as those for computer codes and languages. Serving for many years as its President, Pinkerton helped to build ECMA into an organization respected worldwide for the quality and timeliness of its work, most of which was subsequently endorsed by the necessarily slower processes of the official international standardization bodies.

- After his retirement from ICL in 1984. Pinkerton continued his active involvement in Information Technology. As well as working with BSI he made many contributions to the wider general understanding of the subject by lectures, articles and books, as well as making contributions to documenting the history of its development, as, for example, in the Science Museum's recorded interviews with UK pioneers.

Previous Issues

Design automation tools used in the development of the ICL Series 39 Level 30 system
Design and manufacture of the cabinet for the ICL Series 39 Level 30 system
Manufacturing the level 30 system I Mercury: an advanced production line
Manufacturing the Level 30 system II Merlin: an advanced printed circuit board manufacturing system
Manufacturing the Level 30 system III The test system

Vol. 4 Iss. 2 – November 1984

Modelling a multi-processor designed for telecommunication systems control
Tracking of LSI chips and printed circuit boards using the ICL Distributed Array Processor
Sorting on DAP
User functions for the generation and distribution of encipherment keys
Analysis of software failure data(1): adaptation of the Littlewood stochastic reliability growth model for coarse data
Towards a formal specification of the ICL Data Dictionary

Vol. 4 Iss. 1 – May 1984

The ICL University Research Council
The Atlas 10 computer
Towards better specifications
Solution of the global element equations on the ICL DAP
Quality model of system design and integration
Software cost models
Program history records: a system of software data collection and analysis

Vol. 3 Iss. 4 – November 1983

Expert system in heavy industry: an application of ICLX in a British Steel Corporation works
Dragon: the development of an expert sizing system
The logic language PROLOG-M in database technology and intelligent knowledge-based systems
QPROC: a natural language database enquiry system implemented in PROLOG
Modelling software support

Vol. 3 Iss. 3 – May 1983

IPA networking architecture
IPA data interchange and networking facilities
The IPA telecommunications function
IPA community management
MACROLAN: a high-performance network
Specification in CSP language of the ECMA-72 Class 4 transport protocol
Evolution of switched telecommunication networks
DAP in action

Vol. 3 Iss. 2 – November 1982

The advance of Information Technology
Computing for the needs of development in the smallholder sector
The PERQ workstation and the distributed computing environment
Some techniques for handling encipherment keys
The use of COBOL for scientific data processing
Recognition of hand-written characters using the DAP

ICL Systems Journal

Guidance for Authors

1. Content

The ICL Systems Journal has an international circulation. It publishes papers of a high standard that are related to ICL's business and is aimed at the general technical community and in particular at ICL's users, customers and staff. The Journal is intended for readers who have an interest in computing and its applications in general but who may not be informed on the topic covered by a particular paper. To be acceptable, papers on more specialised aspects of design or application must include some suitable introductory material or reference.

The Journal will not usually 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 material relating to unannounced products of any of the ICL Group of companies.

Letters to the Editor and book reviews may also be published.

2. Authors

Within the framework defined in paragraph 1, the Editor will be happy to consider a paper by any author or group of authors, whether or not employed by a company in the ICL Group. All papers will be judged on their merit, irrespective of origin.

3. Length

There is no fixed upper or lower limit, but a useful working range is 4,000-6,000 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.

4. Abstract

All papers should have an Abstract of approximately 200 words, suitable for the various abstracting journals to use without alteration.

5. Presentation

5.1 Printed (typed) copy

A typed copy of the manuscript, single sided on A4 paper with 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 Electronic version

Authors are requested to submit either a magnetic disk version of their copy in addition to the manuscript or an e-mail attached file or both. The format of the file should conform to the standards of any of the widely used word processing packages or be a simple text file.

5.3 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 be numbered for reference in the text and the text marked with the reference and an appropriate caption to show where each should be placed. Authors should check that all diagrams are actually referred to in the text and that copies of all diagrams referred to are supplied. If authors wish to submit drawings in an electronic form, then they should be separated from the main text and be in the form of EPS files. If an author wishes to use colour, then it is very helpful that a professional drawing package be used, such as Adobe Illustrator.

5.4 Tables

As with diagrams, these should all have captions and reference numbers. If they are to be provided in electronic form, then either a standard spreadsheet (Excel) should be used or the data supplied as a file of comma/tab separated variables. A printed version should also be supplied, showing all row and column headings, as well as the relevant units for all the quantities tabulated.

5.5 References

Authors are asked to use the Author/Date system, in which the author(s) and the date of the publication are given in the text, and all the references are listed in alphabetical order of author at the end. e.g. in the text: "...further details are given in [Henderson, 1986]" with the corresponding entry in the reference list:

HENDERSON, P., "Functional Programming Formal Specification and Rapid Prototyping," IEEE Trans. on Software Engineering SE 12, 2, 241-250, 1986.

Where there are more than two authors it is usual to give the text reference as "[X et al ...]". Authors should check that all text references are listed; 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, to conform to the Journal's general style for spelling, punctuation or notation, will be made by the Editor.

7. Proofs, Offprints

Printed proofs are sent to authors for correction before publication. The Editor will, however, always be prepared to send electronic versions to authors, either in PDF or as output files of the production system used for the Journal—PageMaker, Illustrator and Photoshop.

8. Copyright

Copyright of papers published in the ICL Systems 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.

ICL Systems Journal November 1997

ICL Research & Advanced Technology
Lovelace Road
Bracknell
Berkshire RG12 8SN