A Model-based Approach to Construction of Integrated Internet CSCW Systems

Michael Andrew Swaby

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy

The University of Leeds
School of Computer Studies

September 1998

Abstract

Internet technologies provide ubiquitous infrastructure for Computer Supported Cooperative Work (CSCW) applications, many of which share similar fundamental requirements for coordination, collaboration and information management services. However, there is a lack of structured architectural support for building and maintaining these systems. This thesis is directed towards an investigation of development mechanisms for integrated internet CSCW applications which promote reuse and rapid reconfiguration of CSCW services. A model-based approach to development of internet CSCW systems is proposed, based upon definition of reusable CSCW services and a specification language which describes user interaction with those services within an application context. At runtime, the specification is used to drive a Web user interface generator that dynamically integrates access to required CSCW services. This development approach enables many application changes to be affected quickly at the modelling level, rather than requiring code recompilation. Hence, investigation of the approach was directed towards rapid prototyping and evolutionary maintenance of internet-based CSCW systems. A proof-of-concept system architecture was implemented and applied to a case study cooperative working scenario within a large telecommunications enterprise. Assessment of the implementation found the approach to be useful in reducing iteration cycle times following change requests, thereby enhancing participatory design of CSCW systems. The value of the approach is in increasing communication and feedback between application builders and users by enabling rapid exploration of evolving system designs.

Acknowledgements

I would like to thank the members of the Centre for Virtual Working Environments (CVWE) at The University of Leeds, within which this research project was based. I would particularly like to thank Professor Peter Dew for his supervision of this project and Professor Christine Leigh for providing advice throughout. I am grateful to Dr. David Morris and Dr. Gyuri Lajos for their technical expertise and Dr. Richard Drew, Dr. Neil Hunter, Dr. Jason Wood, Steve Rowett, Dave Small and Judi Thursby for their help and encouragement.

This work has been funded through an EPSRC Industrial CASE award in conjunction with BT Laboratories. I would like to thank the staff of the Agent-enhanced Workflow Group and the Intelligent Systems Unit at BT Labs for their support during the project. In particular I would like to thank Dr. Paul Kearney, Paul O'Brien and Dr. Mark Wiegand for their supervision and assistance.

I have been fortunate to have been able to build upon existing work within the CVWE and at BT Laboratories. The proof-of-concept research implementation was constructed using existing software components where these were available and I would therefore like to acknowledge their authors: The first generation of the DiMe interpreter was created by Dave Morris. Gyuri Lajos enhanced the implementation greatly through the NEST Project. The scene-graph API within DiMe was written by Thorsten Blaise. Builtin HTML object support was added by Rik Wade. The VSP 3D component library was built and managed by Diane Willows. Greg Platt built the VSP UI metaphor set and VRML2 demonstration. Gareth Bottomley produced the prototype client-side Java DiMe implementation. The Process Interchange Format (PIF) model parser used within the workflow helper application was written by Simon Thompson of BT Laboratories.

Contents

List of Figures

Glossary

ADEPT:	Advanced Decision Environment for Process Tasks
AEW:	Agent Enhanced Workflow
ANSA:	Advanced Networked Systems Architecture
API:	Application Programming Interface
CGI:	Common Gateway Interface
CORBA:	Common Object Request Broker Architecture
COSS:	Common Object Services
CSCW:	Computer Supported Cooperative Work
CSS:	Cascading Style Sheets
DII:	Dynamic Interface Invocation
DMSL:	Display Metaphor Scripting Language
DOM:	Document Object Model
DSI:	Dynamic Skeleton Interface
DiMe:	Display Metaphor
E-R:	Entity Relationship
EAI:	External Application Interface
EI:	Enterprise Integration
HTML:	Hypertext Markup Language

HTTP: Hypertext Transfer Protocol

IBS: Intelligent Business Systems

IDE: Integrated Development Environment

IDL: Interface Definition Language

IIOP: Internet Inter-ORB Protocol

ITU: International Telecommunication Union

IVE: Industrial Virtual Enterprise

KIF: Knowledge Interchange Format

KQML: Knowledge Query and Manipulation Language

LDAP: Lightweight Directory Access Protocol

LDIF: LDAP Data Interchange Format

NII: National Information Infrastructure

NIII: National Industrial Information Infrastructure

ODBC: Open Database Connectivity

ODP: Open Distributed Processing

OMG: Object Management Group

OQL: Object Query Language

ORB: Object Request Broker

OSI: Open Systems Interconnection

PIF: Process Interchange Format

PSV: Partially Shared Views

QoS: Quality of Service

RAD: Rapid Application Development

RDF: Resource Description Framework

RMI: Remote Method Invocation

RPC: Remote Procedure Call

SDL: Service Definition Language

SLA: Service Level Agreement

SQL: Structured Query Language

STL: Standard Template Library

URL: Uniform Resource Locator

VRML: Virtual Reality Markup Language

VSP: University of Leeds Virtual Science Park

VWE: Virtual Working Environment

VWS: Virtual Working System

W3C: World Wide Web Consortium

WCCS: Work Coordination and Collaboration Systems

WFMC: Workflow Management Coalition

WFMS: Workflow Management System

XML: Extensible Markup Language

Chapter 1

Introduction

The field of Computer Supported Cooperative work (CSCW) is concerned with the broad and interdisciplinary study of computer-support for coordinated activities carried out by collaborating individuals [41]. Software systems used to support CSCW scenarios are often referred to collectively as *groupware*, defined by Ellis et al as "*computer-based systems that support groups of people engaged in a common task and that provide an interface to a shared environment*" [27]. In essence, these systems attempt to bring people together with information and technology to create an effective computer-mediated working environment (Figure 1.1).

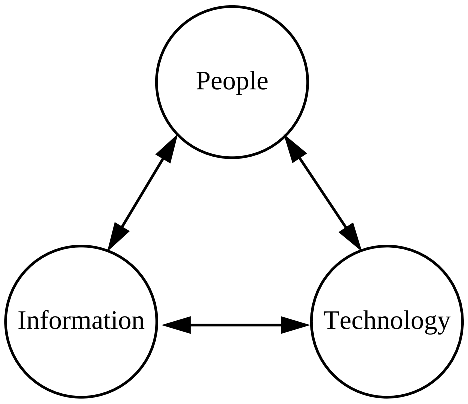


Figure 1.1: General aspects of computer-supported cooperative work

Far from being a distinct and isolated field, CSCW represents a confluence of interests that unites academics and practitioners from a wide spectrum of backgrounds e.g. distributed computing, social science, management, psychology, human computer interaction etc. Given this inherently multi-perspective nature, it is difficult (and probably inadvisable) to attempt to derive a precise universal definition of CSCW. This is reflected in the softness of most widely cited characterisations of CSCW. For example, in their early critique of the field, Bannon and Schmidt identified three general motivating requirements that CSCW as a discipline seeks to address rather than offering an all-encompassing definition [7];

Articulating co-operative work: coordinating people and resources in contributing towards the performance of a common task;

Sharing an information space: ensuring that group members can share data, information, concepts and heuristics in a structured way;

Adapting technology to the organisation (and *vice versa*): creating an organisational context within which group activities may be situated, and enabling appropriate interactions within this setting.

Investigation of these motivating requirements may be viewed in terms of the basic aspects of CSCW shown in Figure 1.1, in which three inter-related concerns are separated; the 'people' aspect is concerned with human interaction within cooperative work activities (encompassing notions both of human–system interaction and human–human interaction within the system); the 'information' aspect is concerned with the aquisition, representation and control of informational artefacts within the CSCW environment; and the 'system architecture' aspect is concerned with the provision of computing infrastructure through which the cooperative working requirements may effectively be delivered. The work reported within this thesis holds a *system-centric* perspective upon these elements of CSCW. From this perspective, the above motivating requirements may broadly be addressed through an amalgamation of several basic *classes* of system services, which together form the framework for the research (Figure 1.2, based upon [106]). From the systems perspective, CSCW solutions may be viewed as (often partial) integrations of three major component services; *coordination* services which direct people and resources towards achievement of a common goal, *collaboration* services which enable structured and *ad hoc* inter-personal communication, and *information management* services which provide a shared information context for cooperative work activities.

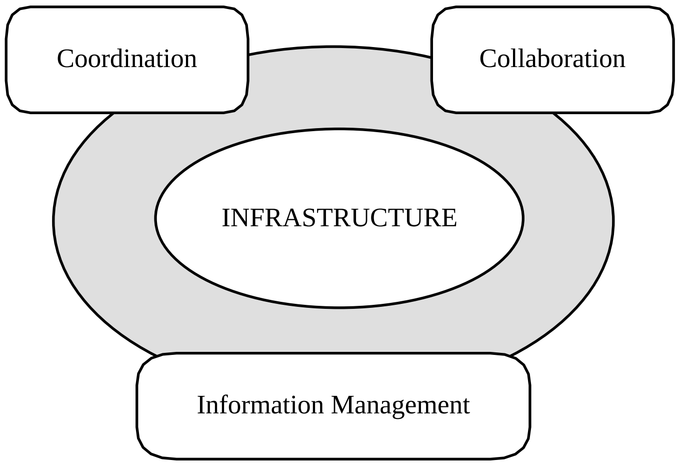


Figure 1.2: System-centric aspects of computer-supported cooperative work

Systems-centric CSCW research and development efforts may be characterised through comparison of the relative emphasis placed upon supporting these basic constituent services. The research reported in this thesis has been motivated through collaboration between the Centre for Virtual Working Environments (CVWE) at The University of Leeds and the Intelligent Business Systems Group (IBS) at BT Laboratories. The CVWE and IBS groups are focused on different research goals and application domains but there is a mutual interest in systems-support for coordination of distributed cooperative work activities. Within the general system-centric view of CSCW systems presented in Figure 1.2, research within the IBS group may informally be positioned towards the left side of the diagram, whereas the problem domain under investigation within the CVWE appears towards the right side of the diagram. There is however increasing research interest in holistic approaches towards supporting group work, in which traditionally disjoint process-oriented and *ad-hoc* collaboration-centric approaches are beginning to converge towards integrated work management solutions (e.g. [106]). This thesis attempts to contribute to understanding within this *middle-ground* area through an investigation of the contrasting, yet orthogonal, VWS and IBS approaches. These two approaches are described in Section 1.1 and Section 1.2 respectively. Section 1.3 then outlines the investigative domain for the research, leading to the identification of specific research hypotheses and objectives in Section 1.4.

1.1 Virtual Working Systems

The Centre for Virtual Working Environments at the University of Leeds brings together an inter-disciplinary team of social and computer scientists investigating a class of CSCW environment referred to as Virtual Working Environments (VWE). A Virtual Working Environment is characterised by the group as a working environment

- where people can undertake focused work;
- within a rich information space;
- based on familiar working metaphors;
- free from the need for physical co-location;
- delivered via internet technologies.

Virtual Working Systems (VWSs) are internet computing systems which create VWEs through the integration of a range of component services and technologies, e.g.

- information integration and knowledge services;
- search, navigation and reporting services;
- document management services;
- collaboration services;
- security services.

Implementations of these *core* VWS services are modularised and configured on a per application basis, using a layered software framework to guide development. Before describing this framework and the generalised VWS services provided within it, it is useful to consider a concrete exemplar of a VWS implementation—the University of Leeds Virtual Science Park.

1.1.1 The Virtual Science Park

The most mature VWS implementation to date is the University of Leeds *Virtual Science Park* (VSP) [24] which provides many of the services associated with physically located science parks through a VWS implementation. The major goal of the VSP is to enhance the University's ability to interact with industry and deliver online professional educational services. Tenants of the VSP are characterised according to the services they offer and the skills and expertise of their staff, providing a *yellow pages* service through which clients can quickly locate services and information they require. Once relevant services have been located, integrated collaborative tools enable communication with tenancy representatives. The VSP enables tenants to deliver a variety of services to their customers [24, 102] e.g.

- consultancy services;
- support for virtual project teams;
- work-based learning;
- research information services.

The VSP is implemented using internet technologies delivered through a World Wide Web interface, based on presentational metaphors for a physical science park e.g. a reception, visitor centre, tenancies, personal offices and conference rooms. Four major types of tenancies exist within the VSP;

Education and training tenants provide professional education-based services (e.g. work-based learning with mentoring);

Professional services tenants provide commercial or industrial services of use to other tenants and clients (e.g. legal services or patents information);

Collaborative project tenancies support collaboration between members of virtual teams through shared document repositories and integrated collaborative tools;

Innovative information brokers provide value-added services over particular information domains (e.g. competitive analysis reports for market sectors or classification and interpretation of research results).

The conceptual *map* of the VSP is shown in Figure 1.3. It should be noted, however, that although the VSP (and other VWS implementations) are delivered through physical metaphors they are not currently virtual reality applications. The technologies for delivering virtual reality VWS implementations are now filtering into the marketplace, but the benefits of adopting such mechanisms are not yet sufficiently understood to warrant detailed investigation. Instead of virtual reality implementations, the VSP and associated VWS implementations provide hybrid interfaces based upon HTML and 2D image-map representations of the physical organisational space. For example, Figure 1.4 shows a personal office area within a VSP tenancy.

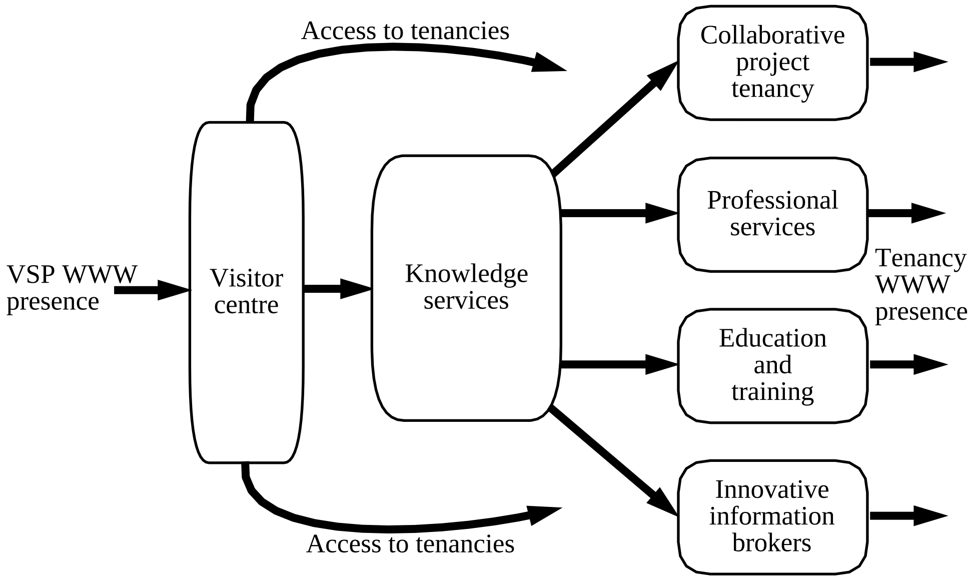
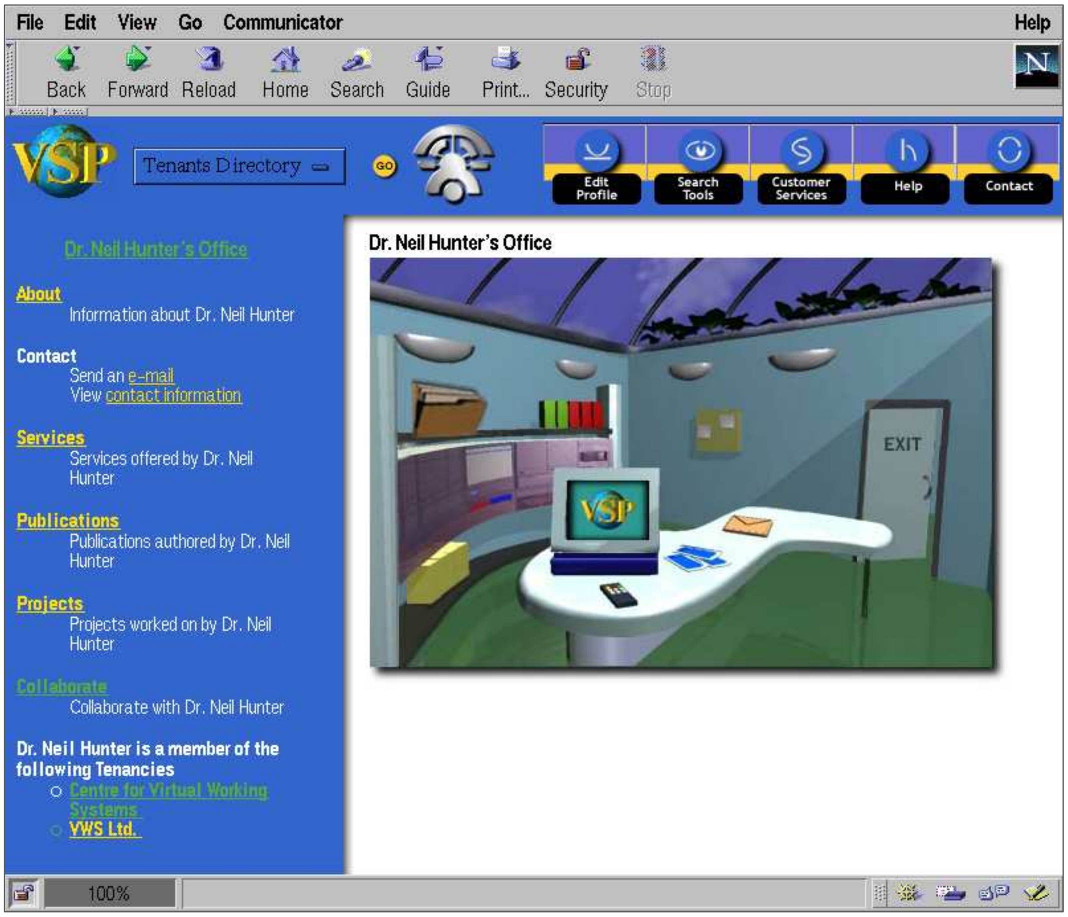


Figure 1.3: Tenancy navigation within the Virtual Science Park

1.1.2 The VWS framework and core services

The VWS software system, within which the VSP is implemented, integrates core VWS services through internet delivery mechanisms. However, VWS implementations are not specifically designed for the public Internet. The provision of private corporate network (intranet) VWS solutions is also of importance to the group.[1] Within an Internet context, VWS services may be regarded as value-added services that are layered above standard Internet and network services as shown in Figure 1.5.

[1]Within this thesis the term *internet* is used to mean either Internet or intranet. VWS solutions for these network domains would be differentiated primarily by their access control requirements. Such factors are inconsequential to this work and the existence of an appropriate security service is therefore assumed.



Dr. Neil Hunter's Office

About
Information about Dr. Neil Hunter

Contact
Send an e-mail
View contact information

Services
Services offered by Dr. Neil Hunter

Publications
Publications authored by Dr. Neil Hunter

Projects
Projects worked on by Dr. Neil Hunter

Collaborate
Collaborate with Dr. Neil Hunter

Dr. Neil Hunter is a member of the following Tenancies
- Centre for Virtual Working Systems
- VWS Ltd.

Dr. Neil Hunter's Office

Figure 1.4: VSP personal office (© 1998 University of Leeds)

A reference framework has been adopted by the group using an architectural style within which services are positioned into functional layers, as shown in Figure 1.6. The framework is largely based upon Wiederhold's I^3 (Intelligent Integration of Information) architecture [116]. The function of each layer is summarised below.

The **information integration** layer maps heterogeneous domain information sources into a consistent information space. For existing VWS implementations, the information space describes member organisations and the services they provide through the expertise and skills of their staff. This effectively provides an *organisational context* within which collaborative work is situated [6].

Information mediation layer components are responsible for providing semantic interpretation services over the information space. In the current VWS implementations, mediation is provided via multiple classification schemes that serve as simple domain ontologies. Future work within the group aims to incorporate intelligent software compo-

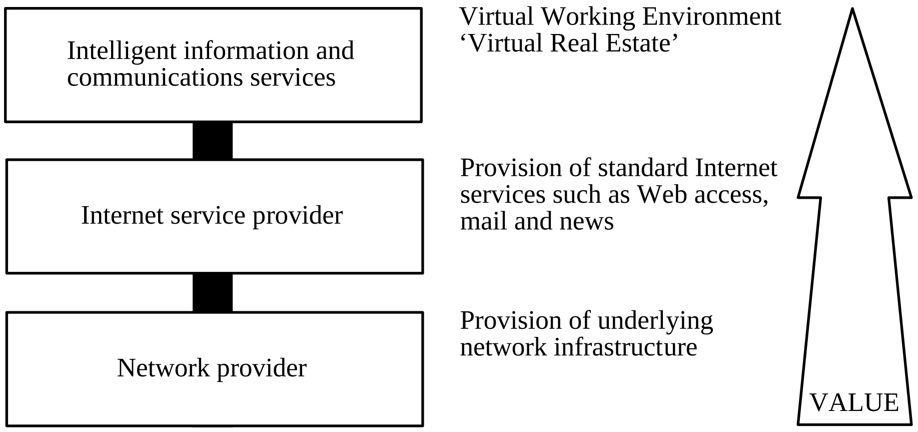

Intelligent information and communications services
Internet service provider
Network provider
Virtual Working Environment 'Virtual Real Estate'
Provision of standard Internet services such as Web access, mail and news
Provision of underlying network infrastructure
VALUE

Figure 1.5: VWS as value-added network services

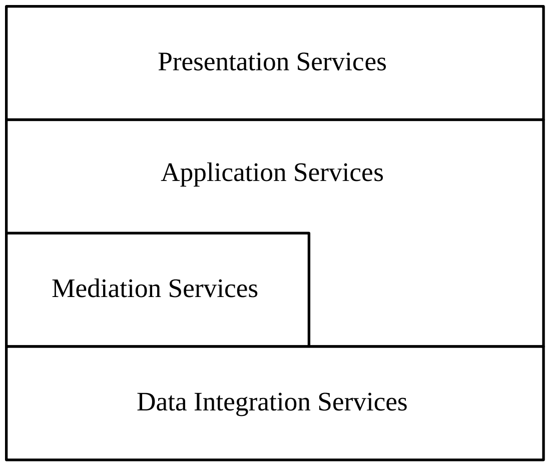


Figure 1.6: VWS layered framework

nents at this layer, aiming to provide problem solving ability (cf. Wiederhold's definition of mediation services in [117]).

The **Application services** layer covers a range of VWS services that make use of lower level information services, either via the mediation services or directly to the information integration layer. Current examples of application layer components are reading rooms, communications services, report generation and data mining.

Presentation layer services support user interaction with the VWS, via a variety of Internet browsers such as Microsoft Internet Explorer and Netscape Communicator.

1.2 BT Intelligent Business Systems

The Intelligent Business Systems (IBS) Group within the Applied Research and Technologies (ART) department at BT Laboratories is concerned with the application of distributed and intelligent systems to enterprise integration problems. To assist the reader, this section provides an overview of enterprise integration and related research efforts with a focus on approaches that are relevant to the IBS Group.[2]

Enterprise integration (EI) research is concerned with the coordination of enterprise resources such as people, machinery and information towards the performance of potentially large and complex business processes [95]. Large enterprises are usually structured into a hierarchy of semi-autonomous business units in which individual groups are responsible for management of their own local resources. Delivery of a product or service to customers requires the *coordination* [68] of these distributed resources through a business process as shown for example in Figure 1.7. In a large modern enterprise, resources contributing towards the business process are often distributed geographically as well as across business units, perhaps spanning several independent companies creating so-called *virtual* enterprises [37, 38].

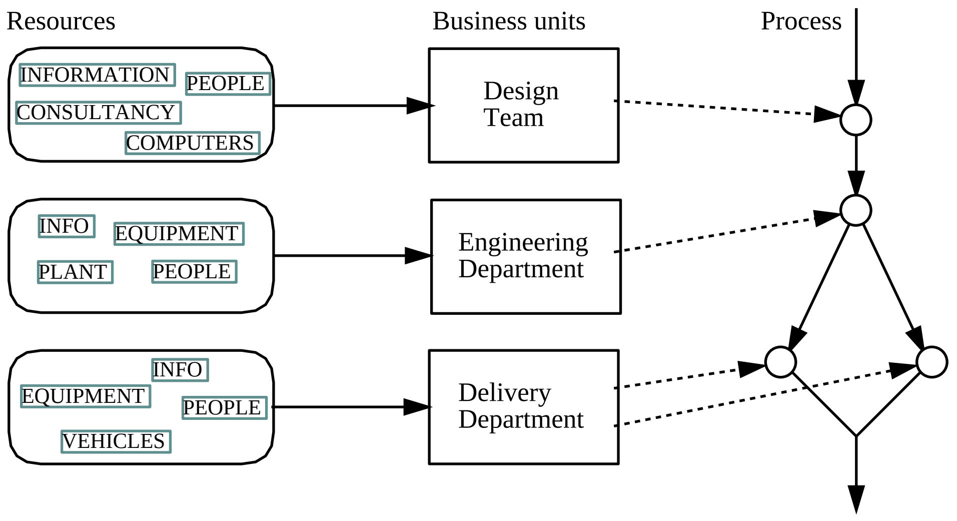


Figure 1.7: Business process spanning business units

Coordination of distributed resources to provide integrated business processes is a very

[2]The aim is to present the background for the PhD from BT's perspective as industrial collaborators, not to describe the activities of the IBS Group in particular; the issues discussed are in general common to all large enterprises.

complex problem for large enterprises. In order to overcome this inherent complexity, model-based enterprise integration approaches have emerged which attempt to raise the level of abstraction at which coordination occurs [95]. The model-based approach simplifies the enterprise to enable manual re-engineering but, importantly, also allows software systems to directly assist in the coordination process by operating upon these abstract models. For example, the Toronto Virtual Enterprise Project (ToVE) [9] uses symbolic AI techniques to model the operation of manufacturing enterprises supporting a degree of automated common-sense reasoning about the enterprise. In the AIAI Enterprise Project [112], an ontology-based toolset enables the creation of executable enterprise models which can, for example, be used to explore what-if scenarios as part of a re-engineering activity.

Research within the IBS Group at BT Laboratories is concerned with AI approaches to enterprise integration similar to those described above, with a particular focus on agent-based approaches to business process management. As a representative example of the approach taken within the group it is useful to consider ADEPT, a recent project under the DTI/EPSRC Intelligent Systems Integration Programme (ISIP) in which BT was a lead partner:

1.2.1 Advanced Decision Environment for Process Tasks

The ADEPT (Advanced Decision Environment for Process Tasks) Project applies intelligent autonomous software agents to business process management. Software agents [120] are distributed, intelligent, autonomous software components with the ability to interact and learn over time. Several typical characteristics of business processes within large enterprises can be modelled effectively through agent architectures [53, 30]. For example;

Distribution: enterprises are constructed of multiple groups which may be physically distributed;

Autonomy: groups have their own resources and are relatively free to manage their local activities;

Decentralisation: ownership of tasks, information and resources is decentralised;

Concurrency: many inter-related tasks are in progress at any given point in a business process;

Unpredictability: processes often cannot be completely specified *a priori* and may be affected whilst in progress through new instructions or error conditions.

Within ADEPT, a cooperative agent architecture was developed that enables business processes to be represented as a community of negotiating agents (Figure 1.8). A central concept in the ADEPT architecture is that of a *service*, representing some abstraction of problem solving endeavour [53]. Agents within ADEPT are responsible for negotiating for supply and consumption of services with other agents within the community. Agents may themselves be comprised of subsidiary agents under their control enabling hierarchical organisational structures to be modelled.

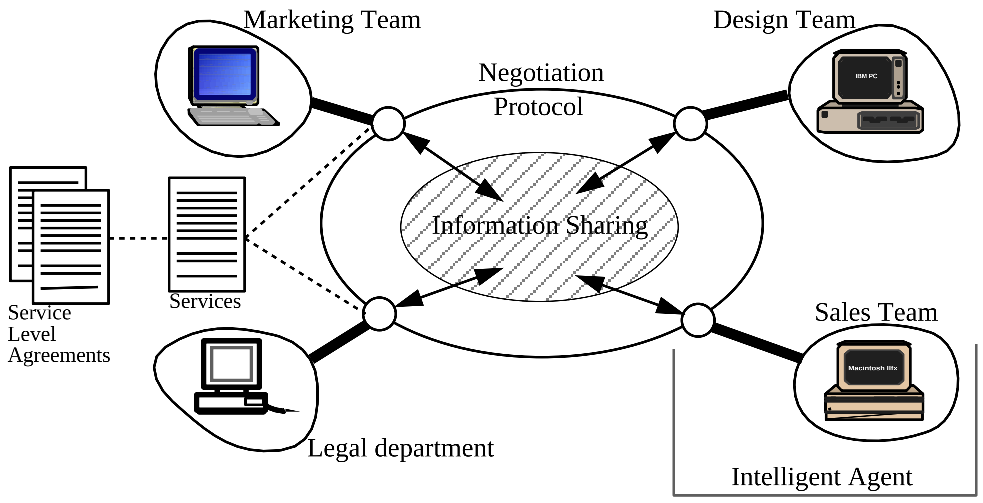


Figure 1.8: The business process as a community of negotiating agents

There are three major phases to the service lifecycle within ADEPT [53] (Figure 1.9). During the *creation* phase, services are modelled using a special purpose language called Service Description Language (SDL). SDL defines the inputs, outputs and components for a particular service using a declarative representation. Secondly, Service Level Agreement (SLA) templates are created that define the parameters over which agents may negotiate service delivery. At the second phase, agents negotiate for delivery of instances of services in a process referred to as *provisioning*. The final phase in the service lifecycle is the *management* of service delivery and dynamic re-negotiation of SLAs should this be necessary.

Agents within ADEPT are general purpose in the sense that they are not explicitly designed to manage specific services. Instead, the same basic agent shell may be loaded

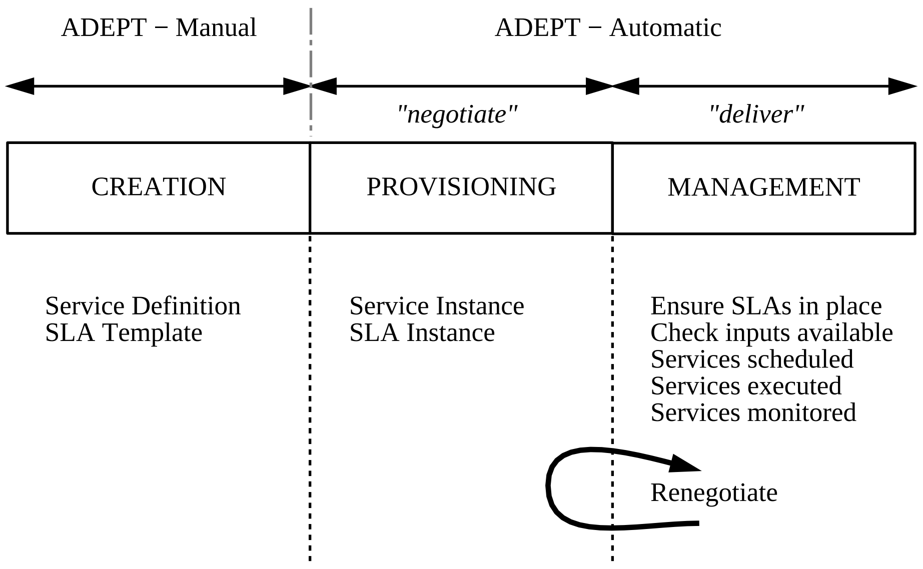


Figure 1.9: Service lifecycle in ADEPT

with relevant service definitions and information models which define how that agent should behave in a particular business process context. This approach enables component re-use across applications and dynamic modification of process characteristics e.g. task constraints or information requirements. The agent-based approach to business process management taken in ADEPT attempts to improve on traditional workflow management techniques in scenarios which require dynamic resource re-configuration or exception handling. However, the ADEPT agent architecture is not necessarily a replacement for conventional WFMS technology. Rather, it can be viewed as a layer above the WFMS which adds value by continually seeking to optimise the underlying workflow environment as it changes over time. In this role, ADEPT provides a degree of system support for *dynamic change management*. If a situation arises which ADEPT agents cannot resolve (e.g. as indicated by non-deterministic service negotiation) then a human manager can be notified in order to make a manual decision.

1.3 Research domain

A common objective of the VWS and IBS research groups is to develop systems which can meet evolving user requirements within dynamic cooperative environments. For example, the VWS group have requirements to create new Virtual Working Systems from existing core services; the IBS Group aim to create agent-based workflow systems that can cope with dynamic changes in business process characteristics. Within these environments, the basic requirement is to design and build software systems in such a way that the cost of iterative development and maintenance is minimised as systems evolve. This requirement might be stated informally as the need to "*engineer for change*."

As discussed earlier, VWS solutions support *ad hoc* human to human collaboration within managed information spaces. This loose coordination style contrasts with the process-centric IBS approach to work management in which software agents dynamically control evolving business activities. One area of mutual research interest between the VWS and IBS groups is the intersection of these people-centric and process-centric approaches, towards deriving systems that provide flexible collaborative support for people engaged in business processes within dynamic enterprise environments.

Virtual Working Systems have been developed primarily to support delivery of high-quality educational and research knowledge services. The broad goal of enterprise integration efforts (under which IBS may be classified) is to *bring together* enterprise resources (such as people, information and computing technologies) in an effective manner to create flexible work processes that evolve as the enterprise environment changes. The information and collaboration services provided by Virtual Working Systems could be of wider benefit within dynamic enterprises if they could be appropriately packaged and redeployed within their systems environments. The collaboration between the IBS group at BT and VWS group at Leeds provided a good basis upon which to explore this basic idea.

A further motivating factor which helped scope the problem was a desire to exploit the increasing potential of Internet computing and the World Wide Web in supporting co-operative work within (and between) integrated enterprises. Initial implementations of Virtual Working Systems were built through compiled languages on Unix workstations. Whilst such an approach proved adequate in demonstrating VWS concepts, it was only after switching development effort towards Web-based systems that commercial exploitation became feasible. It therefore seemed appropriate to investigate the application of these services within a wider context, given e.g. the ubiquity of the Web computing platform on a global scale; the increasing internal use of Internet technologies within large enterprises; the emergence of Web-based collaboration and coordination products.

1.4 Research problem

The *target users* for this research are software engineers responsible for building and maintaining Internet-based cooperative systems within dynamic application environments. When creating systems, engineers must bring together required component CSCW services and integrate these into a coherent application delivered via Web infrastructure. Such systems are referred to in this thesis as *integrated internet CSCW systems.* Software engineering methods based upon iterative system construction, evaluation and refinement are very useful in creating usable systems through participatory design [61, 4]. However, development of cooperative Web information systems through rapid iterative cycles is currently difficult, because of their complexity. The research reported within this thesis seeks to address three specific problems which currently prevent rapid prototyping and evolutionary development of integrated internet CSCW systems;

1. **CSCW services are not reusable**
   Many cooperative work scenarios share similar requirements for basic CSCW services e.g. coordination, collaboration, information management etc. yet significant duplication of functionality can often be observed across CSCW applications used within enterprises, increasing maintenance complexity and potentially introducing data consistency and synchronisation problems [6, 45]. Component reusability is a key rapid prototyping enabler [57], but no such component architectures exist for Web-based CSCW systems.

2. **User perspectives are not adequately reflected during prototyping**
   Cooperative working scenarios often involve participants who possess different perspectives upon their common activity [40]. Participatory design of cooperative systems through prototypes should reflect this heterogeneity but, because of the inherent complexity of the systems, it is difficult to quickly produce new iterations that adequately embody emerging requirements from the user community.

3. **Evolving requirements are not adequately reflected in evolving live systems**
   User requirements change over time of course, and these changes must be reflected back into an evolving system. However, due to the complexity of internet CSCW system implementations, changes typically cannot be affected as often, or at the

fine level of granularity that users desire because the required software engineering effort outweighs the potential benefit of the change.

1.5 Contribution

The contribution of this work towards addressing the above research problems is to investigate *model-based* architectural support for prototypical construction and subsequent maintenance of integrated cooperative systems. The approach proposed within this thesis may briefly be summarised as follows. Firstly, application-specific code, reusable CSCW services and user interface functionality are architecturally separated. Secondly, applications are specified at a high level of abstraction using a modelling language to describe user interaction with applications and services via the user interface. Finally, a runtime support architecture driven by this model is provided through which access to application-specific code and reusable CSCW services are provided via dynamically generated Web user interfaces.

The feasibility of the approach was assessed through a proof-of-concept implementation of a model-based CSCW toolkit called ParaDiMe, through which integrated internet CSCW systems may be constructed. The design of ParaDime (described in Chapter 3) extends the DiMe architecture [79] developed within the Virtual Working Environments Group at Leeds University. DiMe was initially developed as an automated user interface generation tool for the World Wide Web. ParaDiMe extends DiMe through support for a generalised distributed object model, which was provided via CORBA in the proof-of-concept implementation. This fundamental extension to DiMe enables ParaDiMe to provide interactive access to cooperative applications and the basic services from which they are composed. Several architectural components of ParaDime were constructed through this mechanism, summarised specifically as follows. A *remote method invocation* (RMI) service provides access to arbitrary distributed objects via static or dynamic interfaces. An *information manager* component provides a standards-based architectural interface to distributed information sources. A *forms processing* subsystem provides mechanisms for dynamic generation of HTML forms interfaces and associated handler code within distributed applications. Finally, a *collaborative tools* subsystem supports execution and session control of synchronous groupware applications. These component subsystems were integrated into a prototype application built using the ParaDiMe architecture. The *workflow helper* prototype (described in Chapter 4) was built in order to investigate the practical application of the proposed model-based approach to a case study cooperative working scenario.

The broad hypothesis explored within this thesis is that the proposed model-based approach can offer a solution to the research problems identified previously. Hence, three specific research objectives may be identified;

1. To investigate how common requirements in cooperative working scenarios are met by reusable CSCW services and how such services can be brought to together in a structured manner which promotes their integration and reuse within a model-based architecture;
2. To assess the benefit of a model-based approach towards system development through rapid prototyping, to test the hypothesis that the approach reduces prototype development cycle times thereby enabling a higher level of user participation in the design process.
3. To assess the benefit of a model-based approach towards maintenance of systems as user requirements evolve within live applications, to test the hypothesis that the approach reduces the software effort required to affect changes thereby enabling evolving user requirements to be more efficiently fed back into systems.

1.6 Thesis structure

Chapter 2 (Architectural support for CSCW) investigates CSCW services and their integration and reuse within a flexible CSCW system architecture. A number of existing CSCW frameworks are introduced, leading to the identification of a number of core CSCW service classes of central interest to this research. Relevant research and development efforts are then surveyed within this classification and a simple CSCW framework is derived which serves to guide system design.

Chapter 3 (Model-based CSCW architecture) specifies the detailed design for a model-based internet CSCW system architecture called ParaDiMe, which attempts to offer a solution to the research problems.

Chapter 4 (Case study implementation) describes a proof-of-concept implementation used to explore the research approach, through application towards an exemplar cooperative working scenario within a large telecommunications enterprise.

Chapter 5 (Critique) presents a critique of the research hypotheses through assessment of the architecture and proof-of-concept implementation.

Chapter 6 (Conclusions and future work) reviews the contribution of the thesis and offers an outlook upon directions in which the research might be progressed.

Chapter 2

Architectural support for CSCW

2.1 Introduction

The broad goal in this work is to investigate structured construction techniques for integrated internet CSCW systems. That is, CSCW systems that integrate a number of component services and are delivered via internet infrastructure. The motivation for *integration* of services, rather than re-implementation, is in simplifying systems development and reusing existing system components rather than duplicating software engineering efforts. Chapter 1 provided a high-level systems perspective upon cooperative work. Extending this simple model to explicitly include user interaction services provides a basic classification through which related work may be discussed (Figure 2.1).

Before considering these individual classes of CSCW services, however, it is useful to introduce existing CSCW *frameworks* which attempt to bring services together in a structured way. Section 2.2 discusses several existing and emerging CSCW frameworks that are of relevance to this work. Section 2.3 then discusses CSCW services according to the five basic CSCW architecture elements shown in Figure 2.1 (coordination, collaboration, information management, user interface and infrastructure).

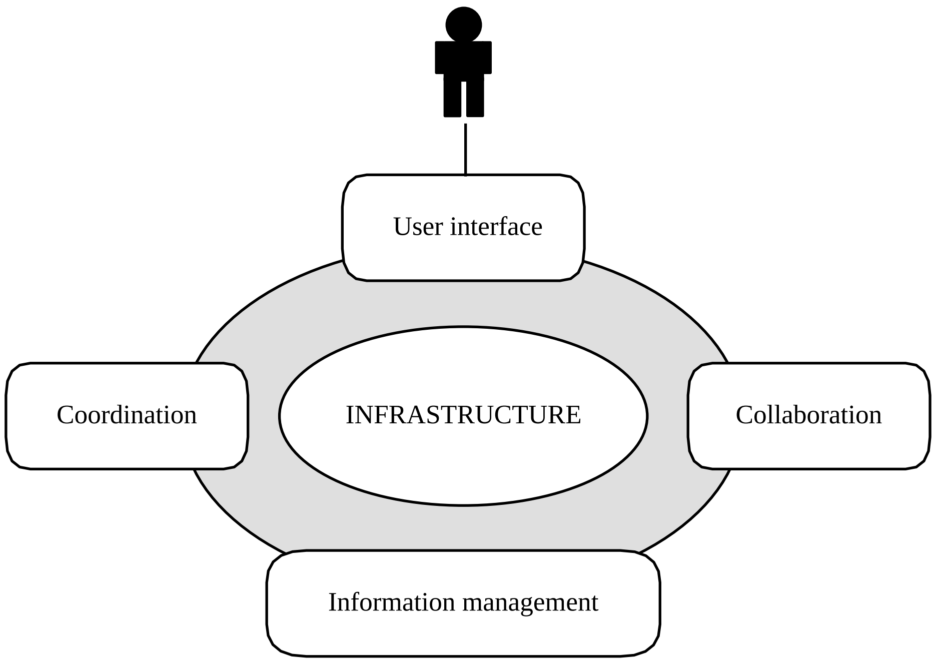


Figure 2.1: User interaction with integrated CSCW services

2.2 CSCW Frameworks

There is a significant duplication of requirements in many cooperative work scenarios, implying duplication in systems services used to support them [45]. Thus, there is increasing research interest in CSCW **frameworks** that attempt to modularise CSCW services and position them within a defined architectural structure e.g. [58, 92, 85]. A CSCW framework, in conjunction with a set of re-usable services, should be able to generate implementations to support a range of different cooperative working scenarios. In order to promote service re-use, CSCW frameworks often separate out application-specific from generic components (Figure 2.2). In [85], Navarro et al note that such a component-oriented approach to CSCW systems development has been taken with several existing projects, although the majority of early work was applied to real-time collaboration scenarios e.g. Rendezvous [93] and MMConf [21].

The work of Navarro et al differs from Rendezvous and MMConf in that it is directed towards a more general class of cooperative applications. They also sought to consider the implications of implementing cooperative systems within standard distributed systems architectures such as ANSA, OSI and ODP environments. Major workflow research groups, such as the LSDIS Laboratory at The University of Georgia, are moving towards

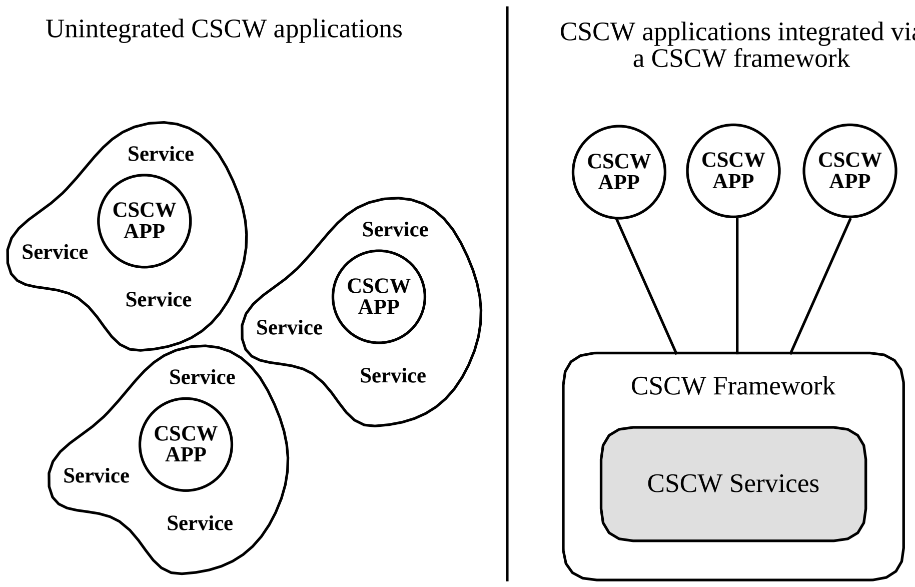


Figure 2.2: Framework-based integration of CSCW applications

more general and integrated cooperative working frameworks. For example, Sheth et al describe their vision of the next generation of workflow as systems as Work Coordination and Collaboration Systems (WCCS) [106], within which coordination, collaboration and information management services are seamlessly integrated. Because of it's generality, the WCCS model has been adopted as a classification framework within this work, as shown in Figure 2.1. The envisaged Work Coordination and Collaboration Systems are highly adaptive, reacting dynamically to changes within the organisation and business process. They also "*support a unified framework for managing coordination, collaboration and information-based decision making activities that naturally occur as part of organisational processes.*" The LSDIS group have a strong research background in innovative approaches to workflow and their emerging WCCS approach is highly relevant to this work. This research is, however, focused upon a small and specific subset of the work addressed within the WCCS research—synthesis of CSCW components within internet-based collaborative working scenarios.

An example of a generic abstract CSCW framework is the Co-Tech architecture [58], shown in Figure 2.3. The architecture specifies four major layers, within which collaboration services are positioned; the HCI layer provides common user interfaces to underlying applications, which are controlled through some form of collaboration management service. These higher level services are implemented through a support layer which provides standard distributed systems facilities.

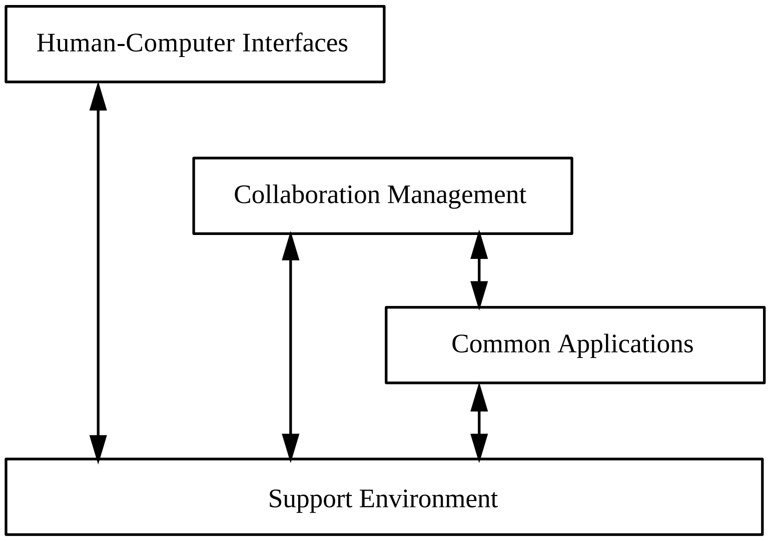


Figure 2.3: Co-Tech CSCW Architecture

Co-Tech is an example of a general purpose CSCW framework which may yield a wide variety of collaborative system implementations. A more detailed CSCW framework which seeks to address system integration issues is the NCR Cooperation architecture, described in [52] (Figure 2.4). The Cooperation architecture is based on NCR's Open Computing Architecture (OCCA) and implemented using distributed object technology.

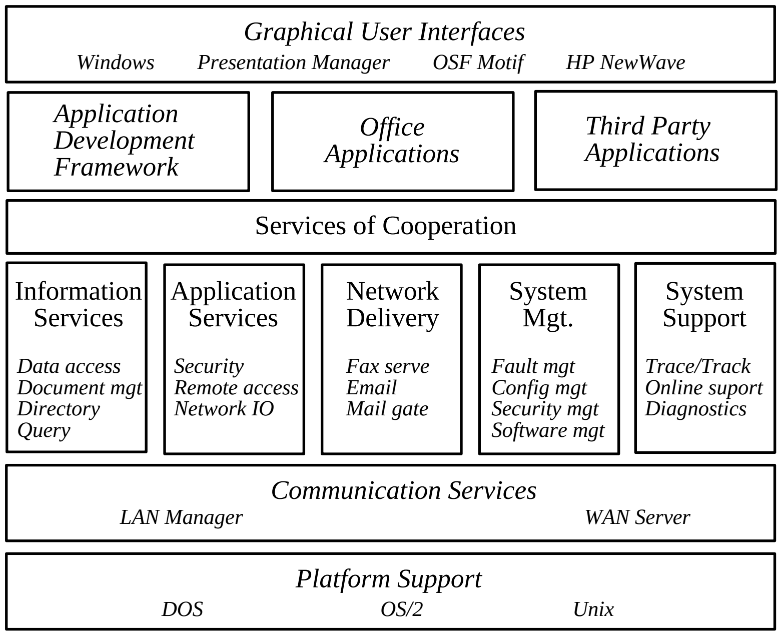


Figure 2.4: NCR Cooperation Architecture

Within the Cooperation architecture, collaboration and coordination services are integrated at the desktop via the HP NewWave interface. Users have limited ability to customise the look-and-feel of their interface onto the collaborative environment e.g. changing screen colours and icon layout with the NewWave GUI. The feature of Cooperation that is of most relevance to this work is, however, the open support for integration and construction of new applications. At the application layer of the architecture, the Cooperation Framework for Application Development provides a standard library of functions which developers can use to access a wide-range of lower level cooperation services e.g. information or applications as specified in the Cooperation Services layer in Figure 2.4.

Internet infrastructure initiatives such as the NII seek to specify the next generation *superhighway* supporting the broad spectrum of commercial, social, and leisure services. Although the general NII framework [84] is important in positioning VWS and related Internet-based collaborative systems in a wider context, its coverage is too broad to be directly of use within this work. However, some industry specific NII initiatives, such as the National Industrial Information Infrastructure (NIII), have been developed in depth and are therefore of immediate relevance. A major milestone in the development of the NIII has been the definition of a reference architecture (NIII-RA), defining the technologies that will be utilised in enabling Internet industrial virtual enterprises within the overall NII effort [80] (Figure 2.5).

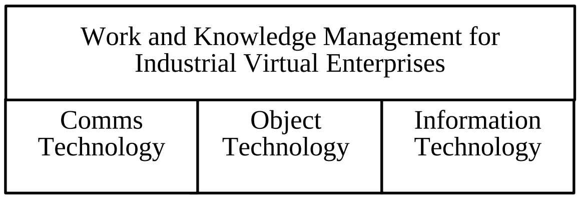

Work and Knowledge Management for Industrial Virtual Enterprises
Comms Technology
Object Technology
Information Technology

Figure 2.5: NIIIP Reference Architecture

The NIII Protocols Consortium (NIIIPC) identify four key technology requirements for industrial virtual enterprises (IVEs) that the NIII architecture seeks to provide:

- common communications protocols
- a uniform object technology base for system and application interoperability
- common information model specification and exchange
- cooperative management of integrated virtual enterprise processes

The strategy adopted by the NIIIPC in the integration of the core technologies is to focus on the user perspective of a work and knowledge management system for virtual enterprises. That is, technologies are positioned with respect to their roles in supporting particular user-driven requirements. A central component of the NIII architecture is therefore the notion of an enterprise model as a reference mechanism for the technology components that support the virtual enterprise.

The NIII-RA is notable with respect to this research in that it is based heavily on the integration and interoperability of component standards. Within industrial application domains (e.g. manufacturing) there are a large number of existing standards and frameworks. Rather than re-invent well-developed, and therefore well-understood architectures, the NIII effort seeks to enable integration of existing components where possible. For example, Figure 2.5 shows the NIII-RA as four broad services; work and knowledge management services operating over communications, information and object technology infrastructure. In specifying each of these service classes, standards-based solutions are proposed e.g. WFMC compliant work management, OMG compliant object services, STEP information management and Internet communications infrastructure. Re-use and integration of standard components is a major objective in this research and the NIII-RA is therefore of obvious relevance. The initiative has also produced results which are of relevance to practical issues of system integration within this work e.g. the task and session object model [81] provides a specification for implementing NIII services in an OMG-compliant environment.

2.3 CSCW Services

2.3.1 Coordination Services

In many cooperative working scenarios there is a general requirement for coordination services which are broadly responsible for "*managing the interdependencies between activities*" [68].

Jayachandra identifies five key properties that must be analysed in order to achieve process coordination [52]

Objectives: what is the purpose of the activity to be performed?

Activities: how can this purpose be expressed as an ordered sequence or functionally decomposed set of subtasks?

Performers: what skills/expertise are required to perform the above subtasks and how should these be mapped?

Interdependencies: what constraints exist between activities that comprise the business process?

Resource allocation: what resources are required at each stage of the process?

Workflow management solutions provide automated assistance for specification of business processes according to the above general criteria. These specifications are then used to drive workflow engines which enable multiple users to collectively enact the business process. Several hundred commercial products claim to support some kind of workflow management functionality [106]. Whilst it is beyond the scope of this thesis to compare the entire range of existing systems,[1] most implementations may be classified according to their level of support for three basic workflow types [105];

administrative workflows involve coordination of simple, predictable, repetitive activities e.g. expenses claims processing;

ad hoc workflows involve human coordination, collaboration and co-decision e.g. software development projects;

production workflows involve predictable repetitive processes which require structured access to enterprise information systems e.g. insurance claims processing, telephone sales.

It should, however, be noted that the the above workflow classes are not mutually exclusive. Thus, there is increasing interest in supporting hybrid workflow scenarios within the

[1]Sheth cites a number of papers providing commercial workflow product surveys in [106, p. 3].

emerging field of *multi-paradigm workflow* e.g. [89]. A typical workflow management system (e.g. IBM FlowMark, Plexis FloWare) provides a set of tools to enable the modelling, enactment and monitoring of business processes as shown in Figure 2.6 [46].

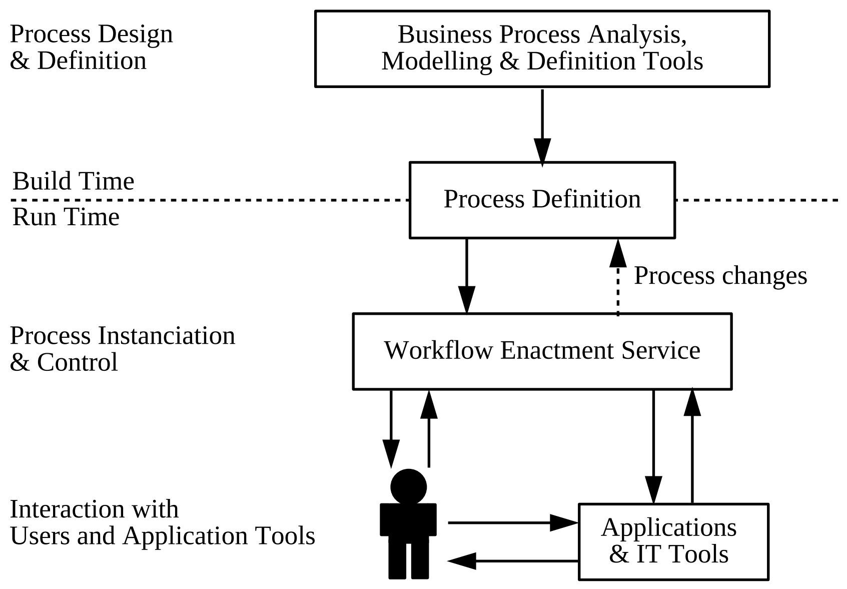
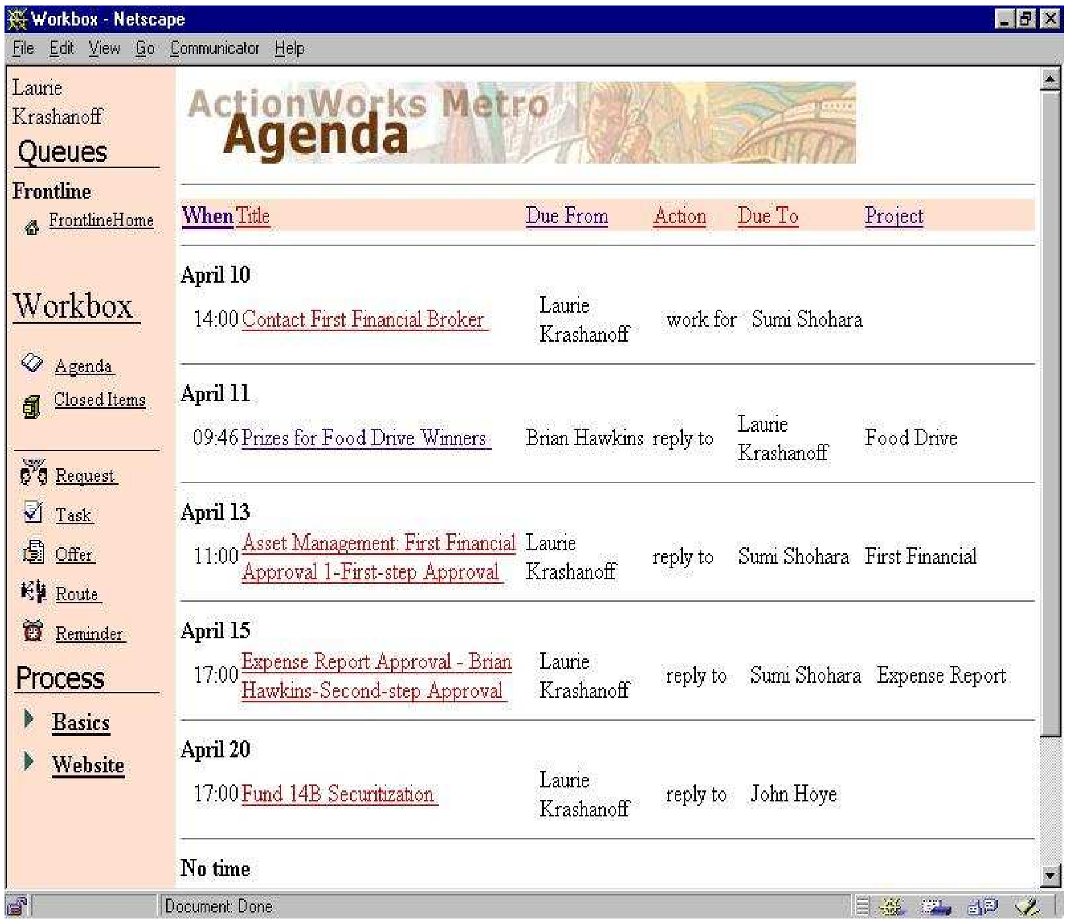


Figure 2.6: Workflow system characteristics

At the modelling stage, a representation of the process is built using a graphical or language-based description technique. Most existing commercial systems provide graphical support for process modelling using IDEF0 [82], Role Activity Diagram (RAD) [76] or similar approaches. Action Technologies advocate a conversation based workflow modelling approach with their ActionWorkflow product set [74] based upon the original Coordinator system developed by Winograd and Flores [31]. Most modelling environments enable incorporation of information flow characteristics within workflows enabling, for example, definition of the information objects modified by an activity. A variety of information modelling mechanisms are implemented within current WFMS solutions, most of which are compliant with the standard entity-relationship model [19] e.g. IDEF1X [83].

Once the activity and information characteristics of the business process have been modelled, a workflow enactment engine is responsible for managing its execution. A provisioning phase maps activities to processing entities such as people, machines or systems. Several systems support *pooling* of resources to enable dynamic balancing of incoming work e.g. a new job is automatically routed to the person with fewest current assignments. End users typically interact with conventional WFM systems via a *worklist* as shown, for example, in Figure 2.7.



When	Title	Due From	Action	Due To	Project
April 10					
14:00	Contact First Financial Broker	Laurie Krashanoff	work for	Sumi Shohara	
April 11					
09:46	Prizes for Food Drive Winners	Brian Hawkins	reply to	Laurie Krashanoff	Food Drive
April 13					
11:00	Asset Management: First Financial Approval 1-First-step Approval	Laurie Krashanoff	reply to	Sumi Shohara	First Financial
April 15					
17:00	Expense Report Approval - Brian Hawkins-Second-step Approval	Laurie Krashanoff	reply to	Sumi Shohara	Expense Report
April 20					
17:00	Fund 14B Securitization	Laurie Krashanoff	reply to	John Hoye	
No time					

Figure 2.7: Action Technologies Metro worklist interface

The worklist provides summary information for each piece of work assigned to a workflow participant and provides forms through which the work can be completed or progress tracked. Dependent upon the class of workflow, the worklist may be email, document or process-oriented. For example, an administrative workflow may involve simple email routing of on-line expense claims; processing insurance claims may involve routing of scanned forms via document imaging technology; industrial workflow may track products through several stages of their production process. In some cases, the worklist interface will also integrate access to other applications and tools required during task performance. For example, access to imaging equipment may be provided at the worklist interface for tasks requiring document routing.

Most large WFMS implementations support application programming interfaces of some kind (usually proprietary). In this work it is assumed that implementations are structured according the WFMC reference model as shown in Figure 2.8 [46].

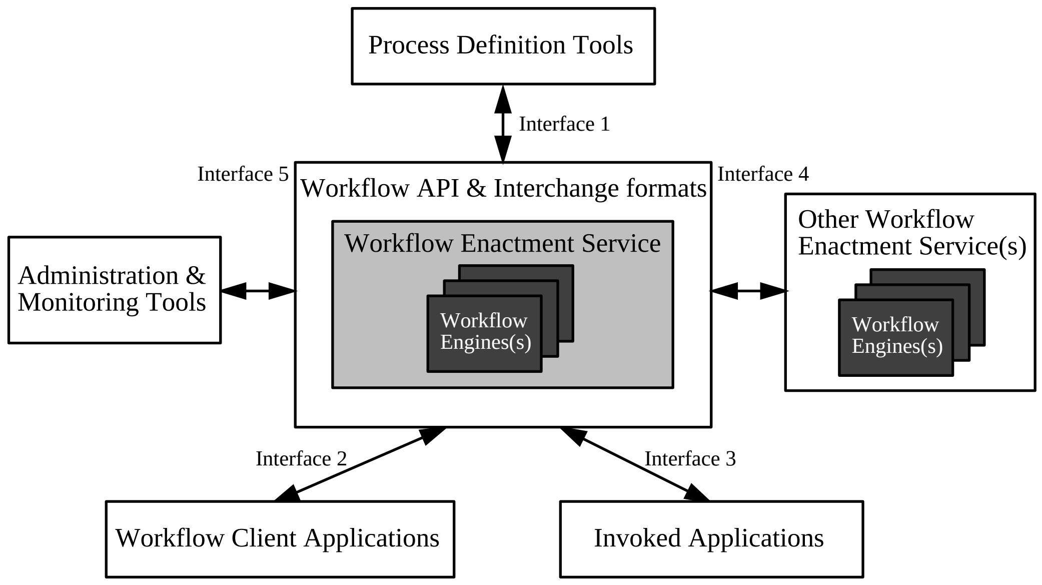

Process Definition Tools
Interface 1
Interface 5
Workflow API & Interchange formats
Interface 4
Workflow Enactment Service
Workflow Engines(s)
Other Workflow Enactment Service(s)
Workflow Engines(s)
Administration & Monitoring Tools
Interface 2
Interface 3
Workflow Client Applications
Invoked Applications

Figure 2.8: WFMC Workflow reference model–components and interfaces

The majority of major existing WFMS enactment services are either compliant with the WFMC reference model, or can be augmented with wrapper interfaces to achieve compatibility. The WFMC reference model is significant to developers of coordination-based CSCW systems as it defines how workflow services interoperate with each other and with external systems. The interfaces of primary importance to this work are those supporting process definition interchange (Interface 1), workflow client applications (Interface 2) and invoked applications (Interface 3).

Workflow definition interchange Interface 1 of the workflow reference model provides a separation between the build-time and runtime workflow environments. It's primary purpose is to link process definition tools with the enactment service in a standard way. Business process changes are accommodated at the enactment level through modification and resubmission of the process definition through this interface. Most existing WFM systems support proprietary process definition techniques and representation languages; cf. ActionWorkflow [74], FlowMark [49] and InConcert [72] for example. There is however, increasing interest in process definition interchange languages which enable models to be shared between different WFM systems. The Process Interchange Format (PIF) [63] is one such translation interlingua for workflow definitions which is of particular interest to this work. PIF describes processes according to a frame-based declarative approach with KIF-like syntax [33]. The core set of fundamental PIF language constructs (e.g. describing activities and constraints between them) is self-extensible through a mechanism of partially-shared views (PSVs) [64]. PSVs enable new concepts to be expressed in a PIF compliant way without violating the existing language.

Workflow client interface Interface 2 provides a mechanism through which client applications can interact with the workflow engine. A worklist handler is the most common type of client application, through which a human user is presented with tasks for completion. Some workflow systems provide API facilities which enable customised client software to be created but this usually requires programming changes to executable code. A flexible approach taken in many newer workflow products is to provide Web access via HTML forms interfaces e.g. Metro (Figure 2.7).

Invoked applications interface Interface 3 defines an execution and control mechanism for sessions with applications invoked during a workflow session. There are many conceivable examples of such applications although the most common usage scenario involves provision of access to enterprise information systems. For example, as part of a goods ordering workflow, access to a stock database management system might be provided. This interface could also be used to manage access to collaborative working tools as required within this research, although suitably exposed control interfaces would be required of invoked groupware tools.

2.3.2 Collaboration Services

Within a CSCW architecture, collaborative services provide system-mediated support for *ad hoc* human-human interaction within the distributed environment. Johansen's space-time matrix is often used to classify implementations [54], as shown in Figure 2.9.

Synchronous colocated collaboration tools support real-time face-to-face interaction. Most groupware implementations supporting this type of interaction are electronic meeting room or group decision support systems, designed to aid group brainstorming sessions by overcoming problematic characteristics of conventional meetings e.g. poor floor control, power struggles etc. Typical architectures provide each meeting participant with a workstation enabling (potentially anonymous) access to a shared whiteboard or doc-

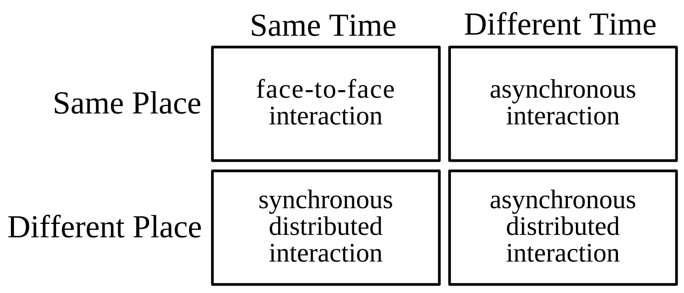


Figure 2.9: Space-time groupware taxonomy

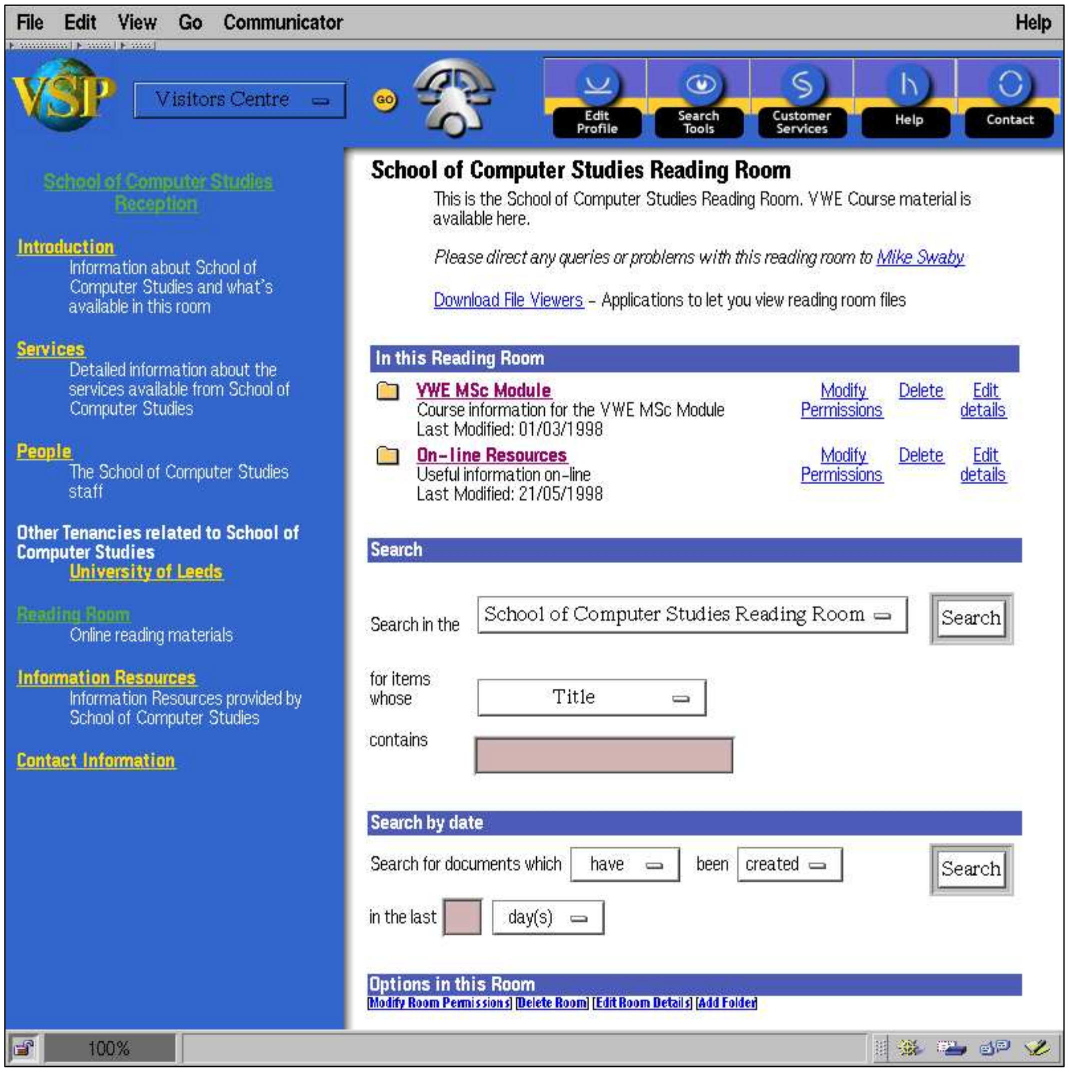
ument. Well known implementations include the Capture Lab [69], Liveboard [28] and Colab [109].

Asynchronous collaboration tools are in common use e.g. electronic mail, news and bulletin board systems would all be characterised within this class of application. Same-place different-time groupware implementations are rare, although shift-based environments such as helpdesk management tools fall into this category. Integration of message-based asynchronous groupware facilities within internet-based CSCW architectures is uncomplicated. Major vendors (e.g. Microsoft, Novell, Netscape) generally ship mail and news client software free of charge, choosing instead to charge for server side components which integrate closely with network operating systems. As Internet mail and news protocols (e.g. SMTP and NNTP) have been widely adopted there is a large and expanding range of interoperable asynchronous collaboration technologies based upon these standards.

The major remaining problems with respect to integration of message-based collaborative services within CSCW architectures are security and meta-information consistency. Access control can be a particular problem in situations where collaborative groups span multiple systems, security domains and/or organisations. Due to the general lack of integration of mail, conferencing and other groupware tools in current enterprises it is also common to find significant duplication of information between services. For example, a personal email address book may contain user information duplicated in an organisational directory, potentially leading to synchronisation problems. This problem is diminishing within internet environments however, as the use of information management services becomes more common.[2]

[2]Section 2.3.3 discusses information management services in greater detail.

Another important class of asynchronous groupware service are **document management systems** e.g. DocMan [5] and BCSCW [13]. Web-based implementations are becoming increasingly popular as solutions can be created easily using standard infrastructure services (e.g. Web servers, server access controls, document upload etc). A document management facility has proved to be one of the most popular features within VWS implementations. The reading room service, shown in Figure 2.10, enables VWS tenants or user communities to create private repositories through which documents may be stored and exchanged.



School of Computer Studies Reception

Introduction
Information about School of Computer Studies and what's available in this room

Services
Detailed information about the services available from School of Computer Studies

People
The School of Computer Studies staff

Other Tenancies related to School of Computer Studies
University of Leeds

Reading Room
Online reading materials

Information Resources
Information Resources provided by School of Computer Studies

Contact Information

School of Computer Studies Reading Room

This is the School of Computer Studies Reading Room. VWE Course material is available here.

Please direct any queries or problems with this reading room to Mike Swaby

Download File Viewers – Applications to let you view reading room files

In this Reading Room

VWE MSc Module Course information for the VWE MSc Module Last Modified: 01/03/1998	Modify Permissions	Delete	Edit details
On-line Resources Useful information on-line Last Modified: 21/05/1998	Modify Permissions	Delete	Edit details

Search

Search in the: School of Computer Studies Reading Room [Search]

for items whose: Title

contains

Search by date

Search for documents which have been created [Search]

in the last day(s)

Options in this Room

[Modify Room Permissions] [Delete Room] [Edit Room Details] [Add Folder]

Figure 2.10: VWS reading room (© 1998 University of Leeds)

Reading room services provide access controls on documents, enable version control for collaborative authoring and can also check incoming material for viruses. Currently, reading rooms have to be set up by system support staff on an individual basis. Once this setup has taken place, however, nominated users may create new sub-folders and control the management of their own reading room areas.

Synchronous distributed collaboration tools enable human-human communication across geographically separated spaces. One of the most successful initial research efforts in this area was the Media Spaces work at Xerox during the late 1980s [44]. The first generation of CSCW *infrastructure* projects were also directed towards support for synchronous distributed collaboration, notably Rendezvous (shared real-time interfaces) [93], Liza (shared user interface objects) [35] and Shared X (shared windowing) [43]. Related research work within the CVWE at Leeds University has investigated conference architectures for use in Virtual Working Systems. In [48], Hunter proposes an architecture for secure, user-centred conferencing within VWS implementations based upon the ITU H.323 and T.120 standards for audiovisual and data conferencing (Figure 2.11).

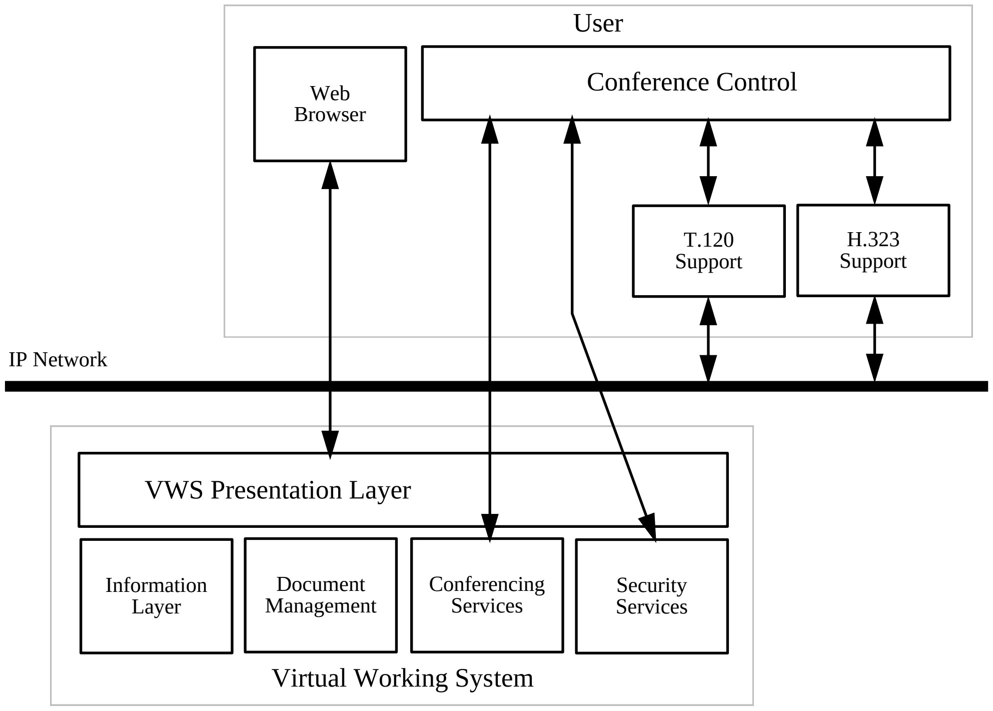


Figure 2.11: VWS secure user-centred conferencing architecture

An initial implementation of this architecture integrates directory and security services to create a person-centric conferencing environment within a VWS based around Microsoft's NetMeeting synchronous groupware product. NetMeeting facilitates audiovisual conferencing and application sharing within an internetwork environment. The work has important implications for this research as it demonstrates how synchronous collaboration services can be accessed within an integrated CSCW environment.

2.3.3 Information Management Services

Most cooperative working scenarios involve collaboration over shared information artefacts. Services that manage access to such artefacts are therefore of central importance in CSCW architectures. The goal of information management services is to provide CSCW system users with an interface to a virtual *information space* appropriate to the task(s) in which they are engaged. There are a large number of existing individual systems and tools which would be positioned as information management services within an integrated CSCW framework. Taking for granted basic database management tools and systems, most services of interest to this work can be characterised within three broad classes;

- Information integration,
- Organisational context provision,
- Value-added information services.

Information integration

In many collaboration scenarios there is a basic need to share heterogeneous information objects. Within the University of Leeds Virtual Science Park, for example, tenant organisations typically already have their own internal information models and database instances which need to be integrated into the VSP information space. Information integration services encompass a variety of mechanisms that enable databases and other information sources to be accessed in a uniform manner. This layer of service typically provides *syntactic* integration before value-added services such as mediation software provides *semantic* interpretation of this information e.g. the information integration and mediation layers of the VWS framework as shown in Figure 1.6.

Information integration services attempt to insulate end-users from the complexities of underlying database structures by mapping information into a structure with which they are familiar. For example, a VWS implementation may utilise several SQL relational databases but the system user is hidden from the physical database structure through integration and presentation services which create a consistent view of the information space.

There are a number of feasible approaches to integration of multiple databases [65] at the syntactic level. The most commonly practised technique is to map underlying data models (and thereby, instances) into a global conceptual schema [10]. This approach is taken in the VSP and related VWS implementations through the use of database wrappers, interactive translation tools and bulkload adapters [25]. Alternative techniques (e.g. federation and semantic unification [95, pp. 1–14]) rely on higher level services to provide more sophisticated integration services.

In this work there is a clear need to support information integration services, but the research requirements can largely be met by existing tools. The assumption made with respect to information integration services (at a system architectural level) is that, whilst a range of heterogeneous database systems must be supported (e.g. relational data in SQL tables, organisational directory information in X.500), all such data structures can conceptually be characterised using a conventional entity-relationship (E-R) model [19]. Since the relational model of data is theoretically rigorous, this is a reasonable assumption to make. The practical implication for this work is an emphasis on achieving integration using a standard approach, rather than aiming to provide a computationally efficient implementation. It should, however, be possible to create an efficient implementation of the architecture within a production-quality system.

Organisational context provision

Work is generally situated within an organisational context. In a physically-colocated work scenario this context is implicit in the familiar office environment, but in a computer supported geographically-distributed collaboration scenario this context must be represented explicitly within the system [6]. A model of organisational context within a CSCW environment describes the objects and relationships that are of interest in a particular enterprise, e.g. people, projects, roles management structures, locations etc. [45]. Once the enterprise has been characterised according to this model, CSCW applications can be built that operate (and interoperate) within it. For example, within the VWS environment proposed in [48], multimedia conferencing clients use a shared directory service as an address book and session record.

The standard E-R model (as identified when discussing information integration services) is appropriate for modelling organisational context, although a range of representations have been adopted in existing systems. A common implementation technology, especially within large enterprises, is the use of directory services such as X.500 [51]. Directory services are designed primarily to provide highly scalable *white-pages* facilities which reflect organisational hierarchies. In order to support a richer modelling capability (e.g. *yellow pages services*) the standard X.500-style directory model is often augmented with extra object classes/attributes. For example, the proof-of-concept VSP implementation [24] developed a person-centred model of organisational context that ensured navigation or search of the information space would always lead to a *person*. This was achieved by implementing a typed relationship model over the standard X.500 structural model, as shown in Figure 2.12.

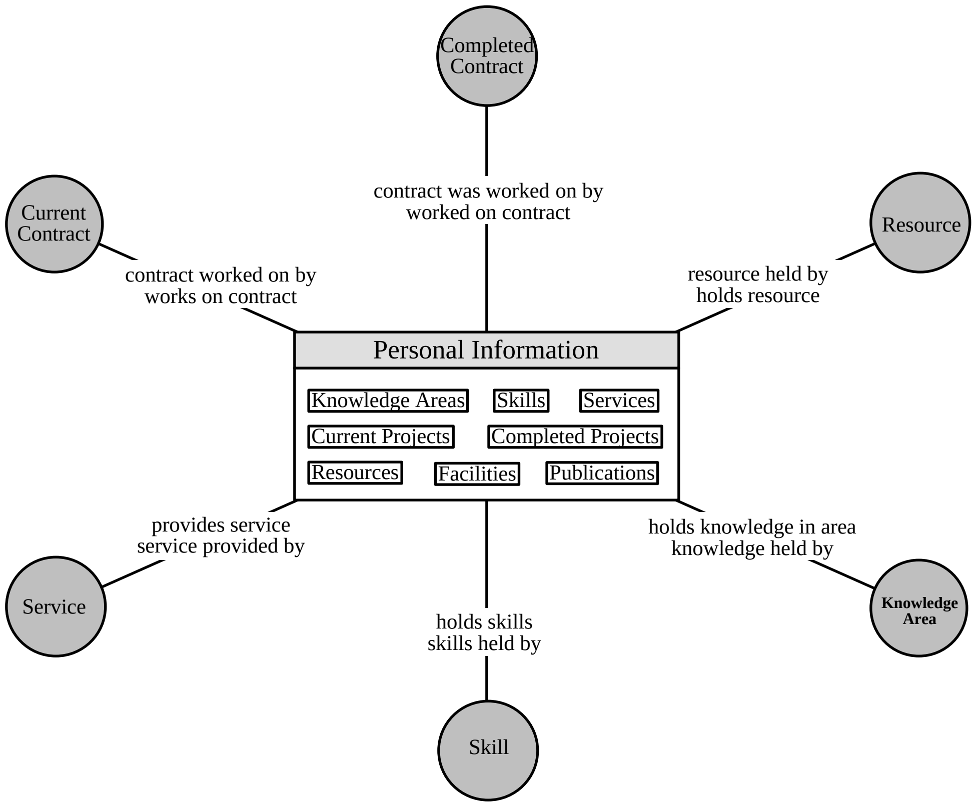


Figure 2.12: Person-centred VSP information model

In the extended directory model, new directory entry classes were created for six types of enterprise objects of relevance to the VSP domain (Organisation, Contract, Person, Expertise, Service, Facility). Typed relational attributes were then associated with instances of these classes e.g. Person *works for* Organisation. This model enabled intuitive information services to be constructed, enabling efficient resolution of queries such as "*find all information about service X; which companies provide X; who within these companies should I speak to about X*" etc.

Directories (especially when extended to enable richer descriptive capability) provide a highly appropriate medium for representing organisational information within integrated CSCW environments [98]. Research systems such as the Virtual Science Park [24] and Nexor's Enterprise Information System [45] were amongst the first projects to utilise extended directory services within CSCW environments. Since the initial phases of these projects, several major network computing vendors have embraced directory technology (cf. Netscape, Novell, Microsoft). This is fuelling increased organisational adoption of directory services and, significantly for this research, Internet-based directories to support inter-enterprise collaboration.

Another relevant perspective on organisational context provision is that of researchers working on *executable enterprise models*. For example, the Toronto Virtual Enterprise (ToVE) project [8, 9] developed a set of organisation ontologies enabling a degree of automated *common sense* reasoning to be applied to an enterprise model. The ADEPT project, as introduced in Chapter 1, defines a business process ontology which could be viewed as focused subset of an organisational context model. The CIMOSA[3] framework [113] enables derivation of executable engineering enterprise models that can directly drive CIM processes. The AIAI Enterprise project [112] developed an ontology-base enterprise modelling toolkit enabling e.g. business process simulation and predictive reasoning.

Value-added information services

This broad class of services operates above syntactic information integration services as described previously. The goal of services at this layer is to provide (or assist in the) *semantic interpretation* of information sources mapped into the information space at the syntactic level. Within the VWS framework such services are provided by a *mediator* component based upon Wiederhold's I^3 architecture. In [117], Wiederhold provides the following definition of mediation:

[3]Open Systems Architecture for Computer Integrated Manufacturing (CIM).

> "a *mediator* is a software module that exploits domain knowledge about certain sets or subsets of data to create information for a higher level of applications."

There are many conceivable instances of such services e.g. information gathering, filtering, profiling, ranking, abstraction and reporting. For example, the Carnot project at MCC[4] [47] developed an agent-based architecture for semantic integration of heterogeneous information sources. This research was extended in the InfoSleuth project [119] which adapted the Carnot architecture to operate in dynamic information environments with no centralised control, such as the World Wide Web. The KRAFT Project[5] [39], within which BT Laboratories are a partner, applies constraint satisfaction techniques to mediation of heterogeneous information sources such as product specifications.

The above examples make use of AI techniques to encode domain knowledge and provide interpretive mediation services. This is not a pre-requisite for mediation services however. Within the VWS architectural framework, mediation services are provided through detailed domain classification schemes through which information is characterised. For example, within the VWS which supports the NEST Project,[6] users searching for particular services may only have a broad idea of relevant research areas. The use of detailed research classifications enables users to quickly narrow down their potential search space. Within NEST, the domain classification scheme is specialised for description of research information, but the general approach is widely applicable in different communities and problem domains [29].

The What's New and Relevant (WNAR) service is a further example of a value-added information service [25]. Here, VWS users can build up a personal profile describing subjects and events in which they are interested in (e.g. "*notify me when project information is updated.*"). The service has been implemented within the ADVISER project to provide a personalised notification service within the domain of EU RTD programmes [86].

[4] Microelectronics and Computer Technology Corporation.

[5] Knowledge Re-use and Fusion/Transformation

[6] Network for Exploitation of Science and Technology

2.3.4 User interface services

Within an integrated CSCW framework, user interface services are responsible for supporting and structuring user interaction with application services. The primary delivery medium for Virtual Working Systems is the Internet, and the World Wide Web in particular. User interfaces for Virtual Working Systems are therefore primarily constructed using the Hypertext Markup Language (HTML). The Web interfaces are built dynamically using compiled and scripted *gateway* programs operating via the Common Gateway Interface (CGI) [42]. These programs typically access information services and then generate HTML to present results to the user via the Web browser interface. For example, a personal office within the VSP (e.g. Figure 1.4 on page 7) is generated by retrieving a database object from an organisational directory and then populating a HTML template with this information. In taking a broader view of user interface provision within a CSCW architecture, there are three distinct but related requirements to consider;

Service synthesis at the user interface: to draw together access to application services via a user interface with a consistent look-and-feel,

Abstraction of presentation services: to support as wide a range of Internet browsers and support software as possible,

User interface adaptivity: to create user interfaces that more closely reflect the requirements of individual users or classes of users.

2.3.4.1 Service synthesis at the user interface

Many existing Web–database connectivity engines provide good support for generation of standard user interfaces for navigation and search of relational databases e.g. Microsoft Active Server Pages and Allaire Cold Fusion products. Such products typically rely upon the creation of HTML templates into which chunks of executable script are embedded cf. [62]. Templates may be created manually or, in more sophisticated development tools, through a visual interface. At runtime, when a particular page representing a database query is requested through the Web server, the embedded script is executed and the result set merged into the HTML template. The populated HTML page is then forwarded back to the client browser for rendering. Template-based approaches to Web interface generation for information services have proved to be appropriate in many application domains. It is, however, difficult to integrate other types of service (e.g. coordination or collaboration services) through this interface generation technique in existing commercial systems.

2.3.4.2 User interface adaptivity

Cooperative working scenarios often involve people with differing perspectives on the collaboration. The perspective they hold upon the collaborative scenario might be defined or influenced by their role, responsibilities, expertise or personal preferences. Differing perspectives are often reflected in differing user requirements in interaction with the cooperative system. For example, a workflow user in an administrative role would be primarily concerned with their own worklist, whereas a managerial user might require an overview of the progress being made by all subordinate staff; an expert VWS user may prefer a terse informational interface whereas a neophyte may prefer a more intuitive interface. Thus, there is a basic requirement for the CSCW environment to adapt it's user interface to meet the heterogeneous requirements of the user community. In [104], Schneider-Hufschmidt et al provide an informal rationale for adaptive user interfaces, noting four general cases in which adaptive interface behaviour can be beneficial;

1. a system is used by users with different requirements,
2. a system is used by a user with changing requirements,
3. a user works in a changing environment,
4. a user works in different system environments.

Most CSCW scenarios exhibit at least one of these properties, hence it is useful to incorporate some user interface adaptivity features within a general CSCW architetcure. This work is directed towards an investigation of support for CSCW in *internet* environments, hence focusing upon Web user interfaces.

Within internet environments, there are two basic classes of interaction supported within major browsers;

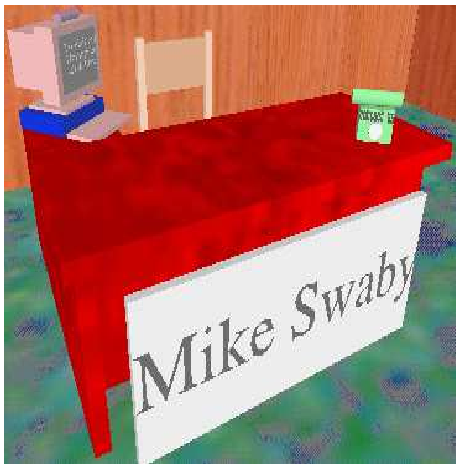
- 2D interaction through HTML Web pages
- 3D interaction through VRML worlds

Most existing Web information systems provide support for a single basic interactional style. That is, a uniform interface is specified for all system users. One barrier to flexible generation of Web user interfaces is the deep linkage of content and presentational constructs in HTML. This problem is well-known however, and emerging Web standards such as XML [50] and cascading style sheets [114] are moving towards a solution.

The pace of development of Internet technologies is so fast that it is difficult to decide when to upgrade Virtual Working Systems to include the latest features. A broad goal in this work is to assist in this dilemma by enabling new interface generation features to be embraced whilst still retaining regressive compatibility with existing techniques.

This problem is particularly apparent when considering 3D Internet interaction [26]. The Virtual Reality Modeling Language (VRML) [94] and associated interface generation and browser software provides scope for multi-user Internet interaction through physical metaphors. As computing and network performance increase, 3D navigation and interaction will undoubtedly become commonplace within Internet computing environments. Existing research projects have demonstrated the benefits of 3D interaction in supporting exploration of large-scale information spaces e.g. cone trees [66], 3D database navigation in Virgilio [20], visualisation of organisational information in VR-Vibe [17]. In addition, physical metaphors (e.g. the rooms metaphor) can be used to create intuitive collaborative working environments (CVEs) e.g. cooperative working within Virtuosi [101] and DIVE [12]. Virtual Working Systems possess characteristics that suggest a 3D interactional style may be beneficial i.e. the basic requirement for distributed system support for interpersonal interaction within an organisational context and a large corpus of domain information. For example, within the Virtual Science Park project it has been appropriate to adopt a physical metaphor in order to provide an intuitive working environment for users (e.g. users log on through the *reception area* and then move into their home *tenancy* or the information services *building*). However, the role of 3D delivery technologies within VWS implementations is currently unclear. At the time of writing, Internet 3D technologies (e.g. VRML2, VRML EAI, Java3D, Microsoft Chrome) are in their infancy and there is much current debate as to their relative merits .

One foundation for the research reported within this thesis is the DiMe (Display Metaphor) language [79], built within the CVWE at Leeds University. DiMe is an Internet system architecture that enables the automated creation of 3D physical metaphors (e.g. rooms and offices) in VRML. The central feature of DiMe is an object-oriented scripting language (DMSL) that enables component based definition of user interfaces. For example, Figure 2.13 shows a metaphor used to generate a VRML office characterising a person object within the VSP information space. Later work on DiMe extended the architecture to support output of HTML as well as VRML interfaces.



```
define OfficeMetaphor
extends RoomMetaphor {
used_to_display PersonObject
in_style 3D

set floor to_contain {
  add desk1 as Desk(name=$id) at 2,3
  add chair1 as Chair() at 3,3
}
set desk1 to_contain {
  add tele1 as Telephone() at 0,0
    Action select
      call $user audio
  add video1 as Videophone() at 1,1
    Action select
      call $user video
}}
```

Figure 2.13: DiMe metaphor example

2.3.4.3 Abstraction of presentation services

CSCW systems often support large number of users distributed amongst various organisational, and hence computing contexts. A major existing problem when developing cooperative applications within such domains is ensuring that user interfaces are consistent across computing platforms, operating systems and browser implementations. The simplest solution is to adopt the *lowest common denominator* approach by supporting a limited but ubiquitous set of interface constructs (e.g. the base HTML 2.0 standard). Whilst such an approach provides widest compatibility, it does not allow users to take advantage of enhanced interface functionality offered by browser providers. For example, the two most popular Internet browsers (Microsoft Internet Explorer and Netscape Communicator) both support proprietary tag sets that augment standard HTML as defined in the W3C HTML 4.0 specification [100]. In addition, there is an increasing emergence of client-side scripting languages [99] that provide support for interactivity within Web pages (e.g. JavaScript, JScript, VBScript, etc.) Again, there are many inconsistencies in scripting implementations within different browsers and across platforms. To alleviate these interface inconsistency problems, there is an increasing interest within the Web systems community in *abstract* interface generation techniques (as shown conceptually in Figure 2.14). This approach uses a neutral description language to generate Web interfaces tailored to particular platforms or browsers. The approach is useful within an internet-based CSCW system in supporting as wide a range of user platforms as possible.

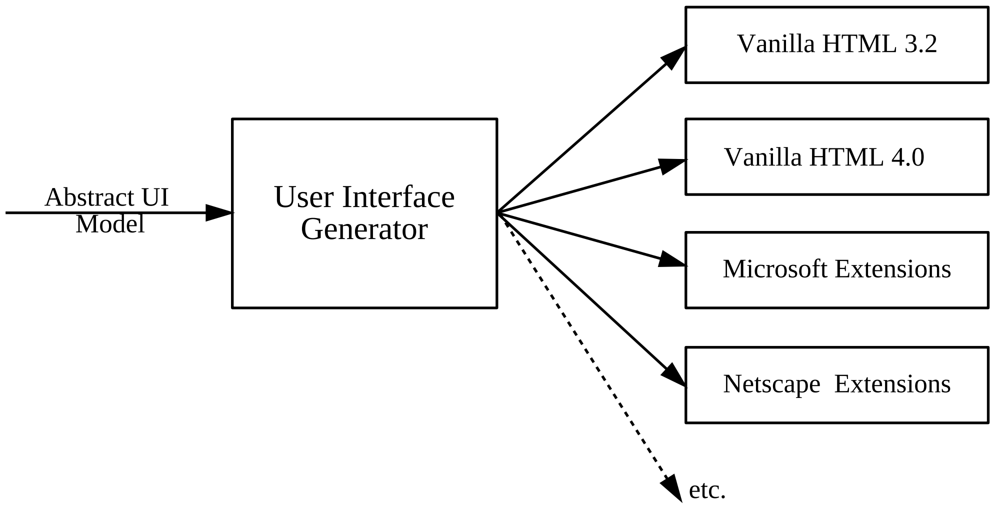


Figure 2.14: Abstraction of presentation services

2.3.5 Infrastructure Services

CSCW applications attempt to overcome several types of *distribution* e.g. geographical, organisational and/or temporal. Within an integrated CSCW environment there is therefore a clear requirement for basic support services to provide an infrastructure within which these distributed applications can operate effectively. Object-oriented distributed systems architectures such as ODP [23], ANSA [3] and CORBA [88] can provide many basic services from which to construct CSCW applications e.g. remote procedure invocation, distributed database access, synchronisation and security. Although there can be potential problems in utilising unmodified distributed object architectures to create

CSCW systems (such as unwanted transparency [14]), they can provide a appropriate foundation upon which which to construct integrated systems. Services requiring integration may be purpose built (e.g. VWS services) or exist as legacy systems (e.g. workflow managers). Distributed object services can enable integration through *wrapping* existing applications to make them accessible to other objects within the collaboration environment. A further advantage of a distributed object approach is that several generic (and hence reusable) services are usually provided, cf. support for object relationships, security, event management and database access in CORBA for example [87]. Utilisation of these common object services reduces developer effort and promotes modularity and component re-use, which are major objectives in this research. Existing research projects have produced promising results in applying distributed object technology in delivering internet-based systems. For example, CorbaWeb [75] enables transparent access to distributed object applications through a standard Web browser interface. Web* [1] utilises a similar template-based approach to provide integration services within CERCs Information Sharing System project.[7] ORBWork [22] provides a distributed and scalable workflow enactment environment based upon Web infrastructure.

2.4 Summary

Through a survey of a number of existing CSCW frameworks, this chapter has identified several core services that are required within a general CSCW architecture. Relevant related research and commercial development efforts have then been described within these service classes. The objective in this study is to identify structured mechanisms through which individual services may be brought together within a CSCW architecture. A number of conclusions may be drawn from the investigation, which serve to guide development of the architecture proposed within this thesis;

Coordination services provide support for managing the interdependencies between resources contributing towards a pre-defined or emergent common goal. Integration of coordination services within a CSCW architecture requires an external control interface. The workflow management coalition's reference architecure (WFMC-RA) was described as an example of a coordination service which could feasibly be integrated within a general

[7]Concurrent Engineering Research Center, West Virginia University

CSCW architecture.

Collaboration services enable inter-personal interaction between members of a cooperating group. Integration of such tools within a CSCW architecture is largely dependent upon the extent to which they support external control via a programming interface.

Information management services integrate domain information sources into an *information space* appropriate to the tasks undertaken within the cooperative working scenario. A number of information service layers were identified, within which related work was positioned. The requirement within this work is to provide basic mechanisms for information access, upon which services implemented within these layers might be implemented.

User interface services enable system users to interact with the CSCW environment in an appropriate manner. Three main roles for a user interface service within a CSCW architecture were identified; service synthesis at the user interface, abstraction of presentation services and user interface adaptivity. The DiMe architecture, developed within the CVWE, was introduced as a flexible Web user interface generator which can be used to deliver user interfaces within a CSCW architecture.

Infrastructure services provide middleware components to integrate CSCW services together within a systems environment. Distributed object architectures such as CORBA were identified as useful in enabling interoperability between CSCW service implementations.

Chapter 3

Model-based CSCW architecture

Chapter 1 introduced the research problem as a general requirement for structured development techniques for integrated internet CSCW systems, that promote service component reuse and enable rapid iterative development. Chapter 2 then characterised existing CSCW services and surveyed related research and development efforts within five broad classes (coordination, collaboration, information management, user interface and infrastructure). This survey identified mechanisms through which services might be brought together into integrated applications by a CSCW architecture. This chapter describes the detailed design of *ParaDiMe*, an architecture for integrated internet CSCW systems which supports a *model-based approach* to iterative systems development. The research hypothesis is that the proposed model-based approach can solve the research problems identified in Chapter 1.

The basic conceptual approach taken within this work is to raise the level of abstraction at which cooperative applications are built and subsequently modified over time. The research problem is fundamentally a component integration and change management problem, current solutions to which are generally applied at the lower levels of layered systems architectures. For example, CORBA supports integration of heterogeneous client-server components through mutual commitment to a common remote procedure call (RPC) mechanism. However, application end-users and the requirements analysts who represent them are largely unconcerned with such levels of architectural detail. When building prototype system implementations or designing systems according to a Rapid Application Development (RAD) method, there are two basic considerations that guide initial development;

Functionality: what functional services are required from the supporting environment,

User interaction: how end-users interact with these services via the human-computer interface.

The approach taken in this work is to provide explicit architectural support for building cooperative systems via component-based *models* which describe user interaction with functional services. Basic CSCW services are often shared between applications, with user interface requirements differentiating functionally similar implementations. Using a model to map between application functionality and the embodiment of that functionality at the user interface offers several advantages over conventional approaches;

- Access to applications is structured and encapsulated, promoting re-use;
- User interface changes can be made at a modelling level, rather than requiring code recompilation, reducing software development effort;
- Several different user interface component sets may be mapped to the same functional components, enabling adaptivity;

The model-based approach is illustrated conceptually in Figure 3.1, comprising of four major interacting system components within the ParaDiMe architecture;

A library of user interface models is defined using a purpose-built modelling language which describes how system users interact with application and component services.

A runtime interface generation service provides application user interfaces at runtime from user interface models defined using the modelling language, directed by commands from the application-specific control component.

A library of CSCW services provides implementations of CSCW services that are built specifically to enable their re-use in different applications.

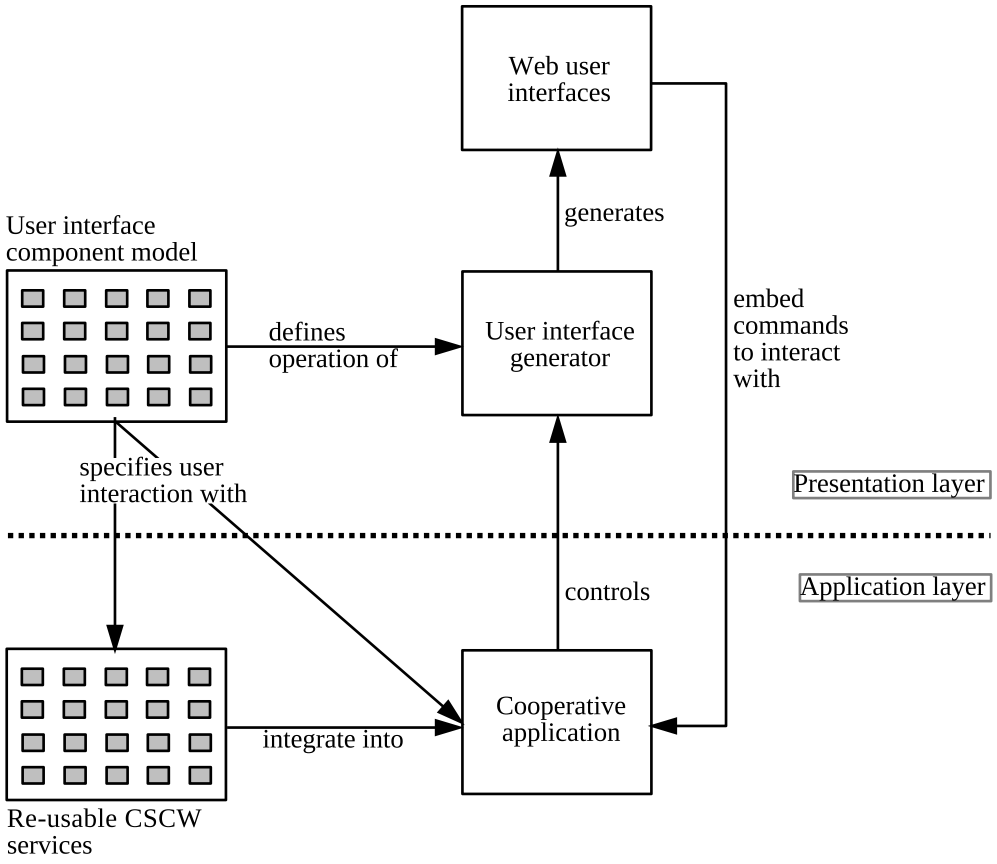


Figure 3.1: Conceptual view of the model-based approach

Application components implement application specific functionality and integrate access to general CSCW services where required.

The ParaDiMe architecture was developed to enable assessment of the proposed model-based systems development approach by supporting exploration of case study CSCW applications. The central architectural component within ParaDiMe is the DiMe user interface generation toolkit [79], created within the CVWE by David Morris and Gyuri Lajos who acted as technical supervisors in this research. ParaDiMe extends the capabilities of DiMe to facilitate rapid iterative development of integrated internet-based CSCW systems [111].

The role of the first generation of DiMe architecture was to automate construction of VRML and HTML user interfaces to organisational directory servers. A batch mode DiMe implementation was built in order to generate VRML models of the directory using a 3D rooms metaphor. In addition, an interactive Web gateway implementation was constructed which enabled HTML interfaces to be created dynamically as users navigate around the directory. In isolation, DiMe performs a similar job to conventional Web database scripting technologies (e.g. Active Server Pages and Cold Fusion) as shown in Figure 3.2. In these systems, output display language templates (e.g. HTML templates) describe how to present query results at the user interface. An incoming query is processed by an evaluator component, producing a set of search results. These results are then merged with the output display language template and returned to the client browser.

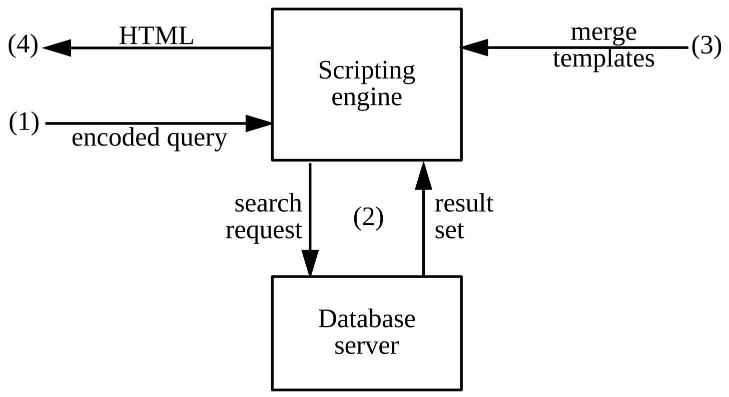


Figure 3.2: Web database scripting

DiMe extends the conventional template-based approach to Web interface generation through the definition of an object-oriented scripting language called DMSL (Display Metaphor Scripting Language). There are two fundamental classes of language construct within DMSL;

1. user interface component definitions called *metaphor definitions*,[1]
2. *commands* which are used to drive the generation process at runtime.

As a precursor to runtime interface generation, a set of DMSL metaphor definition scripts are parsed to create an in-memory abstract representation and a metaphor *index* in tabular form. At runtime, a DiMe command then triggers a metaphor selection process within

[1]The term *metaphor* has been used throughout the development of DiMe over several years and is deeply entrenched within the software and documentation, largely as a result of the initial application towards generation of 3D rooms and offices using VRML. In retrospect however, our use of the term is incorrect. Whilst a virtual room certainly *is* a virtual interface metaphor [96], the vast majority of the components which are referred to as metaphors within DiMe are no more than user interface building blocks (i.e. interactors or widgets). The term *metaphor* is retained within this thesis for consistency, but is used informally to mean *interface component* rather than meaning e.g. *mapping of concepts* [60].

DiMe. During metaphor selection, the index is used to identify and instantiate appropriate metaphors which form a tree representation. This tree is then passed to the DiMe output generators (for HTML/VRML etc.) which traverse the internal representation and emit appropriate target language constructs. This output is then returned to the requesting client through the Web server CGI mechanism. This template-based user interface generation mechanism forms the core of the ParaDiMe architecture through which the model-based approach to construction of integrated internet information systems is explored within this research. DMSL is used as the modelling-language which specifies user interaction with CSCW applications and the services they integrate.

The chapter begins by describing the DiMe object model (Section 3.1) and DMSL metaphor definition constructs (Section 3.3). An understanding of these fundamental DiMe elements is essential in introducing the ParaDiMe architecture which is built using them. Although the author contributed towards the development of these components, it should be clearly noted that they were invented by David Morris and Gyuri Lajos and hence, do not form part of the contribution of this thesis. However, it is necessary to describe them here because they are critical to the work and no suitable (published or internal) material exists to which the reader could otherwise be referred.

Building upon these existing DiMe components, Section 3.4 introduces the ParaDiMe runtime user interface generation service. This extends DiMe via a distributed object architecture which enables ParaDiMe to support integrated access to reusable CSCW services within applications. Within ParaDiMe, several basic mechanisms are provided which may be used to provide integrated access to applications and component services. An information management service is provided through which access to heterogeneous domain information sources is supported (Section3.5). A forms-processing subsystem enables user interaction with applications via HTML forms (Section 3.6). A remote method invocation service is provided, through which access to external services (e.g. workflow enactment services) is supported (Section 3.7). Finally, a collaborative tools subsystem is defined, which supports execution and subsequent session control of appropriate synchronous groupware tools (Section 3.8). Following discussion of these architectural components, from which integrated internet CSCW applications may be constructed, Section 3.9 describes a simple RAD-type methodology which may be used to guide development of systems using ParaDiMe.

3.1 The DiMe object model

Metaphor definition constructs in DMSL are directly reflected by internal operations on metaphor abstract data types (ADTs) within DiMe. It is therefore beneficial to discuss this internal representation before covering DMSL in detail. As mentioned earlier, DiMe currently supports integrated generation of HTML and VRML user interfaces. The first implementation of DiMe solely supported 3D interface generation however, built upon the SGI Inventor 3D graphics libraries [115].

Inventor (in common with several other similar products) provides a *scene-graph* API for construction of 3D models in which hierarchies of objects (called *nodes*) define parent–child relationships between objects. In simplified terms, nodes may represent particular objects in the scene (e.g. a cube or sphere), collect other objects together (e.g a grouping node) or apply some other operation on child nodes (e.g. colour, texture, or transformation). Container nodes implement methods which manage the set of children attached to them e.g. `add child, remove child`, etc. Rendering a scene to screen or saving a description as VRML requires traversal of the scene graph (defined by a root grouping node) by visiting subordinate nodes in a defined order. The object-oriented scene graph approach to 3D graphics programming is simple yet flexible. DiMe adopts this fundamental approach but with two important extensions;

- augmenting the 3D scene graph model with a 2D HTML document object model (DOM),
- an abstract layer of *built-in* objects with enhanced child management capabilities.

Hypertext Markup Language (HTML) uses tags to associate semantics with textual information. For example, `<title>My Page</title>` defines the title of a document to be the string "`My Page`." Upon parsing of this tag, a Web browser would typically name the window within which it is running accordingly. Tags such as `head` and `body` define the structure of the document, leading to a tree-based document representation very similar to scene-graphs in 3D graphics programming. A HTML document library was created using this approach and integrated with the 3D scene generation library within DiMe, as shown in Figure 3.3.[2]

[2]The World Wide Web Consortium have subsequently pursued an analogous approach through their

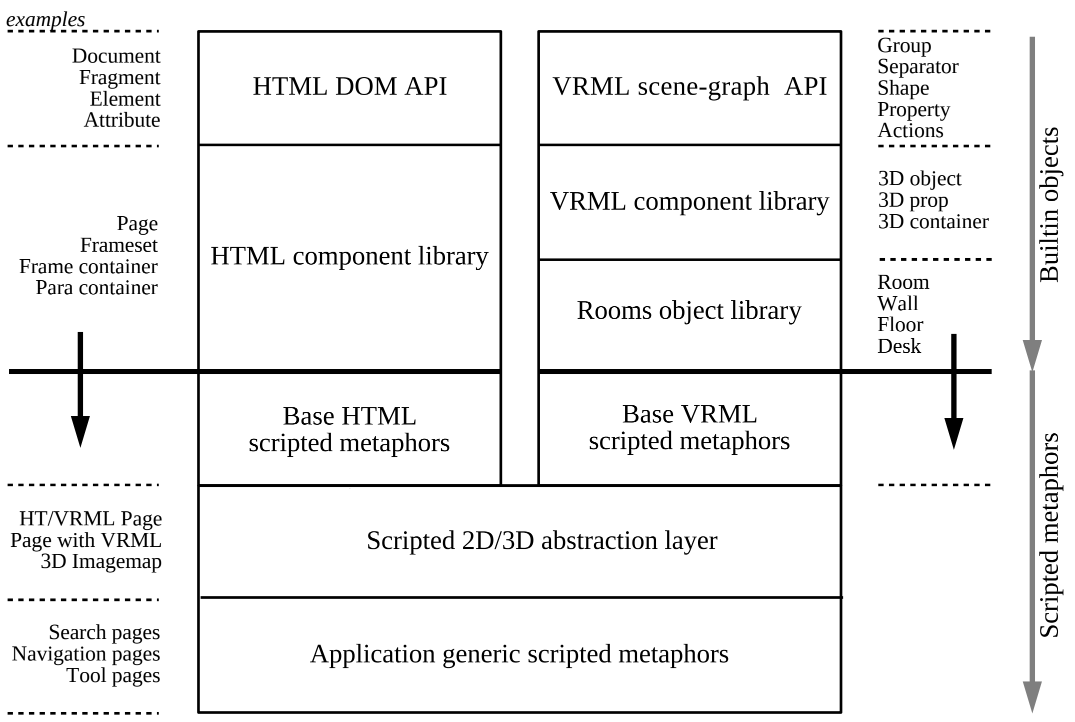


Figure 3.3: DiMe object model

This integrated approach to 2D/3D interface generation provides a common framework for developing different styles of interface using the same syntactic conventions within DiMe. Hybrid interfaces that integrate 2D and 3D interaction can also be supported elegantly through this approach. For example, addition of a 3D scene to a 2D HTML document can be specified such that upon output generation, the scene appears as an embedded VMRL window within the HTML browser.

Although powerful, a raw 3D scene-graph interface is more detailed than required in this work. Through the initial application to generation of 3D rooms, the spatial abstraction at which DiMe operates derives from a fundamental requirement for *planar* object management. This statement is best explained through an example; a room conceptually consists of a set of connected features such as walls, ceiling and floor; objects added to the room are associated with a feature within it e.g. one would expect a filing cabinet to reside on the floor and a picture to be hung on a wall. Physical laws and heuristics govern how objects exist and interact within the room e.g. discrete solid objects cannot generally occupy the same physical space or float in mid-air. However, most existing 3D APIs (or VRML Web browsers) will not enforce such rules and the responsibility for

Document Object Model initiative [121].

creating physically feasible scenes rests with the developer.

Supporting physically realistic object management within 3D environments is a very hard problem, but because DiMe is specialised for *room* scenes structured within a set of 2D planes, some useful support can be provided at low cost. The approach taken within DiMe is to provide a set of built-in 3D objects based on sets of 2D layout planes which embody simple physical and common sense heuristic knowledge about how child objects should be arranged. For example, a request to add three pictures to a built-in *wall* object will result in the pictures being *hung* evenly across the width of the wall at an appropriate height. Adding a telephone to a desk will place the telephone on top of the desk facing the front. A set of several appropriate built-in objects is defined within DiMe (e.g. rooms, walls, floors, desktops etc.) enabling realistic room scenes to be created at an appropriate abstract level within DMSL.[3] Basic support for HTML generation is also provided using built-in objects, although the API is much simpler as there is no requirement for object *layout* other than that implicit in HTML tag ordering.

3.2 Display Metaphor Scripting Language (DMSL)

Display Metaphor Scripting Language (DMSL) is a proprietary object-oriented scripting language which forms the core of the DiMe architecture. DMSL externalises the DiMe object model as described above via an intuitive high-level programming language. Within the current DiMe architecture, support for DMSL is implemented using the PCCTS compiler construction toolkit [91]. At the abstract level however, DMSL is a frame-based language [77] and it should therefore be noted that many other implementations are feasible.[4]

[3]Abstract spatial layout managers (e.g. supporting object grids, lattices and rings etc) have been specified within the DiMe architectured but have not been implemented successfully to date.

[4]For example, early experimental work packages within this PhD project investigated rule-based implementations of the DiMe architecture using Common Lisp [108] and CLIPS [34]. The investigation found such approaches to be highly appropriate for knowledge representation and reasoning within DiMe, but interactive performance was extremely inferior to the custom-built DMSL implementation.

The fundamental concept in DiMe and DMSL is the *metaphor*; A *metaphor* within DiMe represents a user interface component at some level of abstraction in the DiMe object model. The simplest metaphors act as wrappers for built-in objects in the (HTML and VRML) output generation APIs. Scripted metaphors then extend these base objects to create interface components specialised for particular purposes. DMSL is used to incrementally construct a (potentially complex) user interface model for a cooperative system from re-usable building-block metaphor components. At the conceptual level, a DMSL metaphor is a frame consisting of four sets of slots;

variable slots: maintain state through string, integer or floating point expressions,

content slots: define instances of component metaphors

constructor slots map a metaphor to a built-in object in the DiMe object model,

operator slots: specify actions, transformations and other operations to be applied to the metaphor at generation.

As a concrete example of how these slots characterise a user interface component, consider invocation of VWS videoconferencing services within a 3D room interaction style. A feasible interactional scenario is for the system user to navigate an organisational information space and locate the person they wish to contact, perhaps via a 3D representation of their *office*. Upon entering the virtual office, the system user establishes a videoconferencing session with the target user by clicking on a phone object positioned within the scene. This scenario implicitly embodies managed access to the application information space and structured invocation of a groupware tool. A DMSL metaphor can *package up* an executable model of these requirements that can be re-used and specialised across applications. So, a re-usable metaphor encapsulating the above properties might define the following slot values;

variable slots: videoconferencing address to dial, label to put on phone, user id of target user,

content slots: in this example the phone is an atomic object and therefore has no component metaphors,

constructor slots: the phone is an extension of a built-in `prop` object,

operator slots: upon clicking the phone, invoke a videoconferencing call to the target user.

The differentiating feature of DMSL (and therefore DiMe) is that it supports object-oriented concepts such as inheritance and overriding of slot definitions. It is, for example, trivial to extend the "*videoconference by clicking on phone*" metaphor as discussed above to create a related "*shared whiteboard by clicking on computer*" metaphor. There are several types of definition constructs within the DMSL language, as listed below. Because of the object-oriented nature of DMSL, within applications these basic constructs are typically amalgamated together into more sophisticated compound definitions;

- Metaphor definition constructs,
- Information search and navigational constructs,
- Interaction support constructs,
- Forms handling constructs,
- Remote method invocation constructs,
- Command constructs.

These constructs provide the basic functionality from which potentially complex user interfaces may be built. The following sections discuss these constructs and the architectural operations they represent within DiMe.

3.3 DMSL Metaphor definition constructs

The basic unit of abstraction within DMSL is referred to as a *metaphor definition*. A metaphor definition is either a wrapper for an object in the DiMe object model (Section 3.1) or a container for a set of subordinate metaphor definitions. Within DMSL, a metaphor is represented according to the grammar shown informally below;

```
1:    define <metaphor_id> extends <metaphor_id>
2:      <usage context definition>
3:    {
4:      <variable slot definitions>
5:      <constructor slot definition>
6:      <content and operator slot definitions>
7:    }
```

Lines 1 and 2 in the grammar represent the metaphor head constructs; Lines 4, 5 and 6 then define the constructor, variable, content and operator slots introduced earlier in this section. The first line of the definition provides a system-wide unique identifier for the metaphor and then, through the `extends` syntax, identifies the metaphor from which the current metaphor is to be subclassed at runtime. The base metaphor for any inheritance hierarchy of metaphors must represent a built-in object within the DiMe object model. The usage context definition construct (Line 2) provides a *hint* to the user interface generator about when it is appropriate to use the metaphor. The syntax of the usage context definition construct is shown below;

```
used to_display <objectclass> within <viewingcontext> in_style <style>
```

The `used to display` construct associates a class of information or application object with the metaphor definition, and is most commonly used in search and navigational metaphors described below. The `viewingcontext` parameter further contextualises the application of the metaphor and is typically used within 3D *rooms* style interfaces. For example, one metaphor might represent a `person` object as a picture in a `photograph album` viewing context whilst another might represent a `person` object as a name on a door in a `hallway` viewing context. For HTML interaction styles, a single default viewing context, called `Viewer`, is defined. Finally, an arbitrary `style` token may be associated with a metaphor to further guide automated selection of interface components at runtime. This construct is only useful when associated with a set of metaphors which together facilitate a defined style of interaction, and there is more than one style set to choose from in a particular interface generation scenario. In the VSP organisational browser application for example, two duplicate sets of metaphors are provided implementing `brief` and `detailed` views of each directory object e.g. a business card metaphor presents a person object in a `brief` style; a curriculum vitae metaphor presents the same object in a `detailed` interaction style;

```
define BusinessCard extends HTMLPage
  used to_display PersonObject within Viewer in_style Brief ...
define CV extends HTMLPage
  used to_display PersonObject within Viewer in_style Detailed ...
```

Definition of metaphors tailored to specific object classes, viewing contexts and styles is central to runtime interface generation flexibility within DiMe. The body of a metaphor defines values of four component slots as introduced earlier (variables, constructor, content and operator slots). The variable slot supports basic definition of integer, floating point and string variables, with identifiers scoped to the local metaphor. Literal and simple expression values are supported. The variable definition grammar is shown below; the `add` construct adds a new variable to the metaphor and assigns it an initial value, the `set` construct is used to alter the value of a defined variable, the (seldom used) `remove` construct removes an inherited variable from the current metaphor definition.

```
add|set <id> = <expression>
remove <id>
```

Variables are a critically important feature of metaphor definitions, providing the same functionality (and advantages) as function parameters in conventional block structured procedural programming languages. Within DiMe, metaphor variables fulfill two roles. Firstly, they can be used to specialise metaphors added at build-time to the content slot (covered later in this section). Secondly, they enable dynamic binding of variables as metaphor parameters at runtime (covered in Section 3.4).

The *constructor slot* within a metaphor can only be defined in base metaphors, and identifies which built-in object should be instantiated at runtime to generate the interface components represented in the metaphor. All metaphors must be associated with a constructor before interface generation, and once defined the constructor may not be overridden. The use of the constructor definition is illustrated in the example shown below. This DMSL fragment, taken from the DiMe VSP demonstrator, defines a base *bookcase* metaphor binding to a built-in 3D object called `vsBookCase`. Note the use of variable slots to provide parameters to the built-in object.

```
define BookCase extends Metaphor3D {
   add object_name = "BookCase"
   add number_of_shelves = 3
   add title = "default bookcase title"
   construct vsBookCase(&object_name, &number_of_shelves, &title)
}
```

The main body of a metaphor definition is comprised of *content* and *operator* slots. These specify component metaphor definitions (content slot) and optionally define some *action* to perform when the component metaphor receives relevant events (operator slot). As introduced in discussing the DiMe object model, the content slot is further subdivided into *feature* slots primarily to enable automated planar object layout in VRML user interfaces e.g. layout of books along a shelf in a bookcase. HTML metaphors do not require such layout assistance[5] and therefore usually only define a single feature representing the component in it's entirety. In general terms, a content slot definition within a metaphor is a set of feature constructs of the form shown below;

```
set <feature_id> to_contain {
  add <id> as <metaphor_operation>
  replace <id> as <metaphor_operation>
  remove <id>
}
```

Within a feature (e.g. a HTML `page`, VRML desk `surface` or `drawer`) parameterised component metaphors may be added to, replaced in or removed from the parent metaphor.[6] For example, a four-shelved bookcase may be added to the `rearwall` feature of an `office`

[5] Component layout should be done by the browser during page rendering.

[6] It is possible to redefine non-final metaphors using this syntax within DiMe (i.e. metaphors which have been subclassed elsewhere). Although this property can be useful in certain situations, we have yet to find a way of ensuring the integrity of orphaned metaphors other than writing out and reparsing the entire internal metaphor tree. Because of the expense of doing this, redefining non-final metaphors is not recommended to application builders.

metaphor using the following syntax (which also demonstrates variable overriding in content metaphor definitions);

```
set rearwall to_contain {
  add b1 as BookCase(number_of_shelves=4)
}
```

For each component defined within the content slot of a metaphor (e.g. as shown above) an operator slot definition may associate an *action* to be performed when a particular runtime event occurs that is of relevance to the component. This slot provides the basic mechanism through which all user interactivity is supported in DiMe. The operation of this mechanism will be discussed in detail later, but it is useful to introduce the basic approach at this stage.

Within standard HTML pages, the most common interactive operation is hyperlink selection e.g. clicking a textual, image or imagemap link. A wider range of events are feasible using client-scripting or within VRML browsers e.g mouse-over, proximity sensing etc. Through the operator slot definition construct, the DiMe runtime architecture supports three types of interactive operation;

1. generation of another metaphor,
2. execution of an information space query,
3. invocation of a remote application-layer method.

Within the runtime architecture, interactive operational support is achieved by embedding DMSL *commands* within URLs[7] that form part of the generated output language (e.g. HTML `href` or VRML `WWWAnchor` fields). This mechanism is shown conceptually in Figure 3.1, and will be discussed in detail in the following sections. However, an informal appreciation for the operation of the construct can be gained with reference to a simple example.

[7]Uniform Resource Locators.

```
add link1 as HyperTextLink()
  Action select view Data
  in "Viewer"
  in_style "DetailedPage"
  label 'View ' +  Data.name + ' in more detail.'
```

The above DMSL fragment shows a content definition with an associated operator slot definition provided through the `Action` syntax. This example is part of a HTML metaphor used in navigating an information space. It generates a textual hyperlink which, when selected, executes a DiMe command to view the current information object (denoted by the reserved `Data` keyword) in a more detailed style. Before describing support for interactive operations within the ParaDiMe architecture, it is useful to describe DiMe commands and the operation of the runtime interface generator.

3.4 Interactive operation of ParaDiMe

3.4.1 DMSL commands

The previous section introduced metaphor definition within DMSL. A set of appropriately-written DMSL metaphor scripts (utilising styles, viewing contexts, and object class references) provides a comprehensive and flexible user interface model for a collaborative application. DMSL is implementation-oriented however; that is, DMSL is designed to be executed as a scripting language by an interpreter or similar runtime processor rather than serving a purely representational role. Within ParaDiMe, which extends the DiMe output generation architecture, there is therefore a distinction between *build-time* when metaphor definitions are parsed to form an internal user interface model, and *runtime* when this model is used to guide *interactive* generation of HTML and VRML user interfaces (Figure 3.4)

At build-time, a set of DMSL metaphor definitions is parsed to form an internal representation of user interface components. Runtime interface generation is driven through the interpretation of a simple set of DMSL *command* constructs, guided by the internal interface model. The command interpretation process formulates a tree of metaphor ob-

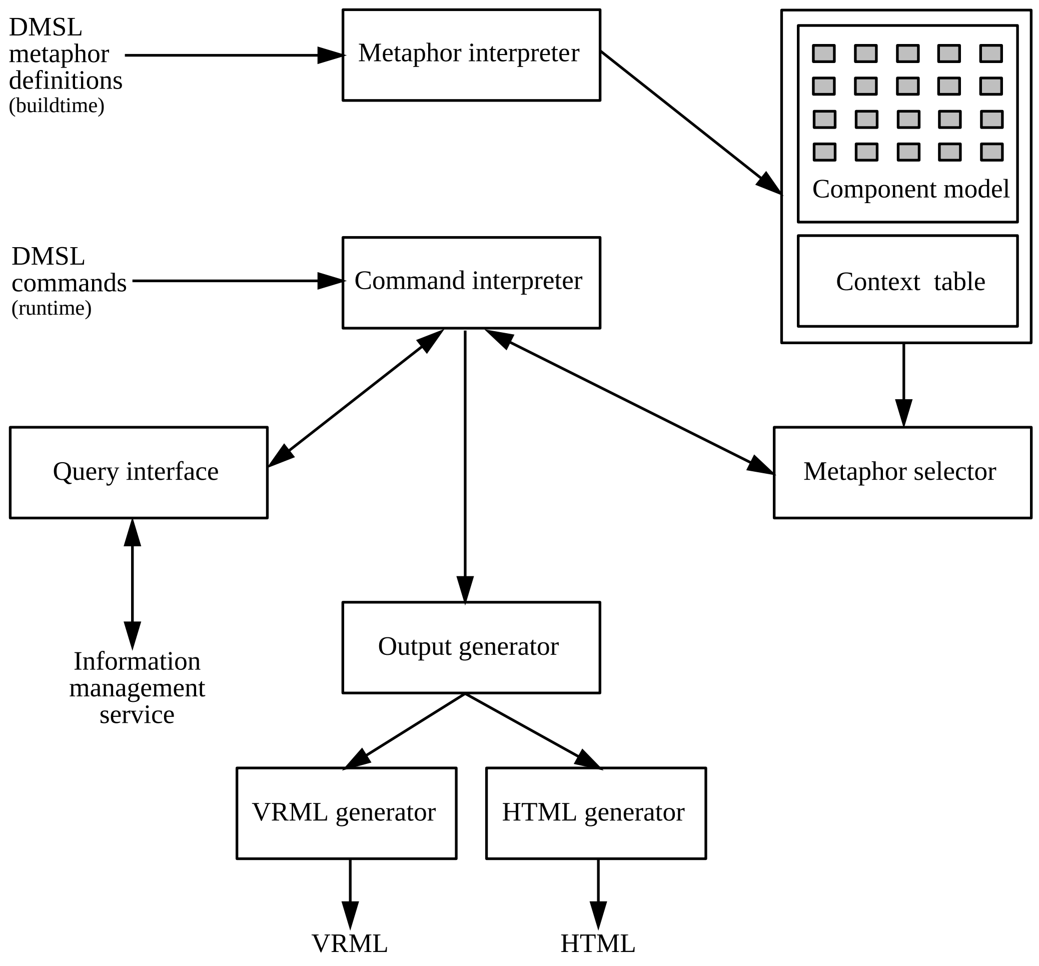


Figure 3.4: Runtime interface generation

ject instances. The objects within this tree are instances from the basic set of built-in objects from which all scripted metaphors must inherit. Following construction of this intermediate representation, the tree of built-in objects is passed to the appropriate output language generator.

The link between the DiMe object model and object-oriented constructs in DMSL becomes clear when considering the output language generation process. Built-In objects within the DiMe object model must support

1. mechanisms to add other built-in objects to themselves as contained children and,
2. mechanisms to support their own output language generation.

Therefore, given a container–component tree of built-in object instances, output language generation can be achieved via a top-down left-to-right tree traversal.[8] In order to initiate the output generation process, the ParaDiMe output generator must be provided with the name (and optionally instance parameters) for a particular metaphor. Although this information is often known explicitly by clients, ParaDiMe also supports system-initiated interface component construction through an architectural component called the *metaphor selector*.

As discussed earlier, DMSL metaphor definitions may contain context definition constructs which inform ParaDiMe about when it is appropriate to apply them. The construct is typically used to associate a metaphor with a particular class of object in the information space, a viewing context or an interaction style. During build-time interpretation of DMSL metaphor definitions, a context table (called the `used to display` table) is built internally. This table lists each metaphor known to ParaDiMe by object-class, context and style. The metaphor selector uses this table to automatically choose an appropriate metaphor during output generation in cases where a DMSL command does not reference a specific named metaphor. There is a single primary output generation command in DMSL (called `show`) which guides the interpreter according to one of three metaphor selection strategies;

- specific metaphor call,
- automated metaphor selection within current context,
- automated metaphor and context selection

The relationship between these commands and the potential number of metaphors they can match in a typical application is shown graphically in Figure 3.5.

The simplest DMSL output command is a **specific metaphor call**, which instructs ParaDiMe to generate output for a particular named metaphor. This command must therefore always map to a single metaphor definition within the internal interface component model. Execution of this type of command effectively bypasses the metaphor selector

[8]This technique reflects the scene-graph approach used in object-oriented 3D graphics toolkits such as SGI's Open Inventor product, through which VRML interface support is currently implemented within ParaDiMe cf. [115, ch. 9].

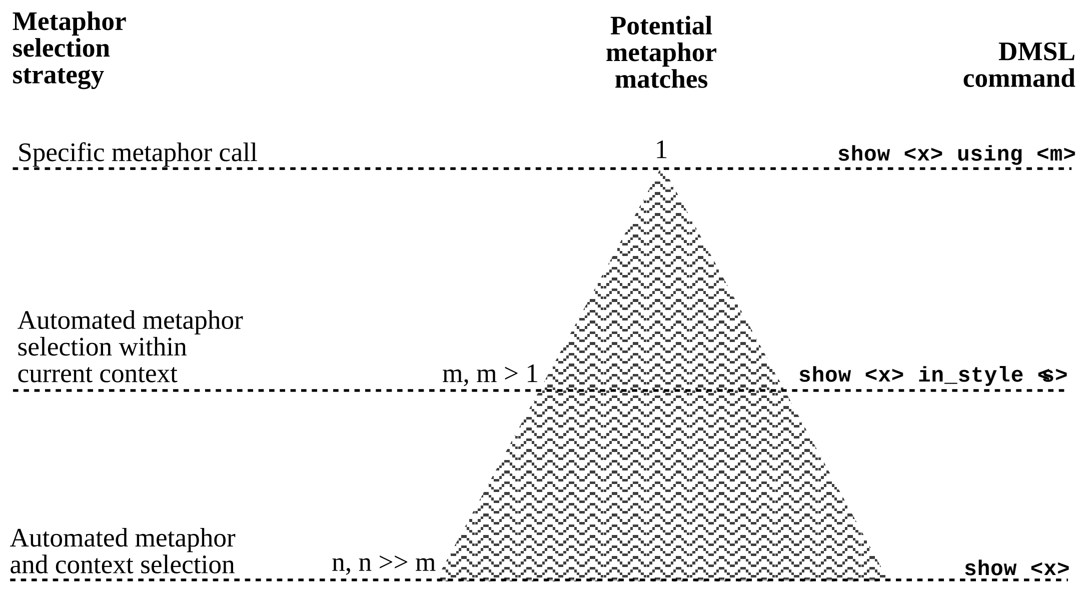

Metaphor selection strategy
Potential metaphor matches
DMSL command
Specific metaphor call
1
show <x> using <m>
Automated metaphor selection within current context
m, m > 1
show <x> in_style <s>
Automated metaphor and context selection
n, n >> m
show <x>

Figure 3.5: Metaphor selection strategies

component and drives the output generator directly. The structure of the command and a usage example are shown below;

```
show <metaphor>(params)
show ProcessWorklist(user=johnsmith)
```

Basic system-initiated interface adaptivity is supported within ParaDiMe through automatic metaphor selection within a specified interaction style. This is used primarily for supporting data (or object) driven application scenarios such as database navigation.[9] The `show in style` command instructs ParaDiMe to match an object (or set of objects) to an appropriate metaphor in a named style. The style parameter is defined at the application level; ParaDiMe has no predefined understanding of the semantics. Rather, the style parameter is used by the metaphor selector to search the *used to display* table to find a metaphor which was defined to be appropriate in the named style at build-time. It is therefore the responsibility of the application designer to make good use of metaphor sets and style keywords. It is useful to support this form of interface adaptivity when several visual representations of the same object are required within the same application. For example, the VSP NEST demonstrator provided navigation and search services over a

[9] Because information access is a central feature of Virtual Working Systems, the ParaDiMe architecture is designed explicitly to support the interactional requirements inherent in these scenarios.

large scale information space characterising research results. For this application, navigation through the graph of related information objects (e.g. projects, people, publications, grants etc.) required several different interaction styles; A *classification browser* style was used to select metaphors which provided a fish-eye view of the information space. The application changes to a *detailed description* style when a user selects a particular object within the broad view. Given the same object, the *detailed* style is used to select a metaphor providing a more comprehensive drill-down view of the object. The structure of the `show in style` command, and a usage example are shown below;

```
show <objectspec> in_style <style>
show find(root.project.nest) in_style 'classification_summary'
show find(root.project.nest) in_style 'detailed_description'
```

The most flexible use of the `show` command automates both metaphor and style selection. This command is not used as frequently as the `show in style` command but is useful in application scenarios where style is unimportant, as long as *any* metaphor is matched. The major use of this type of construct is to support metaphor sets tailored to multiple users or user classes. Here, the same DMSL command will match with different styles and metaphors that have been defined for each class. Application layer components may therefore instruct ParaDiMe via a single command which is interpreted differently for each user class, without needing to be aware of the specific metaphors defined for each class. For example, one user class may define a VRML-based interaction style whilst another may define a HTML-based style. In addition, the general form of the `show` command has been useful is building default metaphors that match with any object or for creating error metaphors in situations where a metaphor cannot be matched. The structure of the general `show` command is shown below;

```
show <objectspec>
show find(root.project.nest)
```

System-initiated interface adaptivity within the current ParaDiMe implementation involves simple constraint satisfaction through a multi-attribute decision based on the contents of the metaphor context table. However, the metaphor selector within ParaDiMe can easily be replaced with a more sophisticated component. As noted earlier, investigative work within the project ruled-out a rule-based implementation for the metaphor selector because of poor performance. However, such advanced approaches could be viable in future implementations, given the improvements in system performance and software since this decision was made.

3.4.2 Interactive operation

The initial implementation of the DiMe architecture supported batch mode interface generation only. That is, a command line interpreter for DMSL was developed that enabled metaphor definitions to be read from the filestore and HTML/VRML interfaces to be written back to the filestore following input of a DMSL `show` command at the terminal. This implementation was found to be useful in generating large numbers of complex but similar interfaces in a single batch operation. For example, a metaphor set was developed for the VSP project which extracted personal information from an organisational directory server; built 3D office representations of that information using VRML; took a 2D snapshot of the scene; created a clickable image map by calculating object locations; and embedded the map into a HTML page which was saved within the filespace of the VSP Web server. This compound operation was too processor intensive to be performed *on the fly*,[10] but could be left running overnight for example to create a large set of complex VRML-based interfaces with low effort.

The interactive ParaDiMe architecture replaces terminal-based control mechanisms with object requests and returns within a distributed computing environment, as shown in Figure 3.6. The current implementation of the architecture uses the WWW Common Gateway Interface (CGI) mechanism [73] to translate between events at the Web user interface (e.g. clicking on a hyperlink) and DMSL commands which are embedded within the interface as structured Uniform Resource Locators (URLs). A CGI program accepts user input and translates this into a remote object request into the application layer, then waits for the resultant output language (HTML or VRML) which is then returned to the browser via standard output in the conventional manner.

[10]Rendering each page took, on average, over 20 seconds of (single) processor time on a Silicon Graphics Power Challenge compute server.

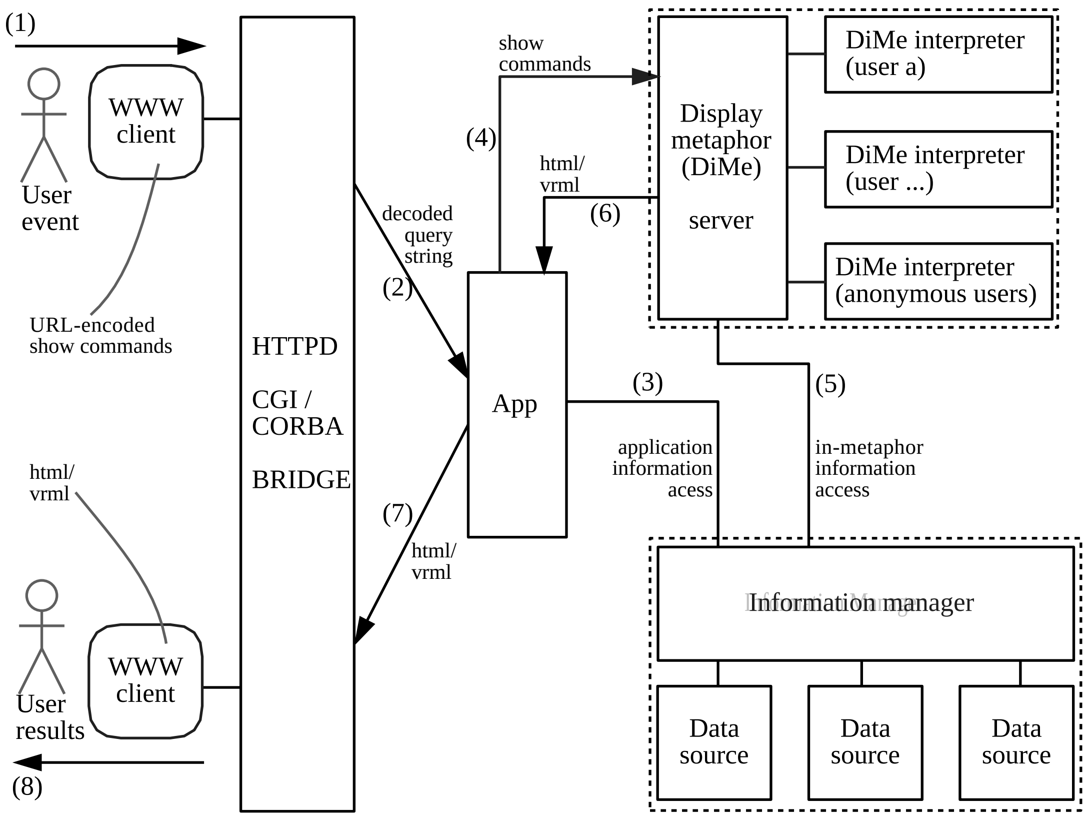


Figure 3.6: Runtime operation of the ParaDiMe architecture

The key to the operation of the interactive ParaDiMe architecture is the use of URL *templates* which are used to embed appropriate DMSL commands within output languages. Because of the stateless nature of HTTP, it is common practise to embed state information in Web pages directly e.g. through URLs or hidden form variables. The use of templates to support interactivity may best be introduced through an example. Consider an organisational directory application within which navigational support is provided by ParaDiMe. One appropriate start page might list all employees with a summary of their contact details on a single line. Upon clicking on the name of a person, the application is to provide a more detailed full-page representation of the directory entry. When generating the index page in this scenario, DMSL commands are embedded as hyperlinks for each name on the page, such that selection of the name invokes a `show` command on the relevant information object in an alternative (detailed) style. Within DMSL, this is achieved through the `Action` syntax introduced earlier;

```
1:    add link1 as HyperTextLink()
2:      Action select view Data
```

```
3:      in "Viewer"
4:      in_style "DetailedPage"
5:      label 'View ' +  Data.name + ' in more detail.'
```

When interpreting the above metaphor fragment for a particular directory entry, a single hyperlink would be generated which embeds a command to be fed back into ParaDiMe (via CGI) upon selection. The form of the URL to be embedded in the output markup language is defined in a set of templates, which for the above example would look similar to that shown below when filled;[11]

```
http://cself21/cgi-bin/Reflector?
username = ms
app = vsp
op = execute
params = show find(root.people.john) in_style 'DetailedPage'
```

In this example `Reflector` is the name of the CGI bridge which is used to redirect CGI-based DMSL commands to a CORBA implementation of the ParaDiMe interpreter. The operation of the runtime ParaDiMe architecture using this approach may now be described as an eight stage process as shown in Figure 3.6;

1. A user event (such as clicking on a HTML link, linked VRML object, imagemap etc) occurs at the Web client;
2. This results in establishment of a HTTP request to a CGI process within the Web server, according to the DMSL embedded command template introduced above. The CGI process extracts a number of parameters from the client request (e.g. user id, DMSL command, session timestamp, form input data etc.) via `post` and `get` mechanisms. This information is then reformulated as parameters to a well known input method on an application component i.e. an application is obliged to support multi-threaded user input events on a well-known IDL interface. The application which receives the input event is located via a name service lookup by the CGI

[11]This is a simplified representation of an actual URL which would, in practise, be encoded using the unicode character set and formulated as a HTTP `post` parameter list.

process. After invoking the input operation on the remote application, the CGI process waits until (string-valued) HTML or VRML is returned via the application.

3. When an application component receives an input event some internal processing or information access may be required before generation of the subsequent Web interface. For example, an embedded `Action` command may result in the execution of a `logoff` operation within the application, before calling DiMe to render a *goodbye* message.

4. Following application-specific processing, the application has two choices for subsequent interface generation through ParaDiMe. Firstly (and most commonly) the DMSL `show` command embedded in the calling link will be passed through to DiMe. Alternatively, the application may decide to pass a different command onto ParaDiMe e.g. a request to view an object is refused because the user does not possess the required level of access and a metaphor reflecting this is generated instead. Each user, or class of user, may be associated with a DMSL interpreter–enabling interface customisation and load balancing of interpreters across machines. By embedding user identifiers within URLs, the appropriate DiMe interpreter may be located.

5. The `show` command is then executed within the appropriate DiMe interpreter and in-metaphor information access constructs are executed if required, resulting in the generation of a stream of HTML or VRML.

6. ParaDiMe then passes the generated output language back to the application (as a string),

7. which passes it back to the CGI process,

8. which passes it back to the Web client,

9. which renders the resultant user interface.

3.5 Supporting access to information services

DiMe was developed primarily to build applications with a major requirement for information search and navigation support. This emphasis had important design implications for the ParaDiMe architecture, which as a result features a close integration of information access features within DMSL and the ParaDiMe runtime environment. Specifically, information access within ParaDiMe is supported through three mechanisms;

an information management service which provides a consistent query interface to heterogeneous underlying data sources using a federation approach;

a client query interface embedded within the ParaDiMe command interpreter which can resolve queries via the information service;

information access constructs used with DMSL metaphor definitions and commands.

The information management service provides a standard interface onto an information space which may comprise of a number of heterogeneous databases and other information sources e.g. relational databases, directory services and distributed objects. As concluded in Chapter 2, several distinct types of information management service are feasible but in this work a simple entity-relationship model is assumed. The information manager federates domain information sources into a single information space which may then be accessed via a uniform interface by applications or presentation services (Figure 3.7).

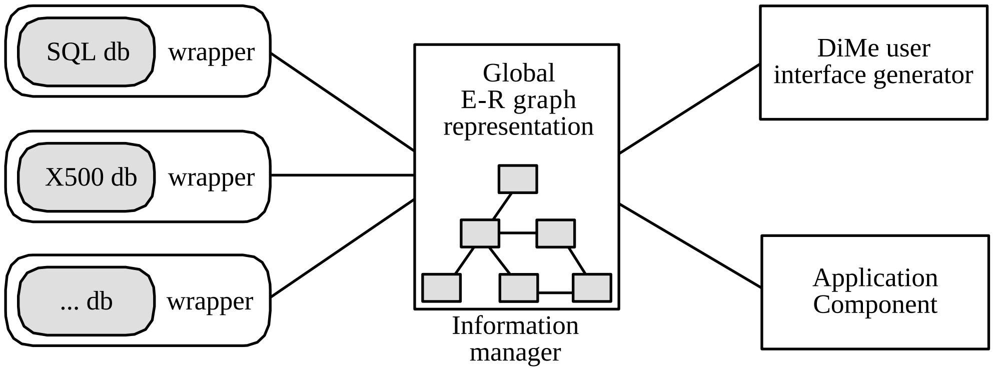


Figure 3.7: Conceptual architecture of the information management service

Federation is achieved through wrappers which map domain information sources into a global conceptual schema and a canonical global namespace. Wrappers are also responsible for mapping navigation and search requests from a generic to a data source specific representation e.g. from the ParaDiMe abstract search interface to SQL via a relational database wrapper. Within the system architecture developed within this work, the information management component provides three specific basic services;

- Federation of heterogeneous information sources,
- Information space navigation via an entity-relationship graph representation,
- Information space query operations.

The basic federation approach adopted within this work is to take a lowest common denominator: it is assumed that individual information sources can be mapped (statically or dynamically) into an abstract entity-relationship (E-R) graph representation. For example, Figure 3.8 shows the E-R model used within the first version of the Virtual Science Park implementation.

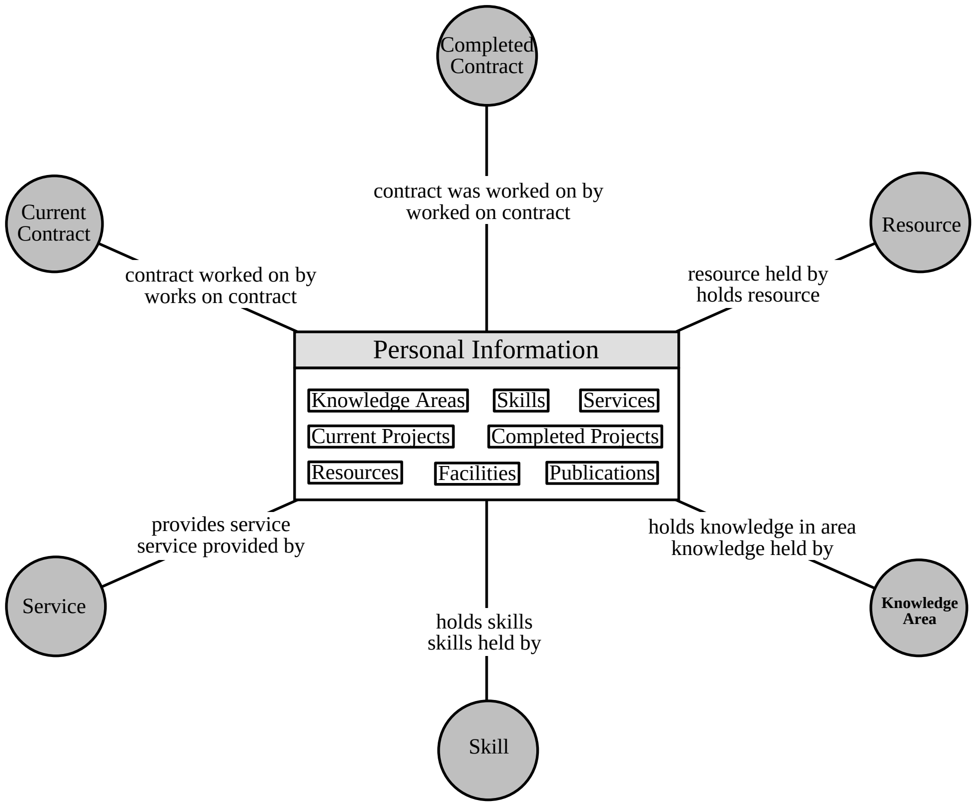


Figure 3.8: Person-centric VSP entity-relationship model

This simple abstract approach is suitable for relational [19] or object oriented [103] modelling. However, query performance can be poor using an abstract representation and it is therefore assumed that implementations utilise efficient physical data models e.g. normalised tables within a database accessed through SQL [2]. As noted earlier, ParaDiMe requires objects within the information space to possess a unique name, a class and a set of name–value attributes. Data access metaphors within ParaDiMe encode knowledge about how to represent particular *attributes* of particular *classes* of object in a particular *context* e.g. how to present a *telephone number* attribute of an object of class *Person* in a *3D office* interactional context. Objects within specific data sources must therefore be mapped to an object type which supports definition and retrieval of these facets in a uniform manner. The approach taken within this work is to ensure that all information objects support retrieval of these attributes, either via internal methods or via dynamically associated object properties. Many underlying information sources may be mapped within the information space using this approach; experimental work within the project successfully derived implementations for four sources;

Object oriented database objects generally map without modification as they already possess identity and class attributes;

Organisational directory objects (e.g. X.500 objects) are already defined as instances of classes and therefore map well. Implicit structural directory links (i.e. that model organisational hierarchy within the naming scheme) should be represented explicitly within implementations using *containment* relations e.g. the University of Leeds `contains` the Department of Computer Studies;

Relational database objects may be mapped to a related object representation at several levels of granularity, e.g. database server, table, view, row, field. At each level of granularity a suitable name and class must be chosen. For example, an object of type `RelationalTable` might be assigned a canonical name of `server1.db2.table3` and possess name-value attributes providing meta-descriptive information about its columns. In the simplest case, a row/column entry can be represented directly as a name-value pair;

Arbitrary distributed objects can be mapped if their implementation can be augmented to provide an appropriate interface e.g. via an object property [87, §13]. In cases where implementations cannot be modified, other object services may be used to provide the relevant interface externally (e.g. relationships or properties).

At the system architectural level, the information management service is viewed as a *query manager* defined in the CORBA query service [87, §11]. A query manager acts as a single point of federation through which queries are evaluated using one or more underlying *query* *evaluator* objects. There is typically a one-to-one mapping between query evaluators and information sources (e.g. one evaluator for relational data, one for organisational directories, one for distributed objects etc.) although in some implementations more than a single evaluator may be required. The query manager is responsible for directing an incoming query to the evaluator(s) that can resolve it. The structure of the information management service is shown in Figure 3.9.

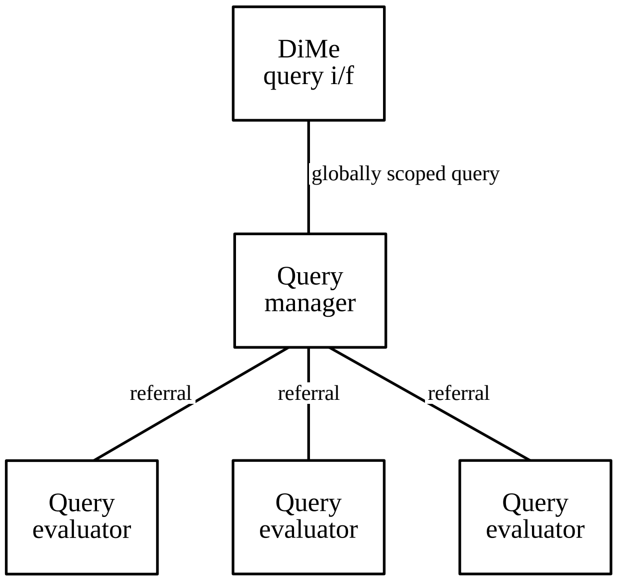


Figure 3.9: Information management service architecture

A universal information namespace is assumed within the system architecture. This namespace serves as the primary mechanism through which query federation is achieved. The naming scheme chosen within this work is a canonical notation based upon textually-delimited syntax used, for example, in the Internet domain name service (DNS) and many file systems. Every object is positioned within a tree structure, creating an object hierarchy which is particularly suited to traversal and manipulation through DiMe. Some example names are shown below;

Object	**Canonical textual name**
Root object	`root`
Organisation object	`root.ldap.vsp`
Person object	`root.ldap.vsp.people.ms`
Relational database object	`root.odbc.db1`
Relational row object	`root.odbc.db1.table3.row8`

The CORBA naming service specification notes weaknesses in using syntactic representations for defining naming contexts [87, §3.1.2], primarily with reference to internationalisation issues. However, such an approach is justified in this architecture because of the need to express object names and contexts within the DiMe scripting language, DMSL. Furthermore, it is trivial to map between the syntactic name representation and a structural representation (as, for example, advocated in the CORBA naming service specification).

Routing of queries to relevant evaluators is performed by the query manager using knowledge of the namespace. Evaluators are connected to the query manager via a registration/callback mechanism. The registration process requires submission of a query evaluator interface stub through which the query manager can issue queries and a naming context over which the evaluator is able/willing to accept queries. For example, a departmental directory service within a VWS implementation may register as a query evaluator for the `root.org.uol` namespace, thereby committing to evaluate queries on all University of Leeds objects. Queries rooted upon objects whose name begins with this context identifier would be referred to the registered evaluator.

Structural inter-object relationships are implied when using a canonical naming scheme e.g. `root.ldap.vws` participates in an implicit *containment* relation with `root.ldap` and `root`. It is therefore important to differentiate between *structural* and *explicit* relationships between objects within the information space. The naming approach utilised within this work was adopted from the VSP project, which for several years utilised an LDAP[12] directory for information management. LDAP (and the X.500 standard [51] upon which it is based) models organisational objects (such as companies, departments and people) according to a tree hierarchy. For this type of data source it is appropriate to view the namespace structure as object relationships. Within the VSP project, structural relationships were augmented with explicit relational modelling constructs using software services layered over the LDAP directory server. In this work, however, it cannot be assumed that all information sources will be of a naturally hierarchical structure. Therefore, the approach taken within this work is to use an explicit relational representation throughout all data sources using an object relationship service. Query evaluator implementations will, of course, typically utilise efficient internal representations for objects and relationships. But these internal representations must be expressed through the object relationship ser-

[12]Lightweight Directory Access Protocol

vice upon externalisation.

As relationships are modelled explicitly within the system architecture there is no technical requirement for a syntactically specified namespace. However, such an approach has been found to be very useful when integrating information sources and working with DMSL scripts. Symbolic names are intuitive to manipulate within DiMe and also provide a simple mechanism to delineate the information space.

The information management service provides an E-R graph representation of the underlying information space through an object relationship service e.g. [87, §9]. Clients of the information service (i.e. DiMe or application layer components) utilise this graph primarily to *navigate* the information space, although support for arbitrary queries is also provided.

Navigational query operations are special forms of query which are rooted at a particular object within the information space and traverse one or more *relational* links to other objects. The root object plus the resultant object set together form a *connected* graph. e.g. "*from the VWS Organisation object, find all objects of class Person linked by an Employs relation.*"

Generalised query operations are scoped over the entire information space and the result set may contain objects that are not explicitly related e.g. "*find objects modified this week.*"

ParaDiMe is designed specifically to support navigational queries within structured information spaces. There are four basic database query operations which ParaDiMe requires from the information management service in order to enable information space navigation;

1. Retrieve an object by name,
2. From an object, find all related objects of any class,
3. From an object, find all related objects of a particular class,
4. From an object, find all objects linked by a specific relational role.

Object retrieval by name is required through the `find()` syntax within ParaDiMe. Typically this syntax is used within commands that present *start pages* for information navigation e.g. within the VSP implementation `show find('root.vsp')` generates the root index to the VSP information space. The other use for object retrieval by name within ParaDiMe is in metaphors that act as search handlers. A text input form element can be linked to a metaphor that attempts to locate (and display) the object named in the entry within the information space e.g.

```
define viewobject extends null {
  add target = 'root.default'
  set this to_contain {
    add result as find(&target);
  }
}
show viewobject(target='root.org.vws');
```

Many ParaDiMe metaphors contain embedded search constructs which operate upon all objects related to a named (or the current) object, irrespective of class. For example, a `Project` object within the VSP information model may be linked via a `has related resources` relationship to objects of type `Person`, `Expertise` or `Publication`. The reserved `Any` token is used to match against any class of object so, using the `Project` object example, the following construct would result in the execution of a query to find all objects related to a specified object;

```
add related_objects as set_of find('root.vsp.project.101'->Any);
```

Execution of this query through the information management service may produce a set of heterogeneous objects (i.e. objects of different classes). The metaphor selector component within ParaDiMe is responsible for choosing an appropriate metaphor for each object in the result set, or selecting an error handling metaphor in the event that an adequate selection cannot be made.

No commitment is made at the system architectural level as to how the information management service is to be implemented. However, as noted earlier, a distributed object architecture providing component services with the functionality of CORBAservices is assumed. There are several relevant common object services upon which the information management service is reliant, although how these services are actually constructed is an implementation issue;

A Query service provides operations to retrieve and update objects within collections such as databases [87, §11],

A Relationship service enables explicit modelling of typed relationships between objects and support for graphs of such relationships [87, §9],

A Property service enables objects to be dynamically associated with named, typed values outside the static type system [87, §13].

Within the CORBA query service, SQL and OQL [18] may be used as query languages. In this work, however, most queries are of a navigational nature and may involve different underlying information sources. It was beyond the scope of the work to implement a fully functional query evaluator for a potentially complex and heterogeneously-structured information space. A simple proprietary query syntax was therefore developed within the project. The syntax enables navigational searches within the information space to be specified textually according to the following syntax;

```
(find <from> <direction> <objectfilter> <linkfilter> <genfilter>)
```

from: the canonical name of the object within the information space at which the navigation is rooted (mandatory parameter),

direction: the direction(s) along which to traverse object relations (mandatory parameter),

objectfilter: a restriction on the class of objects to be returned (optional parameter),

linkfilter: a restriction on the object relationships to traverse (optional parameter),

genfilter: a source specific general search filter which is understood by the target query evaluator (optional parameter).

There are several potential values for the mandatory *direction* parameter, based upon a structural view of the information space. However, navigation requirements in most implementations can be met by four search directions;

one-level-down: follow a relation from source role to target role (e.g. from employing-organisation to employed-person),

one-level-up: follow an inverse relation from target role to source role (e.g. from employed-person to employing-organisation),

one-level-up-and-down the union of a one-level-up and one-level-down search result set,

all-levels-down perform a one-level-down search and (recursively) perform another one-level-down search on each element in the result set.

All-levels-down searches may generate very large result sets and are not recommended in implementations unless they are rooted towards leaf-nodes of the information space graph. Some typical search syntax examples are shown below;

Search		**Syntax**
1	Which people are directly associated with the VSP tenancy?	`(find root.org.vsp onedown personobject * *)`
2	Which tenancy is Neil Hunter a member of?	`(find root.people.neilh oneup * employedbylink *)`
3	What are all the services that VSP members offer?	`(find root.org.vsp alldown serviceobject * *)`
4	What organisational directory entries contain the string 'VWS'?	`(find root.ldap * * * (description=='*VWS*'))`

Examples 1–3 show onedown, oneup and alldown navigational searches respectively. A oneupanddown search merges the results of a onedown and oneup search. Example 4 shows how queries on proprietary information sources may be accommodated via the `genfilter` parameter. In this case, an LDAP search filter is passed to a query evaluator managing access to an organisational directory.

The definition of a simple textual query language specialised for navigational queries is beneficial in this work for several reasons; it closely reflects the abstract entity-relationship structure of the information space; construction of query evaluators for different information sources is uncomplicated; the syntax maps well to symbolic knowledge representation and query languages such as KIF/KQML, enabling integration of value-added information services. However, in large-scale production systems it would be more appropriate to support standard query languages such as SQL or OQL.

3.6 Supporting forms-based information processing

Within many information-oriented application scenarios, there is a basic requirement for user interaction via forms-based interfaces. ParaDiMe provides a simple mechanism for supporting this style of interaction through the use of forms-handling metaphor definitions and reformulated DMSL commands which convert form input elements into metaphor parameters.

A HTML form associates names and values with form elements such as text boxes, buttons and other input fields. Upon form submission, these elements are encoded (either using HTTP `get` or `set`) and passed to a CGI process named within the form. DMSL metaphors can generate HTML forms through exactly the same mechanisms used to generate other types of markup. The key to forms generation (and subsequent processing) within ParaDiMe is that meta-information about how to route the form upon submission is embedded within the form as hidden values. A simple example of a forms generation metaphor is shown below (taken from the VSP User Interface metaphor set—VSPUI);

```
define TestForm extends VSPUIDocument {
  set page to_contain {
    add form as VSPUITemplateForm(
                   cgi_handler="Reflector",
                   method="POST",
                   username=&userid,
                   metaphor_to_use="TestHandler");
    add check as VSPUIFormInputRadio(
                    name="radio1",
                    value="YES",
                    checked="Y");
  }
}
```

In this example, which defines a form with a single visible radio button input, the `VSPUITemplateForm` metaphor embeds several pieces of routing information within the generated form. Firstly, a CGI program is associated with the form submission button (line 4) and a CGI method[13] (line 5). Secondly, a ParaDiMe user name is associated with the metaphor in order to route the processing request to the correct interpreter within ParaDiMe (line 6). The user id may be specified as a literal string but is more usually specified as *the current user* which is represented in DMSL through a special token—`&userid`. Finally, a handler metaphor is specified for the form (line 7). The handler metaphor must define a variable slot for each input element in form.

Upon form submission at the user interface, the form data is passed through CGI (and the object bridge) to the application named as the form handler. The application then reads the submitted form values on standard input and performs any required processing e.g. object updating within the information space. When application specific processing is completed, the application synthesises a DMSL `show` command from the form input element values and the meta-information embedded by the generating metaphor. The structure of the synthesised command, together with a usage example, is shown below;

```
show metaphor_to_use(element=value, element=value...)
show TestHandler(radio1='YES')
```

The `show` command is then routed to the relevant ParaDiMe interpreter and the output results returned to the client in the standard manner. The forms handling subsystem within ParaDiMe provides a useful mechanism for implementing form-based interaction within applications. However, care must be taken by system builders to ensure that form generation metaphors remain consistent with their associated handlers. In order to address this issue, early experimental work has attempted to automate definition of forms generation and handler DMSL metaphor definitions directly from database schemata. This provides a quick way to generate accurate forms metaphors but creates a further level of indirection in the interface generation process i.e. generation of a model to generate an interface. This approach has been useful for applications where the information model remains static, but a better strategy would be to develop constructs which enable new metaphors to be synthesised dynamically when new objects are encountered. This strategy, which should be explored in future work, could move DiMe towards becoming a *generic object browser* in dynamic information environments.

[13]It is normally safer to use `post` rather than `get`.

3.7 Supporting access to remote objects

In addition to information access, a basic application requirement in building integrated CSCW systems is access to functional services e.g. conference control managers, security servers or document management systems. Provided such services have a visible control API, or can be wrapped in an appropriate interface, then access to control methods may be supported within ParaDiMe and DMSL. ParaDiMe is implemented using the CORBA distributed object architecture and support for access to functional services within DiMe is consequently based upon the remote object request approach, as shown in Figure 3.10;

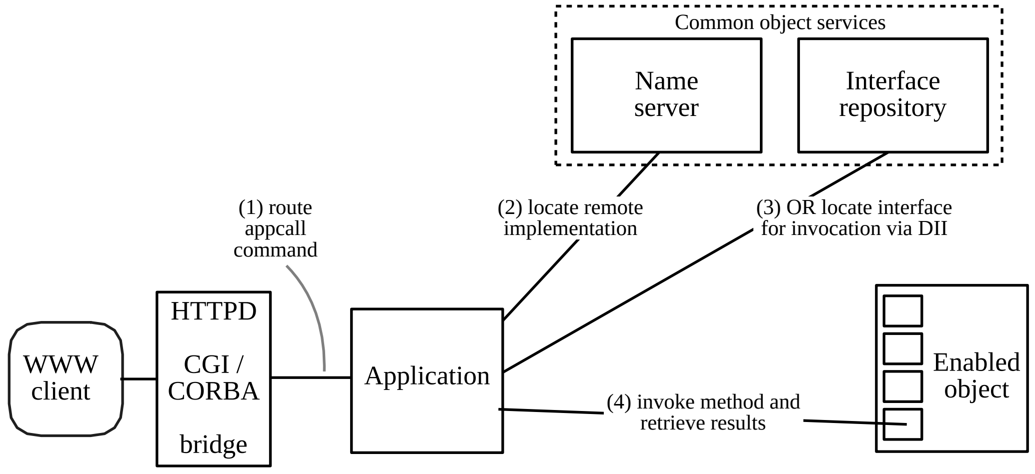

Common object services
Name server
Interface repository
(1) route appcall command
(2) locate remote implementation
(3) OR locate interface for invocation via DII
WWW client
HTTPD
CGI / CORBA
bridge
Application
(4) invoke method and retrieve results
Enabled object

Figure 3.10: Remote method invocation within DiMe

Remote method invocation is supported within DMSL through the `Action` syntax, introduced earlier. The `Action` construct was developed to embed a DMSL `show` command within a URL accessed through hyperlink selection (or other interface event). This mechanism has been extended in this work to support remote method invocation. For example, the DMSL fragment shown below generates a hyperlink that, upon selection, will invoke a `logout` method within a remote security service. Within DMSL, a remote object and method is firstly specified using the `app call` construct (line 3). A list of method specific (stringified) parameters may also be provided (line 4).

```
1:    add logout as HTLink
2:                    Action select
3:                    app_call "security.logout"
4:                    params &userid
```

```
5:                      label "Sign off from the system"
```

At runtime, the object bridge initially routes the request to the application component specified within the `Action` template (step 1 in Figure 3.10). The application then uses the CORBA name service to bind to the service implementation identified within the metaphor definition (e.g. the security service implementation) (step 2). An interface repository lookup may also be required where an interface is not known to the application *a priori* (step 3). After binding to the remote interface, the application may then locate and invoke the method named within the `app call` definition (step 4). Following method invocation, an application is responsible for calling ParaDiMe to generate an appropriate resultant interface—where the `app call` construct is used in existing applications, ParaDiMe generates a simple "*operation performed successfully*" message.

3.8 Supporting access to collaborative tools

The major application of remote method invocation mechanisms within ParaDiMe is in supporting execution and subsequent control and teardown of collaborative working tools e.g. shared whiteboards, videoconferencing tools etc. as shown in Figure 3.11.

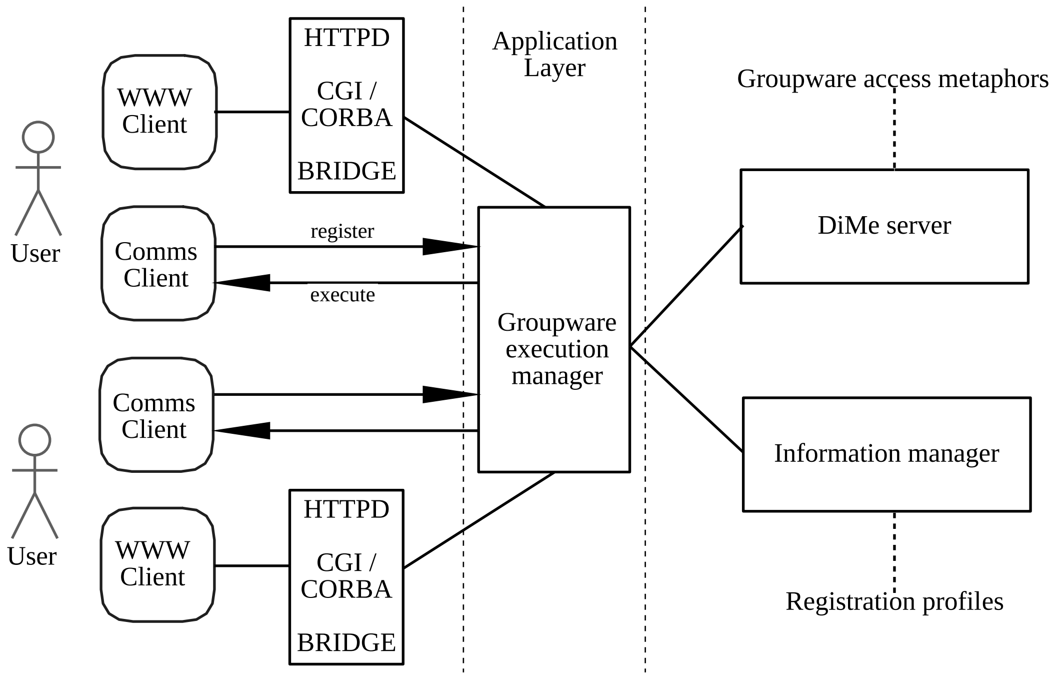


Figure 3.11: Collaborative tool control subsystem

Groupware session control is, however, differentiated from the majority of other uses for remote method invocation within ParaDiMe by a requirement for control of *client-side* functional components. For example, consider establishment of a shared whiteboard session between two users of a ParaDiMe-based group support system. In order to support this scenario, ParaDiMe must firstly be informed that a user wishing to engage in communication is logged in and willing to accept connections from other users. Secondly, when a calling user wishes to establish a whiteboard connection with somebody else, ParaDiMe requires a mechanism for executing client-side whiteboard tools on each client machine and *pointing* the clients at each other.

The approach taken within this work was to develop a standalone client-side component that is separate to the Web browser through which the user typically interacts. The communications client registers a callback interface with a remote conference control component, which is then used to launch and control collaborative tools. As noted earlier in the thesis, the flexibility of this mechanism is ultimately constrained by the level of external control available in particular tools. In a proof-of-concept implementation for example, the Unix `wb` shared whiteboard tool was integrated as this provided a simple command line control interface. Future work may, however, investigate more sophisticated conference control schemes such as the VWS conferencing architecture proposed by Hunter [48].

3.9 Application development methodology

ParaDiMe is useful in building integrated internet-based CSCW systems. Development of such applications within ParaDiMe is most appropriately progressed through an iterative Rapid Application Development (RAD) approach, as shown in Figure 3.12. However, designing and building ParaDiMe applications requires different development emphasis through the design-build-evaluate lifecycle in comparison to conventional software engineering projects. As a development environment, ParaDiMe includes a standard library of re-usable CSCW services and generic metaphor definitions which describe how access to those services may be provided via a Web user interface. An initial system design and build activity using ParaDiMe therefore starts with an architectural design activity in which core services are identified and brought together. Once a set of general services has been brought together, a controlling application may be constructed using a standard template class provided with DiMe. This template (expressed as a set of CORBA IDL interfaces) specifies methods which a control application must support in order to operate with the CGI-object gateway and the ParaDiMe runtime interface generator.

Following an initial build of a controlling application, the set of application wide metaphors can then be built from basic predefined metaphor definitions provided with DiMe. Application wide metaphors are typically developed to mirror application information models and describe the general *look-and-feel* of the entire application. After definition of application generic metaphors, user interface requirements for specific users or user classes may then be translated into specific metaphor sets. These sets of metaphors typically extend generic interface components using the object oriented inheritance features of DMSL. At this point in the initial system built, an operational system may be provided to users or their representatives in order to gain feedback on the system. This evaluative process may result in a *major iterative* system build requiring application level code changes. However, once operational functionality has correctly been implemented at the application level, subsequent change requests from users are typically focused upon the human-computer interface. Such changes may *quickly* be enacted within the ParaDiMe runtime architecture through a *minor iterative build* as shown in the diagram. This activity may be carried out in *real time*, perhaps during an prototype evaluation meeting between system builders and user representatives.

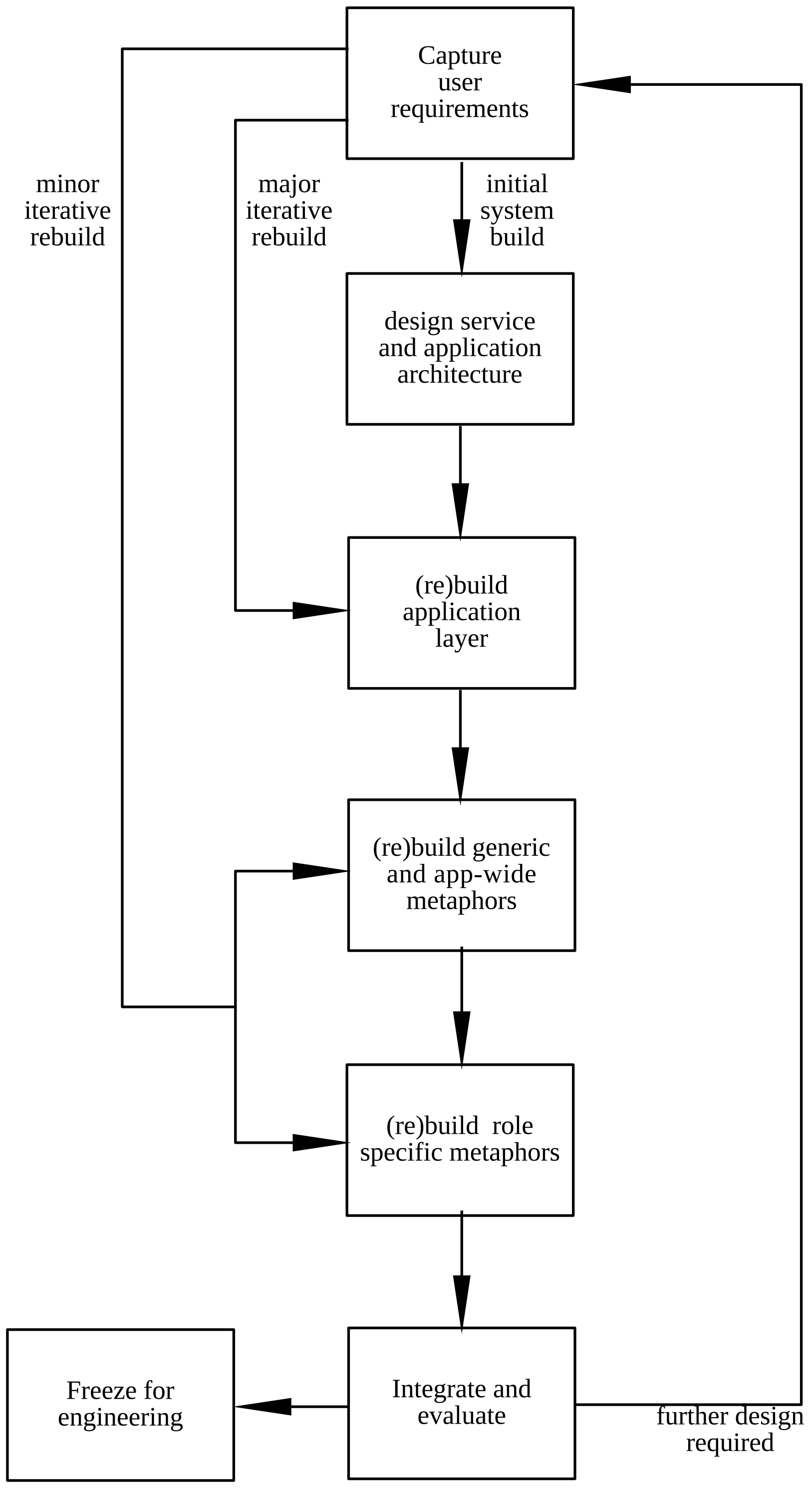


Figure 3.12: ParaDiMe development methodology

3.10 Summary

This chapter has discussed the ParaDiMe model-based CSCW architecture, which attempts to raise the level of abstraction at which integrated internet CSCW applications are built and subsequently modified over time. ParaDiMe provides explicit architectural support for building cooperative systems via component-based models which describe user interaction with functional services. The ParaDiMe architecture is comprised of four major components;

A library of user interface models is defined using a purpose-built modelling language which describes how system users interact with application and component services;

A runtime interface generation service provides application user interfaces at runtime from user interface models defined using the modelling language, directed by commands from the application-specific control component;

A library of CSCW services provides implementations of CSCW services that are build specifically to enable their re-use in different applications;

Application components implement application specific functionality and integrate access to general CSCW services where required.

The Display Metaphor Scripting Language (DMSL) was introduced as the modelling language component of the ParaDiMe architecture. The relationship between DMSL, the DiMe object model and the ParaDiMe runtime interface generation component was discussed. The runtime architecture subsystems supporting user interactivity were described. Finally, a simple methodology for developing ParaDiMe applications within a RAD framework was proposed. Chapter 4 proceeds by describing the application of ParaDiMe towards a case study cooperative working scenario. Specifically, it presents a proof-of-concept ParaDiMe-based implementation of an application to support group work within a telecommunications business process.

Chapter 4

Case study implementation

4.1 Introduction

Chapter 3 described the system architecture of a model-based architecture for construction of integrated CSCW systems from modular and reusable components. As stated in the introduction to this thesis, the architecture seeks specifically to address three problems inherent in many existing CSCW system construction environments,

1. CSCW services are not reusable,
2. User perspectives are not adequately reflected during prototyping,
3. Evolving user requirements are not adequately reflected in evolving live systems.

In order to assess the research hypothesis that a model-based system construction approach can offer a solution to these problems, a proof-of-concept implementation of the model-based architecture was constructed. The implementation was then applied to a case study cooperative working scenario and a number of use cases developed in order to demonstrate the utility of the approach towards solving the research problems. A practical demonstration of the architecture based around these use cases formed the basis for assessment of the research with a potential user community within BT Laboratories, as reported in Chapter 5.

This chapter presents the proof-of-concept implementation of the ParaDiMe model-based CSCW architecture, focused upon a concrete case study business process within BT described in Section 4.2. Whilst it was beyond the scope of this PhD research to implement a comprehensive support environment for the entire process (because of it's complexity), it was felt that the important features of the research could be assessed with respect to smaller elements of the process. The aim was to create tractable problem scenarios to which the research implementation could be applied whilst retaining realistic characteristics of the business process domain. Following description of the case study business process, Section 4.3 then identifies and discusses the particular features of the process to which ParaDiMe was applied. The construction of the proof-of-concept implementation supporting these features is then presented in Section 4.6. The chapter concludes with a discussion of four specific use cases for the proof-of-concept implementation, designed to demonstrate salient functionality of the model-based architecture for research assessment.

4.2 Case study scenario

This section introduces a *data network service quotation* activity as an example of a dynamic cooperative business process within a typical large telecommunications enterprise. This process has already been utilised as a case study scenario within existing BT research projects (ADEPT [53] and KRAFT [39]). This meant that for this work, a large and detailed body of existing descriptive material could be drawn upon without the need to perform field-based requirements analysis work. To protect commercial confidentiality, the quotation process as described here is a sanitised version of that which is actually enacted.[1] However, the salient features of the process with respect to this research remain valid.

Most major business customers of telecommunications enterprises rely heavily upon information systems to support their operations. In situations where these operations are performed over several geographic locations there is an increasing market demand for telecoms services to connect distributed information systems. One class of service offered by telecommunications companies to meet this need is *data services*, which carry digi-

[1]The scenario description is based upon unpublished reports and models from the ADEPT and KRAFT projects.

tal traffic between end-user locations over public networks with some quality-of-service (QoS) guarantee. Although these services are delivered over infrastructure shared with other customers it appears to the customer that they have their own private network, but without the expense of maintenance. Such arrangements are often referred to as *virtual private networks* (VPNs). Data service requirements vary greatly from job to job, dependent upon many constraining factors e.g. bandwidth, latency, availability, length of circuit, routing etc. Providing a quotation for such services requires resource coordination through several related activities within the telecommunications company, as shown in Figure 4.1.

The end-to-end quotation process is comprised of several ordered tasks, performed by a number of cooperating participant roles. The process is currently managed using a conventional workflow management system (WFMS). Figure 4.1 shows the control flow of work through the end-to-end quote process.

A job is initiated when a customer approaches the telecoms company for a quotation. In the first instance, **customer details are captured** by a customer handling centre and entered into the workflow system via a forms-based terminal interface. If the customer has not dealt with the telco before then a new customer details record is created, otherwise the existing customer details record is located. Submission of customer details to the workflow system then triggers the next phase of the quotation process.

Following submission of a customer details record, two activities are progressed in parallel. Because of the high potential cost of data service provision, the **customer may be vetted** through a credit reference agency. Whilst this check takes place, an account manager is assigned to **elicit user requirements**. Representatives are typically assigned by market sector and have good industry and portfolio knowledge. The account manager liaises with the customer and builds up a more detailed picture of their requirements. These are entered into the workflow environment via a forms interface and associated with the customer details record.

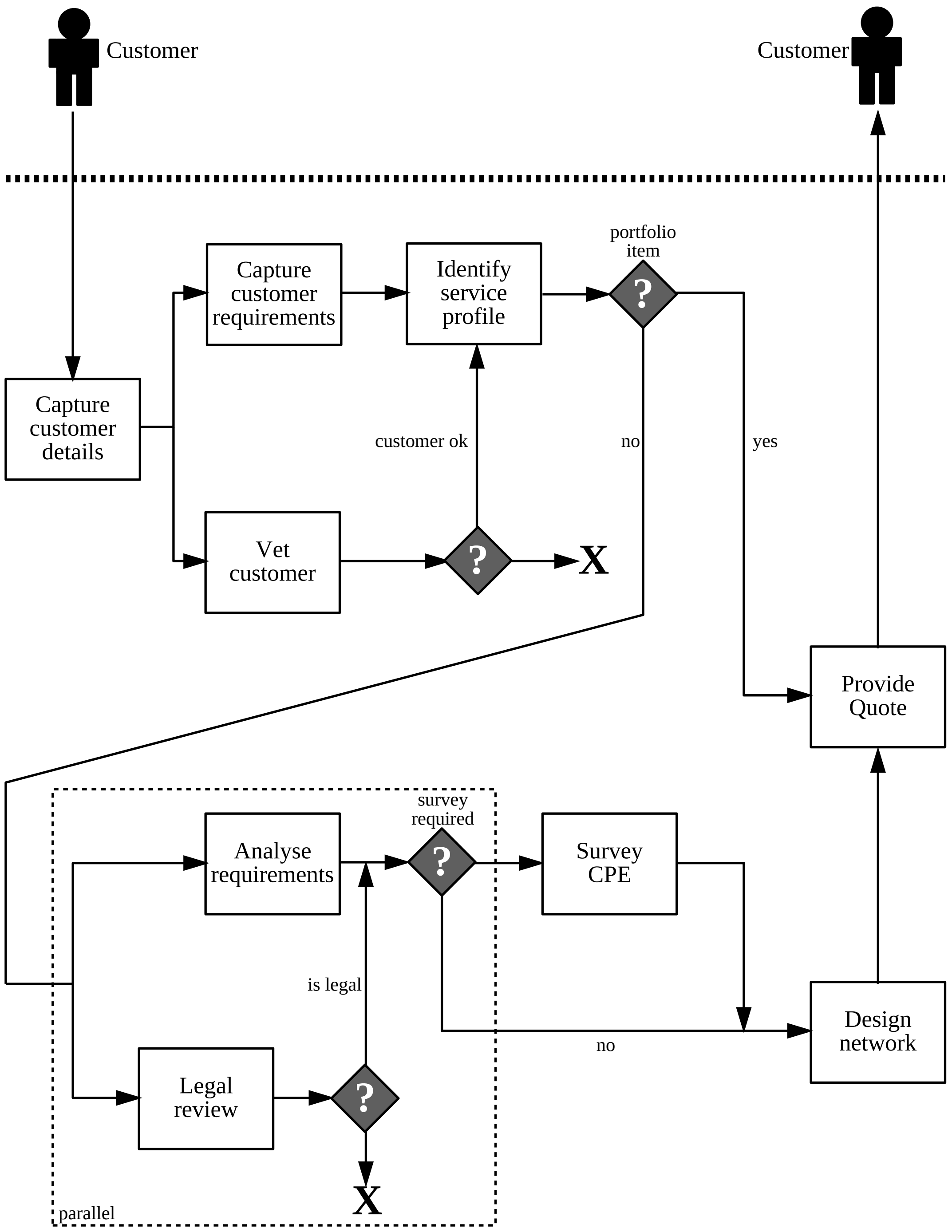


Figure 4.1: Provision of quotations for data network services

Providing the customer has been vetted successfully, **the service requirements profile is identified**. Here, the customer requirements are analysed to assess whether they can be met from the existing telco service portfolio i.e. standard off-the-shelf services. If this is the case, a quotation can be initiated immediately and forwarded to the customer—thereby terminating the quotation job.

If the customer requirements cannot be met from an existing portfolio service, a *data network design* must be created in order to provide a customised solution. At this point, a network design team is assigned to the job and detailed network analysis and design is conducted. Whilst this is underway, a parallel review is conducted by a legal team to ensure that the emerging design does not contravene relevant legislation (it is, for example, apparently illegal to carry encrypted network traffic through some parts of Europe). During the design stage, the account manager mediates between the customer and the various cooperating groups within the telco (e.g. design team, legal advisors and technical support). If the parallel secondary requirements analysis and legal review is successful, a detailed design stage is initiated.

In the **detailed network design** activity, the design team are responsible for producing a customised network architecture that meets the customer requirements. If it is deemed necessary during the secondary requirements analysis phase, the customer premises equipment (CPE) is surveyed on-site by engineers to provide input into the detailed network design activity.

Upon initiation of the detailed design activity, the design team transfer the customer details and requirements records from the main workflow system to a (textual) design database, controlled by an account manager. During the design activity, the design team collaborate using a range of heterogeneous information sources and tools. A designer will, for example, require access to information on geographical site location, protocols, vendor equipment, design guidelines, network performance characteristics etc. Throughout the design process, the account manager elicits further requirements from the customer as needed and feeds back progress on the design. Because of the complexity of the design activity, there is still significant reliance upon paper-based communication and process coordination at this stage.

Once the design team have formulated a bespoke network design for the customer, a profile of that design is fed back into the main workflow management environment and a customised quotation prepared. The **quotation is then delivered** back to the customer and the process terminates. Related installation, testing and billing processes are subsequently enacted if the customer accepts the quotation.

4.3 End-user perspectives and application stakeholders

Provision of quotations for data network services is an involved process requiring structured and ad-hoc collaboration between a number of different groups within the telco. Collaboration occurs within a highly heterogeneous information and systems environment. The inherent complexity of the process (although simplified here) is reflected in the variety of systems used to support the real process. Cooperating groups often utilise internal systems from which information is transferred to the major workflow environment as and when required.

Several distinct roles must cooperate in order to provide a customer quotation (e.g. customer handling centre operative, account manager, network designer and legal advisor). These roles have different skills, expertise, responsibilities and perspectives on the overall process. These differences are reflected in different requirements from the systems used to coordinate the process;

the customer handling clerk handles initial enquiries (e.g. in a telesales environment) and enters customer details onto a form, submission of which triggers the quotation process. No further functionality is required of the support environment for this role.

the account manager the account manager is responsible for a managing a number of jobs and mediating between customers and the assigned technical teams. They are primarily interested in tracking end-to-end job progress and are unconcerned with the highly technical content of the network design.

the network designer utilises a wide range of tools and information sources to create a bespoke network design. The design team need to share information and collaborate internally, whilst liaising with the account manager and legal advisor as the design progresses.

the legal advisor uses a summary representation of the proposed design to assess potential legal ramifications. In making this assessment, they typically communicate on an ad-hoc basis with the design group during the secondary requirements analysis phase.

One reason that different groups within the *real* process support environment utilise internal work management systems is that a single conventional workflow management system (WFMS) cannot *easily* meet contrasting per-role requirements.[2] Facilities exist within many workflow systems to build customised interfaces for particular classes of users, but the development effort required to support this heterogeneity over time as requirements dynamically evolve is often prohibitive (unless support environment customisation adds significant value in process enactment). When constructing the process-support application, as with any system, end-user requirements are of central importance. The major *target users* in this research are those responsible for creating and subsequently managing the environment which most appropriately meets the needs of their end-users. As noted earlier, this work seeks specifically to consider *dynamic* applications domains in which changing requirements must be fed back into support systems in an effective manner. A number of scenarios within the data services quotation process exemplify this. For instance, a new networking technology may be introduced which changes the structure of the telco product portfolio; cooperating groups may relocate to different office locations leading to a changing requirements for communications tools; a new (similar) service might be introduced using the existing process as a template. Within the simplified business process support scenario considered here, three major stakeholders may be identified;

- Process designer
- System manager
- End user role (or representative)

[2]This observation is not unique to data network quotation provision of course; the process as described for this work is fairly generic and similar situations arise in many business processes within typical large enterprises.

The process designer is responsible for analysing business requirements and creating a coordinated set of activities through which the business requirements may be met e.g. the designer models the quotation provision process as shown in Figure 4.1. The system manager is then responsible for creating and managing a system-based enactment environment from this business process specification which reflects end user requirements e.g. the system manager builds a workflow system to enact the process definition created by the process designer.

4.4 Case study implementation requirements

The case study implementation attempted to demonstrate the utility of mechanisms within the model-based architecture in supporting participatory design and subsequent evolutionary maintenance of a realistic cooperative system. The goal of the implementation was to enable assessment of the research according to the objectives stated in Chapter 1. Hence, the implementation did not seek to deliver a comprehensive process-support environment that could be used by end-users in a pilot study. Rather, the objective was to enable exploration and experimental assessment of the key features of the model-based approach to cooperative systems prototyping and evolution.

The remainder of this chapter describes the development of the ParaDiMe-based prototyping environment to support experimental assessment of the model-based approach, according to the simple development method introduced in Section 3. Section 4.5 firstly discusses the information and process-oriented user requirements for the application. Section 4.6 then describes how these requirements may be met from reusable CSCW services and application-specific components. Section 4.7 considers the user interface metaphors that are required to support the application and describes the structure of the resulting metaphor library.

4.5 Application information models

The key feature of the application scenario is the requirement to augment (rather than replace) conventional WFMS process coordination services with a more complete process support environment, which more closely reflects differing end-user requirements. This has important design implications for the application, in that a large proportion of the information model is predefined by the existing system. This is a common situation in software prototyping, where a new system is to be developed based upon a similar existing application. ParaDiMe is designed explicitly to support information model flexibility through a tight integration with a general purpose information manager component (described in Section 3.7).

For the quotation process support scenario there is a need to maintain an interface with a conventional WFMS environment, through which jobs are coordinated. In considering existing coordination services, Chapter 2 identified a number of relevant standards that facilitate interoperability between workflow systems. For this case study, the existence of a workflow manager compliant with the WFMC reference architecture is assumed. It is also assumed that the workflow manager within the application domain supports state querying that would allow a *snapshot* model of the business process to be generated when required within the support environment. It was infeasible (and undesirable) to integrate an existing large-scale WFMS implementation within the prototyping environment, so it was assumed that a symbolic model could be passed from a WFMS and the support environment. This approach has the advantage that the effect of changes in business process can be explored quickly through manual introduction of different models into the system.

Through links with existing research work within the IBS group at BT, Process Interchange Format (PIF) was chosen as a process-description interlingua [63]. The goal of PIF is to enable interoperability between workflow systems and so represents a good choice for process modelling within the case-study scenario. The PIF core information model provides a declarative object oriented (frame-based) description framework for business processes, as shown in Figure 4.2.

Instances of the classes shown in Figure 4.2 may be used to make declarative statements about a business process at a particular point in time or over a set of timepoints, enabling precise modelling of statements such as "*activity B should not start until activity A has been completed.*"[3] For this case-study, PIF enables business process information to be used within the support environment. The specific requirement was to encode the

[3]It is beyond the scope of this thesis to describe PIF in detail. The reader is referred to [63] which fully specifies the PIF standard.

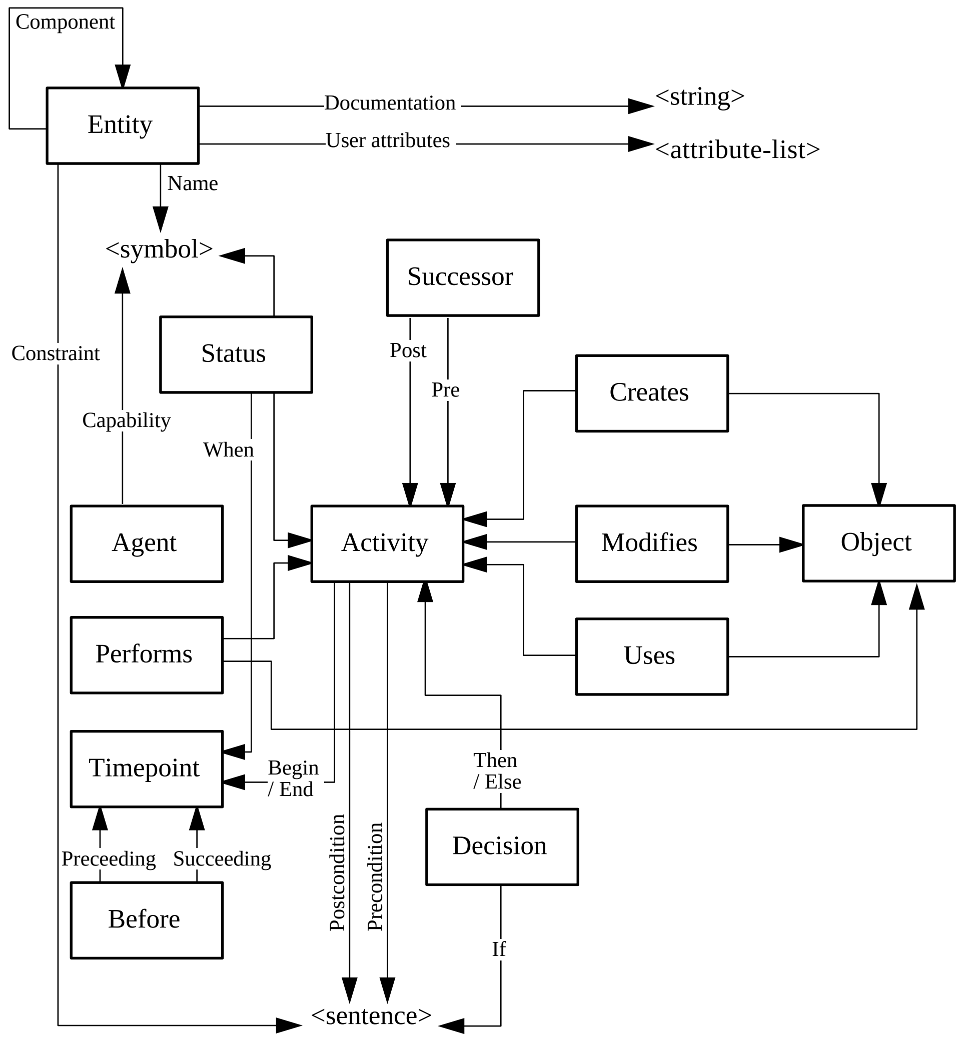


Figure 4.2: Process Interchange Format (PIF) information model

quotation business process (Figure 4.1) in PIF, and then use this description in creating user interfaces for the support environment. However, modeling and implementation of a support environment for the entire quotation process was beyond the scope of the PhD research project.[4] A simpler information model, based on a small subset of PIF augmented with workflow client information was therefore derived (Figure 4.3).

This model adopts the PIF *Activity* class along with *Component* and *Successor* relations to enable characterisation of hierarchical and linear task ordering constraints. Although the model cannot capture decision points and other more esoteric process information, the structure of a wide range of processes can be successfully described using this simple approach. PIF is a process definition interlingua however, and as such is not concerned

[4]Because the PIF parsing tools used to construct the prototype implementation could not handle process decision points at the time the system was built.

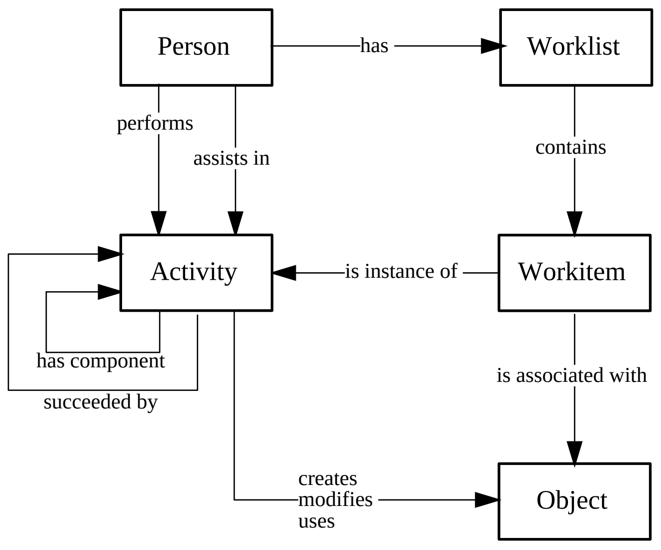


Figure 4.3: Prototype application information model

with workflow *client* information. The end-user roles in the application scenario interact with the WFMS via a client interface however, so the information model for the support application must consequently reflect this. Again, a simple subset of a workflow client information model was derived; A *person* interacts with the WFMS via a *worklist*, which lists a number of *workitems* for completion. A *workitem* represents an instance of a PIF *activity* and is associated with an *information object*.

Several information management assumptions were made in deriving an information model for the application scenario. It was assumed that an information object can hold the artefacts which are created, used or modified at each stage of the business process. Secondly, there is a one-to-one mapping between individual tasks and information objects so, for example, the *capture customer details* task generates a customer details object; the *vet customer* task generates a customer vetting object and so on. In a task sequence, it is assumed that the objects created by preceeding tasks are fed into the succeeding tasks as inputs e.g. the *vet customer* task requires a customer details information object as an input. Finally, it was noted that information within the application domain may be of a complex nature e.g. a network design may be represented as a set of detailed network specifications and graphical topologies. However, within the application it was assumed that all objects can be represented (perhaps by proxy) as simple database objects i.e. a set of named string-valued attributes. An analogous approach was taken within the ADEPT agent-based workflow project, which also used the network service quotation process as a demonstration scenario. Adopting such an approach within this work enabled direct utilisation of the existing ADEPT information models. Because the detailed structure of the information space is not of importance during the prototyping cycle, taking this simplified abstract view of information usage is appropriate. Of course, a production quality implementation would have to address information integration issues within the actual (complex and heterogeneous) information domain.

To summarise, a simple application information model was derived that enables characterisation of the salient features of the scenario, whilst de-emphasising features that are unimportant during prototyping. For the quotation process, the information model serves to describe three types of information upon which prototype user interfaces are dependent;

- structural business process definition (PIF)
- workflow client information (worklists and workitems)
- information use (simple database object for each task)

Following definition of an application information model for the scenario, a proof-of-concept runtime environment was then constructed using ParaDiMe.

4.6 ParaDiMe application implementation

In creating the quotation process support application, there was a need to integrate and control a number of basic CSCW services. The scenario analysis identified that access to a number of basic services were required within the application;[5]

WFMS access provides coordination between the tasks contributing towards the end-to-end process;

Information access enables user interaction with the information artefacts within the quotation process;

[5]Mirroring the coordination, collaboration and information management elements of CSCW as introduced in Chapter 1.

Conferencing tools enable *ad hoc* communication between process participants.

As noted earlier, the role of the WFMS within the application was simulated through an information model defined using a process definition interchange language (PIF). This meant that the business process model could be treated in the same way as other information sources within the application (e.g. worklists, workitems and associated information objects). Integration of process definition and other information sources within the application was achieved by creating specialised data wrappers for the reusable ParaDiMe information manager runtime component defined in Chapter 3. A parser for the PIF language was used to convert from instances of PIF classes into the ParaDiMe entity-relationship format via a PIF *Queryable Collection* (QC) object.[6] The PIF queryable collection acts as a client to the information manager which is connected to the ParaDiMe interface generator at runtime. In addition to the PIF parser, an LDIF parser and associated queryable collection was built to enable description and integration of other information objects and relationships within the application.[7] The PIF and LDIF queryable collection services were implemented as CORBA objects which support dynamic updates through the query interface. This provided the basic method through which information could be dynamically updated within the application so, for example, a new business process definition could be dynamically introduced through submission of an updated PIF file through the PIF parser subsystem.

In addition to WFMS and information access, the case study scenario required the integration of conferencing services within the application. For the quotation process, no changes to individual tools were necessary and so the reusable ParaDiMe conferencing subsystem could be used without modification. For demonstration purposes, the LBL `wb` shared whiteboarding tool was used as a representative example of a synchronous groupware tool. This tool was chosen primarily as it supports a simple command line execution interface which can readily be integrated within ParaDiMe. Synchronous audiovisual conferencing services could be provided through other mbone tools (e.g. `vat` and `nv`) using the same mechanism. The quality of these tools is sufficient for demonstration of concepts during prototyping but a more sophisticated conferencing subsystem would be required

[6]The PIF parser classes were written by Simon Thompson at BT Laboratories and released for use within this PhD project.

[7]LDIF is a simple textual representation for information objects and their attributes, typically used to populate LDAP directory servers.

within a production environment (e.g. the VWS conferencing architecture proposed in [48]).

Following identification of basic service requirements within the application scenario, reusable service implementations must be integrated within a controlling application-layer component. This component implements the control logic within the application, acting as a single sink for all user interface operations (via encoded DMSL commands) and subsequent user interface generation through ParaDiMe. The application was customised using the generic ParaDiMe runtime architecture presented in Section 3. In implementing a support application for the quotation process, shown in Figure 4.4, two changes to the generic ParaDiMe architecture were made. Firstly, queryable collections for the PIF and LDIF domain information sources were constructed. Secondly, a controlling component (called the *workflow helper*) was written as a specialised implementation of the general ParaDiMe application template.

The *workflow helper* component brings together the reusable services that are required within the application, providing three main services within the runtime environment;

Information object update through forms: users interact with tasks and information objects via forms interfaces, thus there is a requirement to map form information back into the information space. This was implemented using the forms-interaction ParaDiMe subsystem.

Conferencing subsystem control: the ParaDiMe conference control subsystem was integrated into the workflow helper application to enable business process participants to collaborate in real-time. Collaboration requirements were modelled within the domain information model, by associating secondary roles with activities using an *assists* relation. For example, collaboration between an account manager role and the network design role during performance of the design activity could be specified by association of the account manager to the design activity via the *assists* relation. ParaDiMe can then use this information to create user interfaces with appropriately embedded communications requests. The workflow helper was used to maintain a list of registered communication client interfaces (i.e. noting which roles were available for conferencing), and executing tools via these interfaces upon reception of a conferencing request from a user.

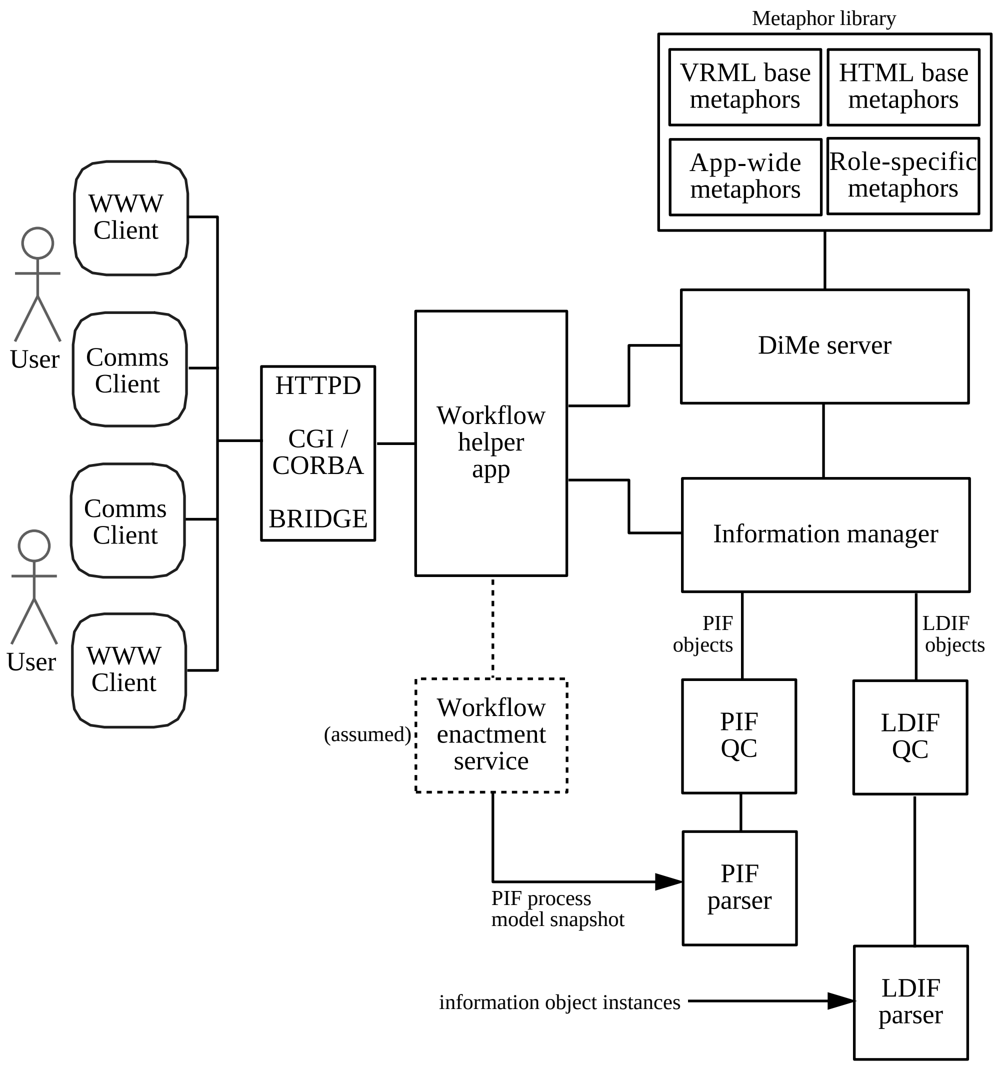


Figure 4.4: ParaDiMe 'workflow helper' application implementation

session control: the final operation which was directly coded into the workflow helper application supported system login and logout functions. These operations are mapped to a private management interface within the ParaDiMe server that controls creation and use of individual interpreters for system users.

ParaDiMe is implemented using the CORBA distributed object architecture which provides a high degree of language and platform flexibility. The workflow helper component was implemented using Java and integrates with the rest of the ParaDiMe runtime architecture via interfaces defined in CORBA IDL (Interface Definition Language).

The HTTP-CORBA bridge was implemented as a simple (50 line) C program executed by an Apache Web server through the CGI mechanism. This would be a poor approach to take in a production system because of the high performance overhead associated with CGI process execution. However, the approach was simple to implement and performance proved adequate with a small number of interactive users. A beneficial side-effect of implementation on a Unix platform was that, for the majority of requests, the CGI process could be obtained from the in-memory file system cache rather than requiring a disk read. In a production system running within a heavily-loaded computing environment, however, a more appropriate bridging technique should be sought e.g. use of a CORBA Web server or compilation of a CORBA module within an existing Web server.

The HTTP-CORBA bridge calls a single method on a specified application layer component (e.g. the workflow helper). This object request is very simple; the CGI query string is packaged and a resultant user interface specified in HTML or VRML is passed back as a string. Following application-specific processing (e.g. conferencing subsystem control within the workflow helper), the application component invokes ParaDiMe to generate a user interface. This is implemented through a single object request using the `DimeAgent` interface as defined below;

```
module DimeExec {
typedef unsigned long Status;
  interface DimeAgent {
    Status execute(in string userid,
                   in string opcode,
                   in string cmdline,
                   out string result);
    };
};
```

The `execute` method accepts a stringified user name token which identifies a system user within ParaDiMe. Within the ParaDiMe user interface generator, each user (or class of user) may be associated with a unique DMSL interpreter. The `opcode` parameter identifies an operation to be performed within ParaDiMe, pertaining to the DMSL interpreter instance for the specified user. Five types of operation are supported, which use the `cmdline` string to carry parameter information;

Login: create a new interpreter for the named user,

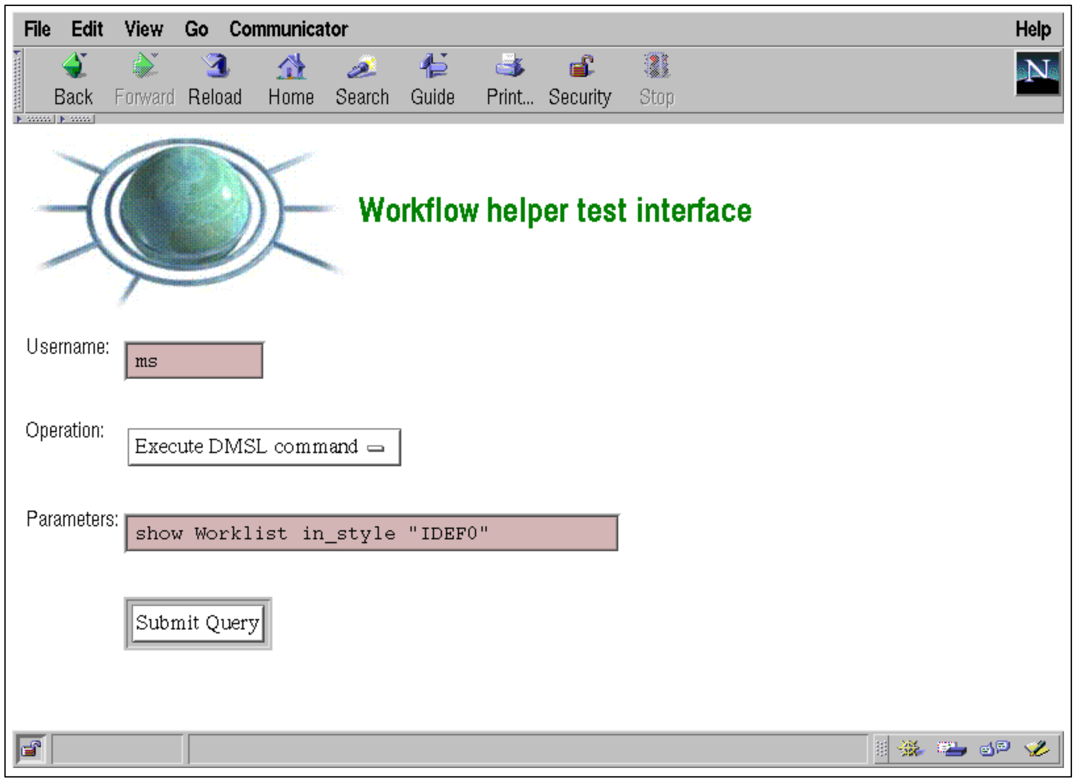
Logout: destroy the interpreter for the named user,

Loadfile: load the specified interpreter with the DMSL metaphor definitions contained in the named file,

Loadbatch: apply loadfile for each of the files specified within the named batch file,

Execute: run the specified DMSL command on the appropriate interpreter.

The result of the above operation calls are fed back to the calling application using a CORBA string holder (`result`). Access to these operations is coded directly into the *workflow helper* application component to provide interactive support for users. However, it was also useful to build a simple ParaDiMe *control console* through which these commands could be constructed manually, as shown in Figure 4.5.



Workflow helper test interface

Username: ms

Operation: Execute DMSL command

Parameters: show Worklist in_style "IDEF0"

Figure 4.5: ParaDiMe prototyping console

This form interface enables a system manager to drive operation of the ParaDiMe runtime environment without having to recompile application code. In the quotation business process scenario, this enables the system manager to make metaphor changes and view the results *in real time*, providing a useful tool in exploring prototype design iteration functionality in review meetings with end-user roles.

The final implementation interfaces to consider within the runtime implementation are the information manager and queryable collections. These are direct implementations of CORBA services, as defined in [87], hence existing applications which utilise this standard may be integrated within the ParaDiMe architecture without modification. The PIF and LDIF queryable collections were implemented using Java, but there is no implicit implementation language restriction provided a CORBA binding is supported.

Following definition of the domain information model and integration of runtime architecture components, the final stage in developing a ParaDiMe application involves definition of the application metaphor library.

4.7 Metaphors within the 'workflow helper' application

Scripted metaphor definitions are the key enabling mechanism through which user interfaces may be developed using the ParaDiMe toolkit. When designing metaphors it is therefore of paramount importance to *design for reuse*, in the assumption that many metaphors used within an application will change as evolving user requirements are fed back into application iterations. Practical experience in building a number of ParaDiMe implementations has identified three important empirical heuristics to observe when creating a metaphor library for an application;

Group functionally-similar generic metaphors together into static modules: many metaphors perform basic services that are required by the majority of applications (e.g. basic HTML and VRML constructs). These should be grouped together into libraries and remain unmodified within applications, other than by inheritance or overriding;

Build upon and create metaphor inheritance hierarchies: the prototyping flexibility of DMSL stems from the object-oriented structure of metaphor definitions. Unless this property is observed when building applications, iteration through metaphor redefinition will not be any quicker than through conventional tools;

Utilise variable slots as metaphor parameters: it is tempting to write metaphors for very specific purposes, but the effort in creating more reusable metaphors that can adapt through variable slot input values is generally repaid through reduced subsequent metaphor coding effort during development iterations.

These heuristics guided development of the metaphor set used within the quotation process scenario, based upon the structure of the ParaDiMe object model presented in Figure 3.3 on page 50. A number of generic and reusable metaphor modules were initially selected. Firstly, base HTML and VRML builtin object wrapper metaphors were specified. A second *value-added* layer of HTML and VRML metaphors was then constructed, mapping to a HTML component library and a VRML *rooms* metaphor library. The HTML component library (called VSPUI[8]) defines HTML user interface widgets representing abstract interface concepts such as menu, search, navigation and query result presentation which are built from simpler builtin HTML constructs. The *rooms* metaphor library provides builtin object support for construction of 3D room interfaces through basic VRML constructs. Following selection of this set of basic metaphors, a hierarchy of metaphor definitions for the quotation process support application was defined, as shown in Figure 4.6. Metaphors were defined within three main tiers, as illustrated in the diagram and described below.

4.7.1 Application base metaphor

A single base metaphor for the application was defined, from which all other application metaphors would directly inherit. Although DMSL does not support multiple inheritance within metaphor definitions, a similar effect may be achieved through careful definition of a single inheritance hierarchy. The application base metaphor was used solely to define a set of variables describing the general look and feel of the application. An extract of the base metaphor definition is shown below (the actual metaphor definition contains around 30 variables used to define application default settings).

[8] Developed within the VSP project.

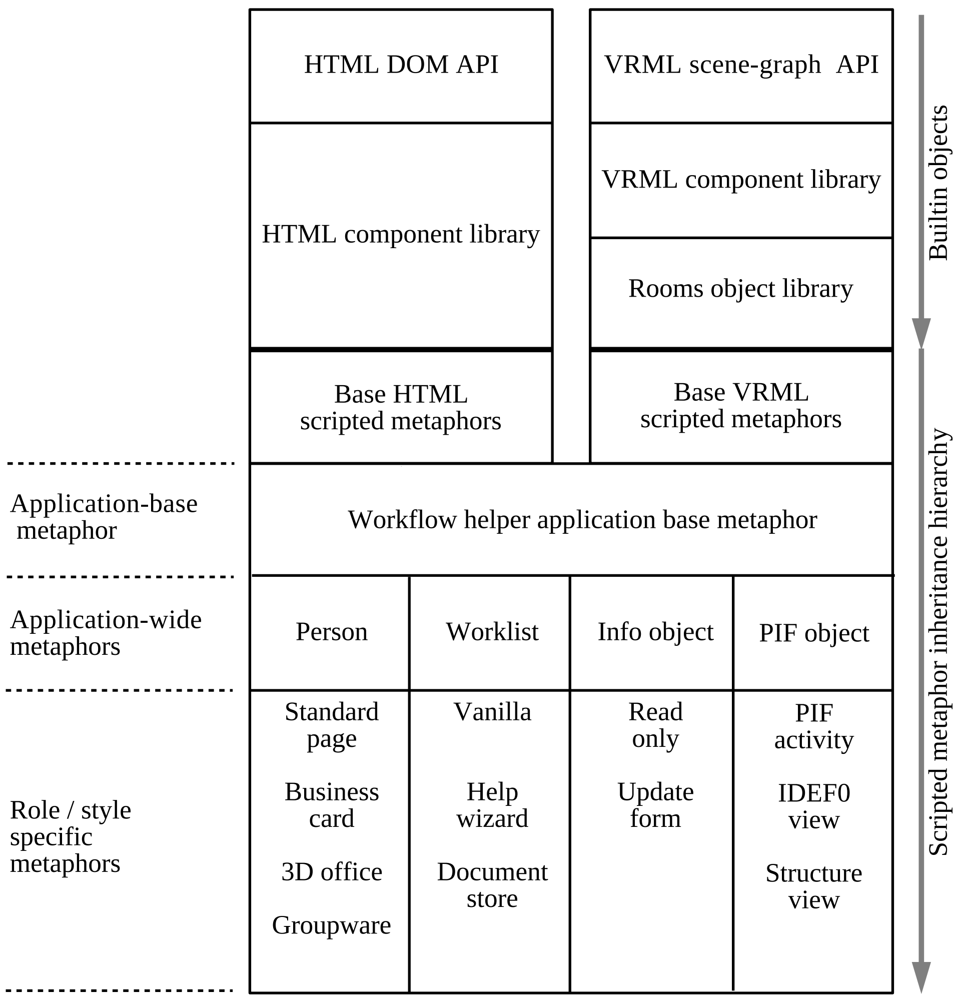


Figure 4.6: 'Workflow helper' metaphors and inheritance hierarchy

```
define QuoteAppBase extends Null {
  add def_font_size = "+1"
  add def_colour="black"
  add def_header_colour="green"
  add def_hotlink_colour="red"
  ...
}
```

Because all application metaphors inherit from this base class, the look-and-feel of interface components can be changed quickly by changing a single metaphor. This approach is similar to the use of style sheets with conventional Web pages.

4.7.2 Information object metaphors

A set of information object metaphors were created using the VSPUI metaphor set, which provided forms-style interfaces to the simple workitem representations defined within the quotation business process. Two duplicate sets of information object metaphors were created. The first set provided a readonly presentational displays for an information object, whilst the second set enabled information object updates to be implemented via the ParaDiMe form interaction subsystem. `Objectclass` and *Style* metaphor context descriptors were used to enable appropriate automated metaphor selection at runtime. For example, the following DMSL fragment extract shows the context description for the metaphor to enable forms editing of a network design proxy object. An example of the HTML form generated from this metaphor, along with a summary of the runtime processes through which the form is handled, is presented in Figure 4.7.

```
define EditND extends QuoteAppBase {
  used_to_display infondObject within Viewer in_style 'editing'
  ...
}
```

4.7.3 Activity metaphors

Two sets of metaphors were defined that enable HTML-based navigation of PIF process definition in different presentational styles. Firstly, a compact representation providing an indented list of task names through which a user could *drill-down* was created (based around a style called StructureStyle). A more sophisticated process navigation metaphor set was then defined to meet the needs of the account manager and process designer roles, who require a more detailed end-to-end view of the business process. A common graphical modelling language for business processes is IDEF0 [82], based upon the SADT modelling technique. This describes processes as sequences of activities which may themselves comprise subordinate activities. An IDEF0 style for presentation of PIF activity objects was constructed, through which the account manager or process designer could navigate the quotation business process model. The metaphors make extensive use of DMSL information access constructs to follow relational links within through the process definition stored as a set of objects within the information manager component. Hence, for each task within the quotation process, the metaphor can also display other useful information (e.g. what information objects are created or updated by the activity, who performs the activity and who indirectly collaborates during it's performance). Variable slot parameters act as switches in selecting a *level-of-detail* for the interface enabling, for instance, a process view just showing who is responsible for performing each stage. The structure of the DMSL definitions used to create the IDEF0 metaphor set are shown in Figure 4.8. Sample HTML output for the StructureStyle and IDEF0 style metaphors is presented in Figure 4.9.

The PIF activity metaphors were designed to operate with arbitrary process descriptions (defined using the simplified activity, component and successor constructs introduced earlier). Hence changing business process characteristics, as signified during application iterations by submission of a modified PIF file, does not require metaphor recoding. The metaphors were also designed to enable their incremental refinement upon subsequent addition of further PIF constructs as shown in Figure 4.2.

4.7.4 Worklist metaphors

Within the case study scenario it was also desirable to provide mechanisms facilitating exploration of alternative representations of worklists. As introduced earlier, worklists provide workflow users with 'to-do' lists of jobs for completion, usually providing some summary information about the job status e.g. the initiator, the type of activity and related tasks or information artefacts. It was identified that different roles within the quotation process could require different worklist representations. For example, a legal advisor might be called into the process quotation on a relatively infrequent basis, hence could benefit from the addition of extra context help within the worklist interface. The network design team need to collaborate over a set of design documents in performing their contribution to the process and could therefore benefit from a document management facility integrated within the workflow client interface. To demonstrate metaphor extension and specialisation through subclassing, a basic *vanilla* worklist metaphor was defined which provides a conventional workflow client user interface (Figure 4.10). Two extended versions of the basic worklist metaphor were then derived by subclassing the vanilla metaphor definition. In the first specialisation, a context sensitive help feature was added. The second specialisation links the worklist to a shared document repository, which could be used by process participants to share information artefacts.

The goal in developing the worklist metaphors was to demonstrate the application of the model-based architecture towards rapid prototyping, in which a number of similar interfaces to the same functional application component may be explored quickly with end users.

4.7.5 Person object and groupware execution metaphors

The final set of metaphors which were built within the workflow helper implementation were used to demonstrate access to collaborative working tools via the groupware control subsystem within the model-based toolkit. Within the case study context, there are several conceivable points at which participant roles need to collaborate in order to perform activities. For example, the network designer and legal advisor collaborate during a design review task to ensure the proposed network infrastructure does not contravene relevant legislation. In tracking jobs through the process chain, the account manager needs to communicate with the staff assigned to the job in order to ascertain progress. The requirements for collaboration were demonstrated within the workflow helper application by extending the business process description language (PIF) to associate communication requirements with activity definitions. A PIF partially-shared view (PSV) module was used to create a new PIF relation, called *assists*, which models the situation where a person helps during an activity but is not directly responsible for *perform*ing that activity. This was defined through extension of the PIF core *performs* relation;

```
(define-frame ASSISTS
:own-slots
((Name "WfhPsv.ASSISTS")
(Subclass-of PERFORMS)
(Documentation "Agent collaborates on activity")))
```

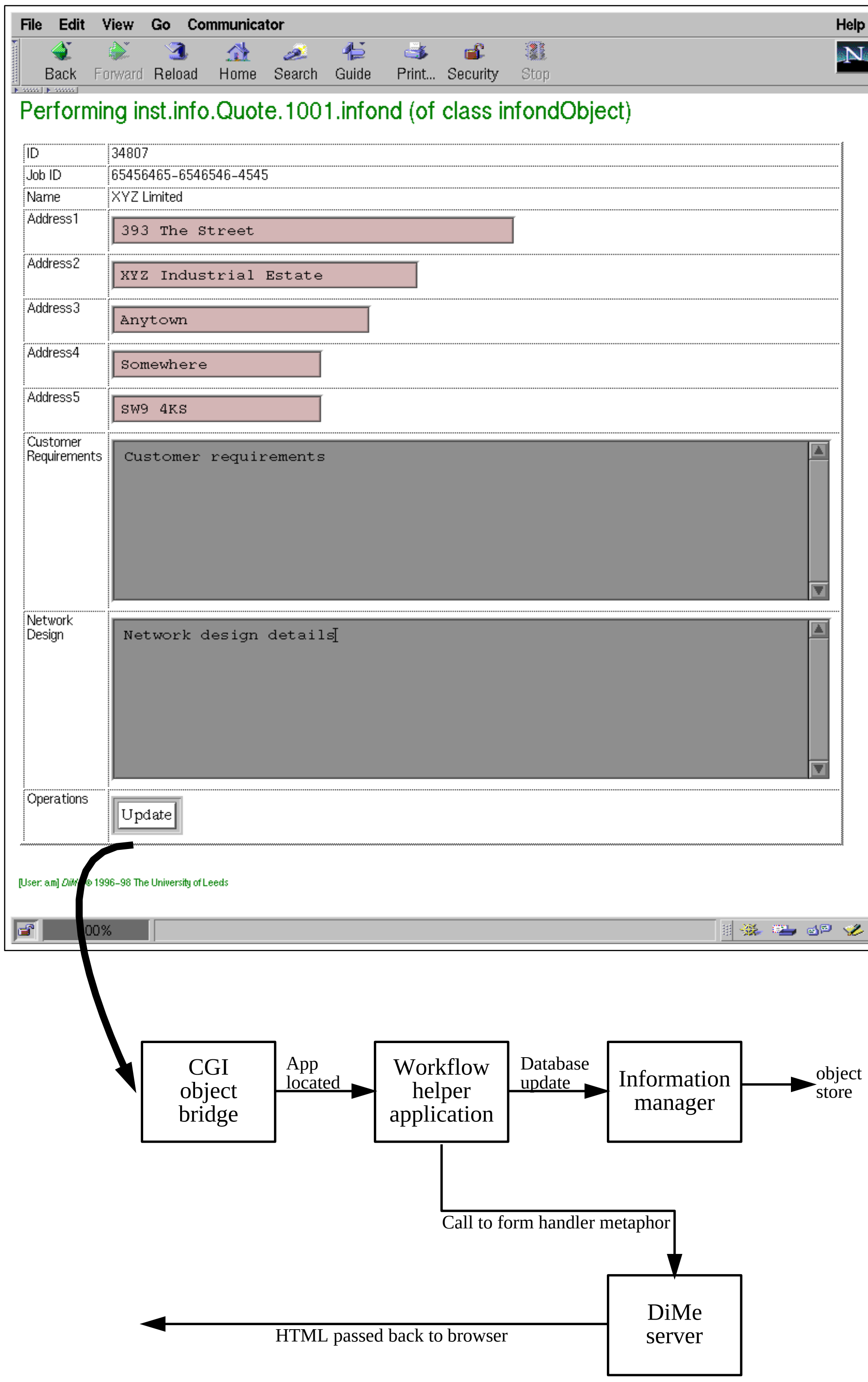
This enabled simple collaboration requirements to be described within PIF files, used to present business process models to the workflow helper application. Two styles of metaphors were used to demonstrate groupware access within the toolkit environment, as shown in Figure 4.11. Firstly, a basic HTML metaphor was built to operate upon information objects describing people within the information space. So, given a set of process participants and an extended process definition including collaboration requirements specified through the *assists* relation, Web pages enabling groupware communications between cooperating roles could be constructed. Secondly, the same functional constructs were embedded within a VRML *3D office* metaphor.[9] Clicking on the computer screen in this interface (as shown in Figure 4.11), establishes a shared whiteboard session with the person whose office is being viewed.[10] Tool execution was implemented through the groupware control subsystem within the ParaDiMe runtime architecture. Access to the LBL wb shared whiteboard tool was used as an exemplar, given that other collaborative tools may be controlled through the same mechanism.

4.8 Summary

This section has presented a proof-of-concept implementation of the model-based cooperative systems construction approach using the ParaDiMe architecture. The goal of the implementation was to enable experimental assessment of the research objectives with a group of potential ParaDiMe users within BT. Thus, Chapter 5 presents a critique of the research through exploration of a number of use cases for the architecture within the case study scenario, driven through the proof-of-concept implementation.

[9]This metaphor was based upon compiled C++ builtin objects written by Thorsten Blaise as his BSc final year project during 1996-97.

[10]It is noted that there is no requirement within the case study scenario for this type of 3D interaction. This part of the demonstration was used primarily to convey the message that the architecture could be used to explore very different interaction styles within applications.

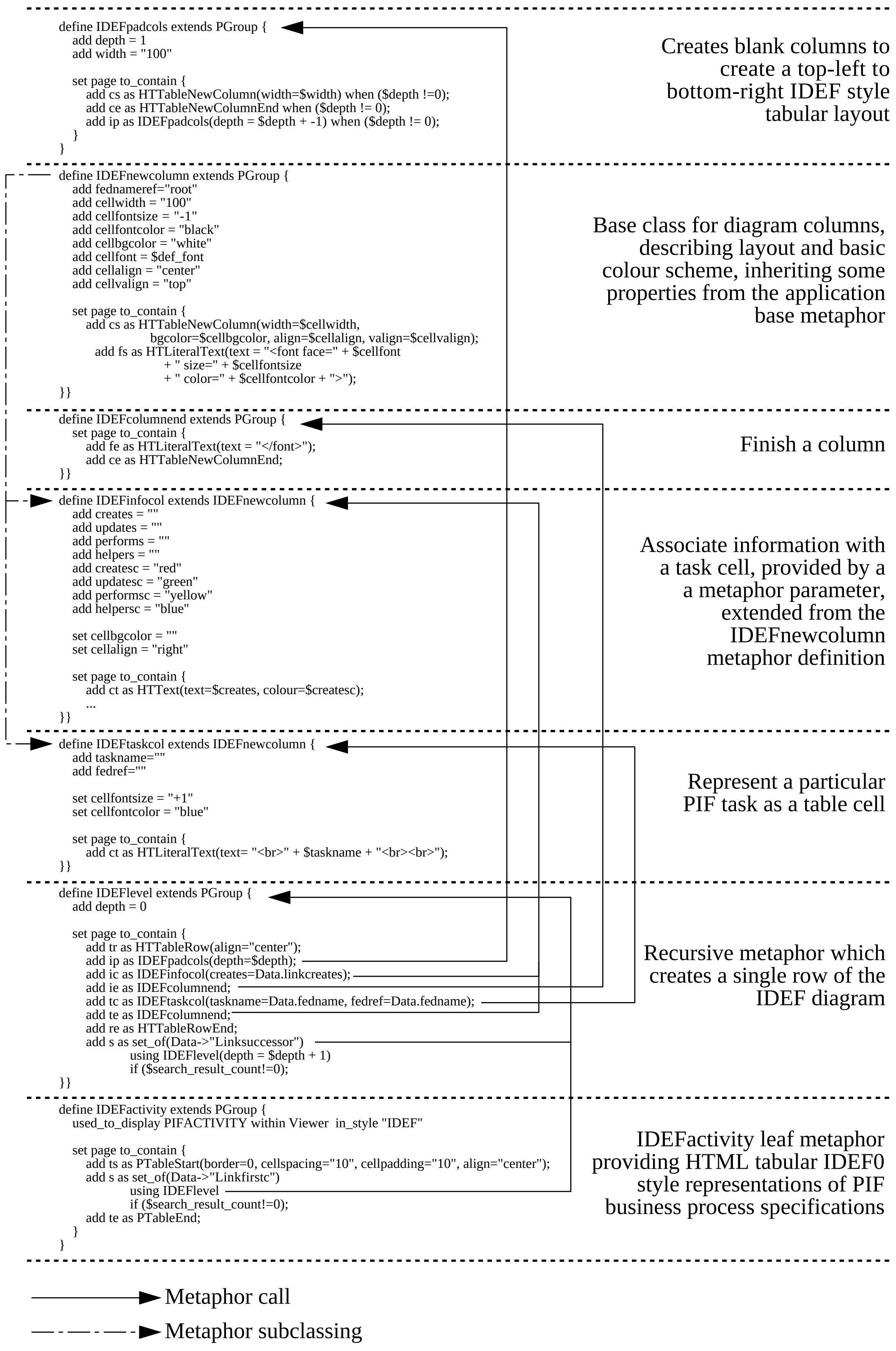


Performing inst.info.Quote.1001.infond (of class infondObject)

ID	34807
Job ID	65456465-6546546-4545
Name	XYZ Limited
Address1	393 The Street
Address2	XYZ Industrial Estate
Address3	Anytown
Address4	Somewhere
Address5	SW9 4KS
Customer Requirements	Customer requirements
Network Design	Network design details
Operations	Update

[User: a.m] *DiM*[illegible] © 1996–98 The University of Leeds

Figure 4.7: Form-based interaction through ‘Workitem’ metaphor



```
define IDEFpadcols extends PGroup {
    add depth = 1
    add width = "100"

    set page to_contain {
        add cs as HTTableNewColumn(width=$width) when ($depth !=0);
        add ce as HTTableNewColumnEnd when ($depth != 0);
        add ip as IDEFpadcols(depth = $depth + -1) when ($depth != 0);
    }
}
```

Creates blank columns to create a top-left to bottom-right IDEF style tabular layout

```
define IDEFnewcolumn extends PGroup {
    add fednameref="root"
    add cellwidth = "100"
    add cellfontsize = "-1"
    add cellfontcolor = "black"
    add cellbgcolor = "white"
    add cellfont = $def_font
    add cellalign = "center"
    add cellvalign = "top"

    set page to_contain {
        add cs as HTTableNewColumn(width=$cellwidth,
                    bgcolor=$cellbgcolor, align=$cellalign, valign=$cellvalign);
          add fs as HTLiteralText(text = "<font face=" + $cellfont
                      + " size=" + $cellfontsize
                      + " color=" + $cellfontcolor + ">");
}}
```

Base class for diagram columns, describing layout and basic colour scheme, inheriting some properties from the application base metaphor

```
define IDEFcolumnend extends PGroup {
    set page to_contain {
        add fe as HTLiteralText(text = "</font>");
        add ce as HTTableNewColumnEnd;
}}
```

Finish a column

```
define IDEFinfocol extends IDEFnewcolumn {
    add creates = ""
    add updates = ""
    add performs = ""
    add helpers = ""
    add createsc = "red"
    add updatesc = "green"
    add performsc = "yellow"
    add helpersc = "blue"

    set cellbgcolor = ""
    set cellalign = "right"

    set page to_contain {
        add ct as HTText(text=$creates, colour=$createsc);
        ...
}}
```

Associate information with a task cell, provided by a a metaphor parameter, extended from the IDEFnewcolumn metaphor definition

```
define IDEFtaskcol extends IDEFnewcolumn {
    add taskname=""
    add fedref=""

    set cellfontsize = "+1"
    set cellfontcolor = "blue"

    set page to_contain {
        add ct as HTLiteralText(text= "<br>" + $taskname + "<br><br>");
}}
```

Represent a particular PIF task as a table cell

```
define IDEFlevel extends PGroup {
    add depth = 0

    set page to_contain {
        add tr as HTTableRow(align="center");
        add ip as IDEFpadcols(depth=$depth);
        add ic as IDEFinfocol(creates=Data.linkcreates);
        add ie as IDEFcolumnend;
        add tc as IDEFtaskcol(taskname=Data.fedname, fedref=Data.fedname);
        add te as IDEFcolumnend;
        add re as HTTableRowEnd;
        add s as set_of(Data->"Linksuccessor")
                using IDEFlevel(depth = $depth + 1)
                if ($search_result_count!=0);
}}
```

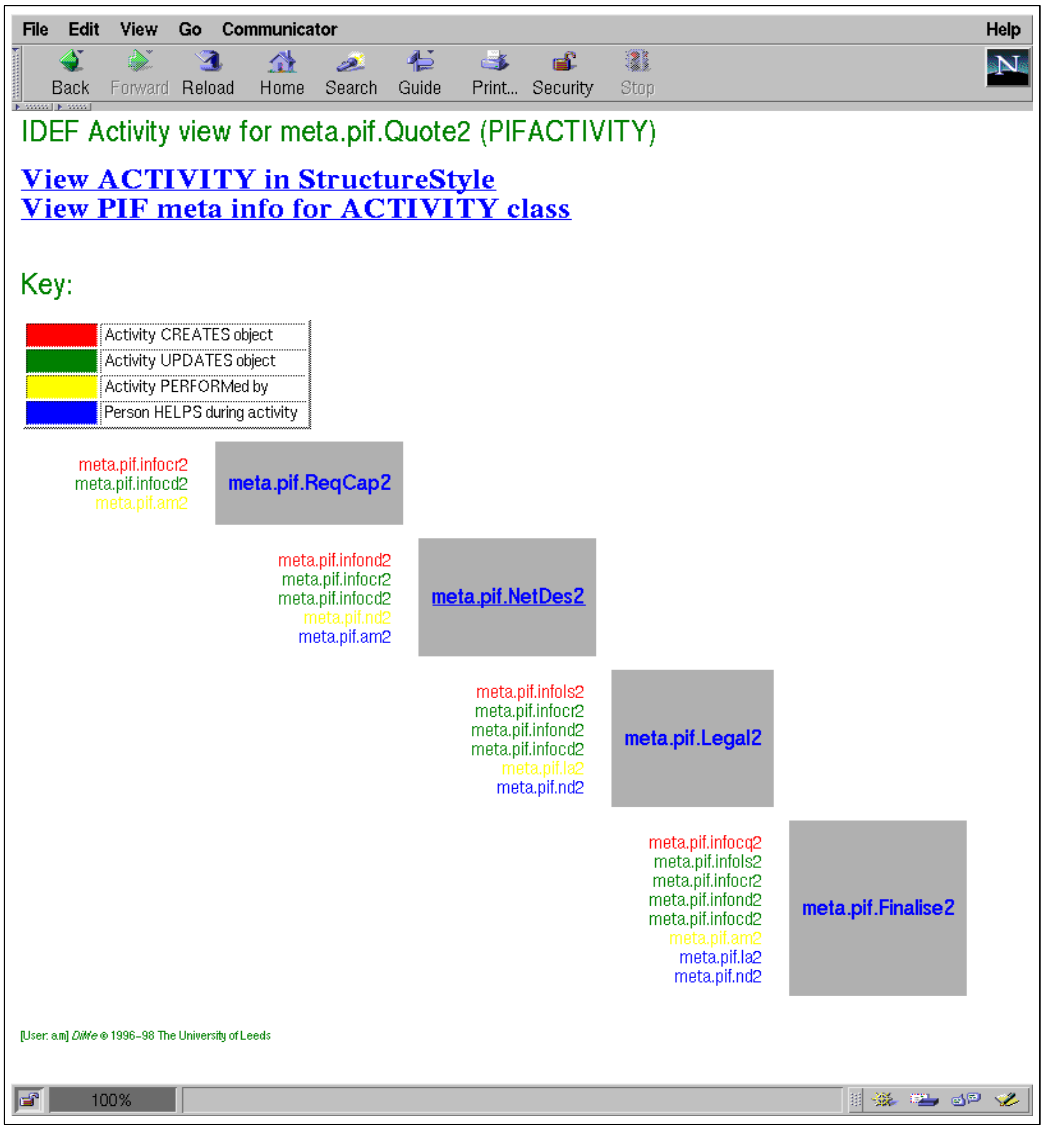
Recursive metaphor which creates a single row of the IDEF diagram

```
define IDEFactivity extends PGroup {
    used_to_display PIFACTIVITY within Viewer  in_style "IDEF"

    set page to_contain {
        add ts as PTableStart(border=0, cellspacing="10", cellpadding="10", align="center");
        add s as set_of(Data->"Linkfirstc")
                using IDEFlevel
                if ($search_result_count!=0);
        add te as PTableEnd;
    }
}
```

IDEFactivity leaf metaphor providing HTML tabular IDEF0 style representations of PIF business process specifications

Figure 4.8: DMSL definitions used to create the IDEF0 metaphor set



IDEF Activity view for meta.pif.Quote2 (PIFACTIVITY)

View ACTIVITY in StructureStyle
View PIF meta info for ACTIVITY class

Key:

🟥	Activity CREATES object
🟩	Activity UPDATES object
🟨	Activity PERFORMed by
🟦	Person HELPS during activity

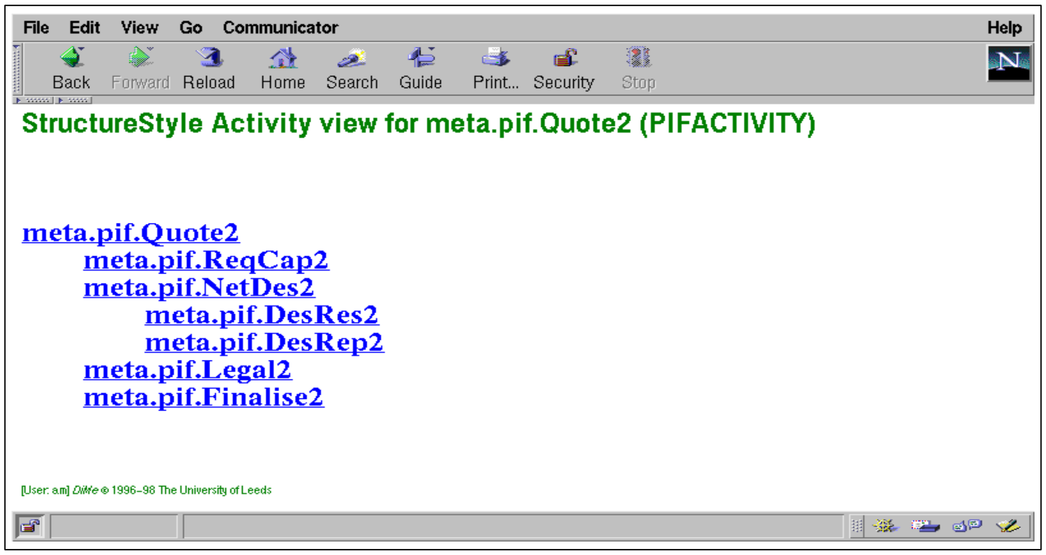
meta.pif.infocr2
meta.pif.infocd2
meta.pif.am2
meta.pif.ReqCap2

meta.pif.infond2
meta.pif.infocr2
meta.pif.infocd2
meta.pif.nd2
meta.pif.am2
meta.pif.NetDes2

meta.pif.infols2
meta.pif.infocr2
meta.pif.infond2
meta.pif.infocd2
meta.pif.la2
meta.pif.nd2
meta.pif.Legal2

meta.pif.infocq2
meta.pif.infols2
meta.pif.infocr2
meta.pif.infond2
meta.pif.infocd2
meta.pif.am2
meta.pif.la2
meta.pif.nd2
meta.pif.Finalise2

[User: am] *DiMe* © 1996–98 The University of Leeds

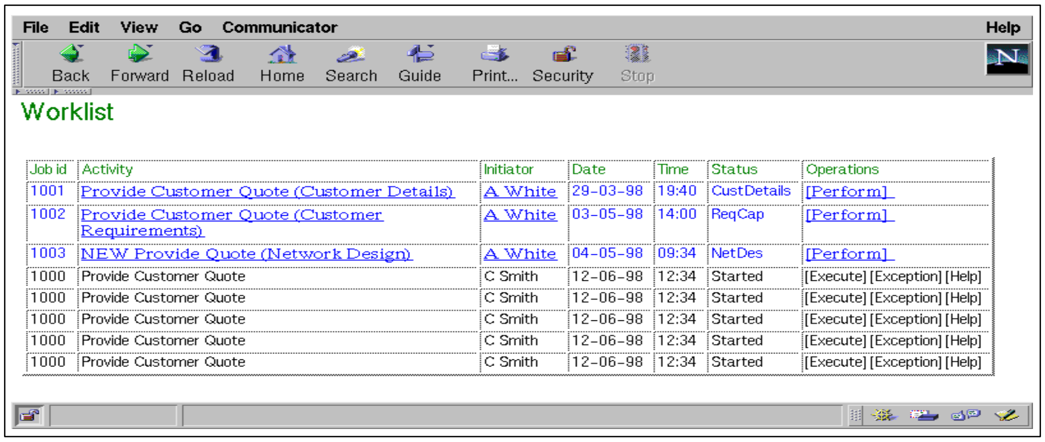


StructureStyle Activity view for meta.pif.Quote2 (PIFACTIVITY)

- meta.pif.Quote2
  - meta.pif.ReqCap2
  - meta.pif.NetDes2
    - meta.pif.DesRes2
    - meta.pif.DesRep2
  - meta.pif.Legal2
  - meta.pif.Finalise2

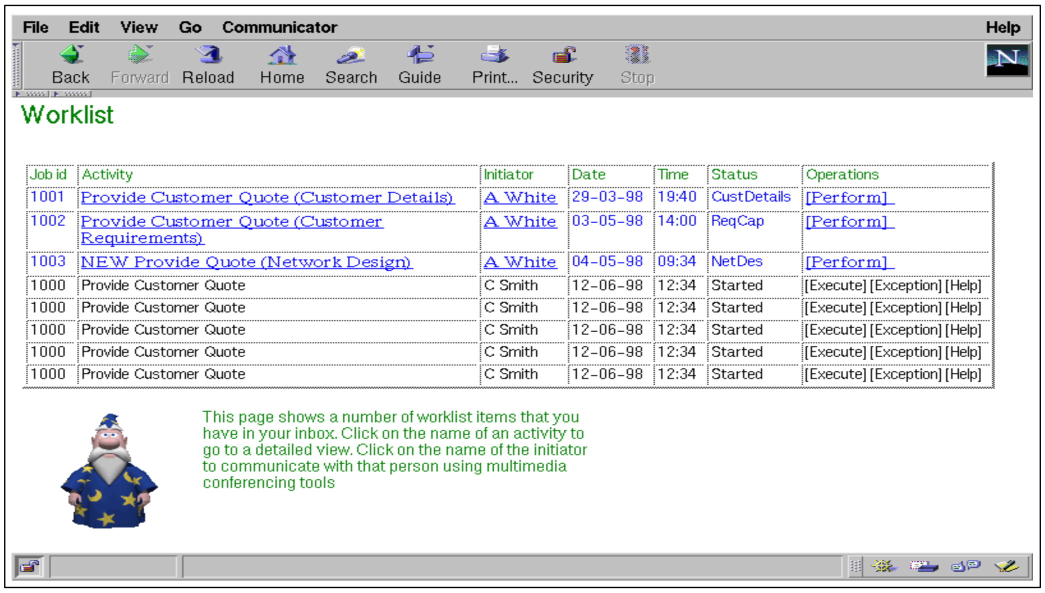
[User: am] *DiMe* © 1996–98 The University of Leeds

Figure 4.9: 'IDEF0' and 'StructureStyle' activity navigation metaphors



Worklist

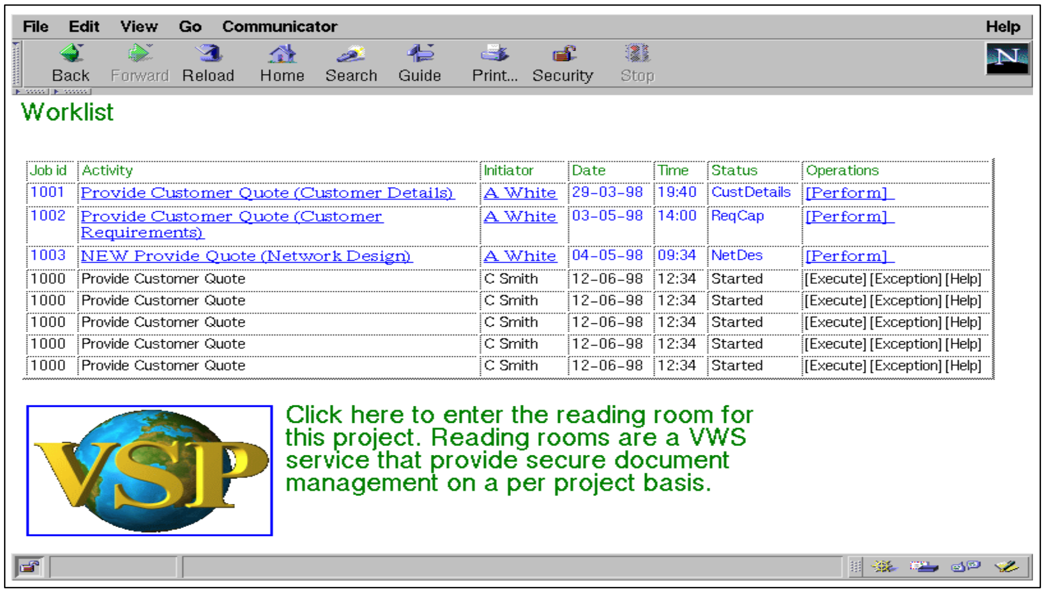
Job id	Activity	Initiator	Date	Time	Status	Operations
1001	Provide Customer Quote (Customer Details)	A White	29-03-98	19:40	CustDetails	[Perform]
1002	Provide Customer Quote (Customer Requirements)	A White	03-05-98	14:00	ReqCap	[Perform]
1003	NEW Provide Quote (Network Design)	A White	04-05-98	09:34	NetDes	[Perform]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]



Worklist

Job id	Activity	Initiator	Date	Time	Status	Operations
1001	Provide Customer Quote (Customer Details)	A White	29-03-98	19:40	CustDetails	[Perform]
1002	Provide Customer Quote (Customer Requirements)	A White	03-05-98	14:00	ReqCap	[Perform]
1003	NEW Provide Quote (Network Design)	A White	04-05-98	09:34	NetDes	[Perform]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]

This page shows a number of worklist items that you have in your inbox. Click on the name of an activity to go to a detailed view. Click on the name of the initiator to communicate with that person using multimedia conferencing tools



Worklist

Job id	Activity	Initiator	Date	Time	Status	Operations
1001	Provide Customer Quote (Customer Details)	A White	29-03-98	19:40	CustDetails	[Perform]
1002	Provide Customer Quote (Customer Requirements)	A White	03-05-98	14:00	ReqCap	[Perform]
1003	NEW Provide Quote (Network Design)	A White	04-05-98	09:34	NetDes	[Perform]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]
1000	Provide Customer Quote	C Smith	12-06-98	12:34	Started	[Execute] [Exception] [Help]

Click here to enter the reading room for this project. Reading rooms are a VWS service that provide secure document management on a per project basis.

Figure 4.10: 'Vanilla,' 'Wizard' and 'Reading-room' worklist metaphors

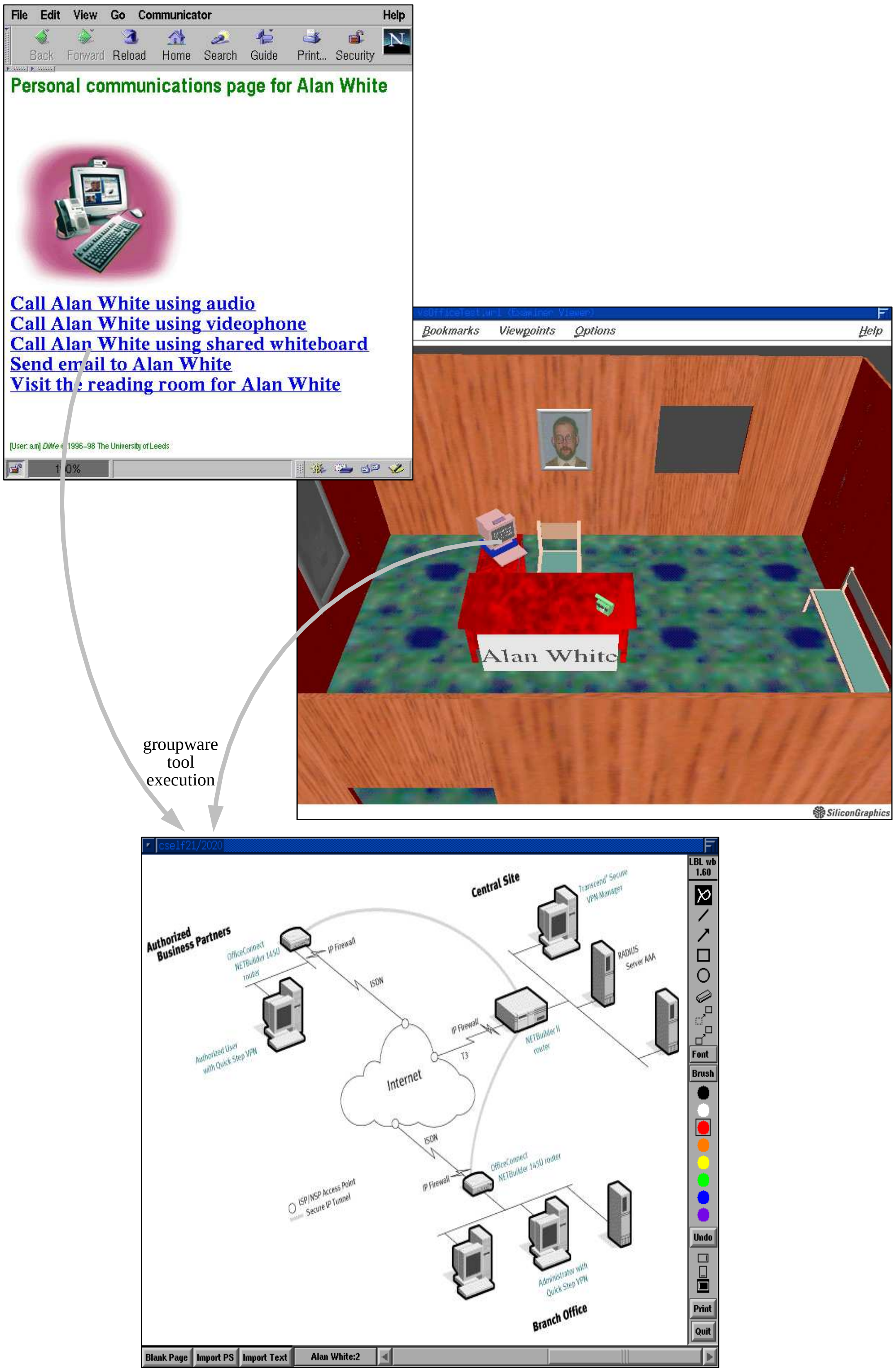

File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print... Security
Personal communications page for Alan White
Call Alan White using audio
Call Alan White using videophone
Call Alan White using shared whiteboard
Send email to Alan White
Visit the reading room for Alan White
Bookmarks Viewpoints Options Help
Alan White
SiliconGraphics
groupware
tool
execution
LBL wb 1.60
Central Site
Authorized Business Partners
Internet
Branch Office
Font
Brush
Undo
Print
Quit
Blank Page
Import PS
Import Text
Alan White:2

Figure 4.11: Groupware control metaphor using the LBL wb whiteboard tool

Chapter 5

Critique

5.1 Assessment methods

The research presented in this thesis proposes a model-based systems development approach for integrated internet CSCW systems. The approach was realised through the design of the ParaDiMe system architecture, as presented in Chapter 3. A proof-of-concept CSCW application was then created according to the model-based approach, using the ParaDiMe architecture. In measuring the extent to which the research meets the objectives set out in Chapter 1, a number of methods were employed to assess the approach, architecture and implementation;

- Software demonstration and feedback session,
- Follow-up interviews with potential users,
- Assessment of feasibility within VWS,
- Assessment from a software engineering perspective.

As described in Chapter 4, the proof-of-concept *workflow helper* application was constructed using the ParaDiMe architecture. The workflow helper was designed to demonstrate the salient features of the model-based systems development approach in supporting participants of a network design business process. An important element of the research assessment was to demonstrate the workflow helper to a potential user community and gain feedback. A group of potential BT users of the workflow helper agreed to attend a presentation on the approach, followed by a demonstration of the workflow helper application.[1] Although unaware of the specific details of the model-based approach, attendees were familiar with the network design business process scenario and the broad objectives of the research. The presentation, which lasted approximately an hour, attempted to describe the motivation for the research and introduce the key architectural concepts that would subsequently be demonstrated. A number of usage scenarios for the proof-of-concept ParaDiMe implementation were introduced in the presentation, as summarised in Section 5.2. These scenarios (based upon the case study network services quotation process) were used to provide a storyboard for the software demonstration. The objective was to help position the work within a familiar telecommunications context and focus feedback towards the specific research objectives. Following the presentation, the key features of the workflow helper application were demonstrated to the audience via the usage scenarios described in Section 5.2. Informal feedback was received during and after the demonstration, and points were written down in note form. These notes are summarised in Section 5.3.

Following the demonstration of the architecture implementation to a group audience at BT Laboratories, six staff agreed to participate in longer one-to-one interviews to provide more detailed feedback on the research. As well as providing useful assessment of the approach, the interviewees made a number of useful practical suggestions that drove further development iterations of the architecture and the workflow helper demonstration. Because of the broad nature of the demonstration and the varying backgrounds of the interviewees, a formal questionnaire-based technique was rejected in favour of informally structured interviewing. Notes were taken during interviewing and then transcribed into a report [110]. Whilst not rigidly adhering to a predefined script, the interviews attempted to elicit answers around the following topics;

- perception of the research approach as potential users,
- feedback on the workflow helper implementation,
- relevance of the approach to BT applications.

[1]The research was jointly supervised by BT Laboratories, but to promote objectivity, the assessment process was carried out in conjunction with staff who were not directly involved with the PhD project.

The interviews were conducted over a week long visit to BT premises. The interviews lasted between fifteen and forty-five minutes, with an average duration of around half an hour. After conducting the set of interviews, similar comments from each interviewee were collated. Feedback was varied, reflecting the range of backgrounds and perspectives of the interviewees. However, the salient results could informally be grouped under several headings, as summarised in Section 5.4.

In addition to obtaining feedback on the research from the software demonstration and follow-up interviews with potential users, an informal assessment of the relevance of the work to system builders within the Virtual Working Systems Group was conducted. The results of this assessment, carried out via informal conversations with VWS staff, is summarised in Section 5.5. Finally, the research was critically assessed in comparison to relevant existing software engineering approaches towards system prototyping and maintenance, as presented in Section 5.6. The remainder of this chapter is structured as shown below.

Section 5.2 describes system usage scenarios for services quotation business process, which were used to drive the practical demonstration;

Section 5.3 summarises the feedback obtained during and after the software demonstration;

Section 5.4 summarises the results of interviews with potential users of model-based tools within BT, following demonstration of the model-based architecture;

Section 5.5 summarises the results of informal assessment of the research within the VWS group at Leeds;

Section 5.6 analyses the relative success of the model-based approach developed within this thesis in comparison to existing approaches, methods and guidelines for prototyping and evolutionary maintenance.

5.2 Demonstration scenarios

The software demonstration consisted of four usage scenarios, presented in terms of the data services provision business process described in Chapter 4;[2]

1. basic iterative user interface customisation,
2. creation of a new business process support application,
3. exploratory prototyping of user interface requirements,
4. evolutionary change in business process characteristics.

All the use cases, as described below, assume the existence of the model-based architecture implementation comprising reusable CSCW services, appropriate libraries of general user interface components and the necessary runtime architecture. The use cases were based upon the stakeholder roles identified for the quotation scenario; a process designer, end user representative and system manager. During demonstration, the implementation was *driven* by an operator performing the system manager role, using the prototyping console (shown in Figure 4.5). The stories around which the use cases were demonstrated are summarised below;

Basic user interface customisation

The workflow helper application is supporting participants of the network services quotation process. Currently, all users interact with the system via a vanilla workflow worklist user interface. However, the account manager is unfamiliar with the workflow environment and asks the end-user representative if the interface could be made easier to use. The end-user representative, account manager and system manager meet and examine the current worklist interface features. The account manager suggests changes to the help feature on the worklist interface. The system manager locates a 'help metaphor' component from the user interace repository, inserts this into the account manager worklist interface using the architecture, immediately showing the enhanced interface to the users. (Visual results for this use case are shown in Figure 4.10 on page 111).

[2]The assistance of Paul Kearney (of BT) is acknowledged in developing these use cases.

Creation of a new business process support application

The process designer is required to put in place a new business process within his business unit. The new process (data network design) is similar to an existing process currently supported by the workflow helper application within the department. Hence, the process designer would like to reuse several elements of this existing support infrastructure within the new application. The process designer discusses the process requirements with the system manager, and provides definitions of the new business process as PIF and LDIF files. The process designer then explains the model descriptions in detail to the system manager, who identifies basic requirements for information, coordination and collaboration services. Required runtime architecture components are selected and populated with the models provided by the process designer. A set of generic user interface components are located within the metaphor library and used to build a quick demonstration of the application. The end-user roles (network designer, legal advisor and account manager) are identified and their requirements analysed in conjunction with the end-user representative. Application-wide metaphors for the support environment (worklists, workitems etc.) are then specialised for each end-user role using the basic technique demonstrated in the first use case.

Exploratory prototyping of user interface requirements

The account manager has been using the workflow helper environment for a period of time, but is generally unhappy with the features it provides to assist him in keeping customers up-to-date with the progress of their quotations. He arranges a meeting with the end-user representative and system manager. The account manager explains that he requires the ability to check job progress and communicate with other staff assigned to component tasks, but doesn't know how these features might best be provided within the workflow helper application. The system manager notes these general requirements and arranges to meet the account manager again in a few days. During this time, the system manager uses the model-based architecture to build and modify a number of similar user interface metaphors that enable navigation through business processes in different styles. A reusable groupware subsystem metaphor is also added to the prototype interface set, to demonstrate communications features. The system manager, account manager and end-user representative meet again, and the system manager demonstrates several alternative interfaces to the account manager. The account manager chooses the version which most closely meets his requirements, which is then moved into the production system by the system manager. (Visual results for this use case are shown in Figures 4.9 and 4.11 on pages 110 and 112).

Evolutionary change in business process characteristics

Support of the data services quotation process through the workflow helper has been operational for some time, and now needs to be modified. Five types of evolutionary changes may be envisaged,

1. change in application-wide user requirements
   e.g. all interfaces must adopt new corporate style guidelines;
2. change in role-specific user requirements
   e.g. a specialised workitem interface for the legal advisor role;
3. change in personal user requirements
   e.g. account manager Alan White requires a context help feature;
4. change in business process model
   e.g. the network design activity is split into research and reporting phases;
5. change in application information model
   e.g. a new attribute is added to the network design information object.

Cases 1–3 can be demonstrated through reference to the previous use cases. Changes in business process characteristics (as indicated by changes in process definition or application information models) may be considered through a single use case;[3]

The process designer navigates the quotation provision process definition using the structural and IDEF activity views within the workflow helper application. Following instruction from business unit management, the process designer is asked to further divide the

[3]Because PIF business process definitions are treated in the same way as any other information sources within the ParaDiMe information manager subsystem.

network design task into two explicit subordinate phases, a research activity and a reporting activity. The process designer uses a modelling tool to modify the process definition, which is then exported from the tool into PIF and passed to the system manager. The system manager loads the application with the updated model at an appropriate point, and codes new workitem metaphors for the participants affected by the new subtasks. No other changes are necessary because the application-wide metaphors are designed to cope with evolving process and information models.

5.3 Feedback from the demonstrations

Demonstrations of the model-based approach contextualised by the above use cases were generally well-received. The proof-of-concept architecture implementation was successful in enacting the four use cases, thereby providing a basic existence proof of the approach taken within this research. The demonstrations were conducted at BT premises using the WWW to access a remote application server based at the CVWE in Leeds. This worked adequately, but poor network latency adversely affected perception of the architecture. Interface generation performance was not a primary consideration during implementation and hence, when compounded by a slow client-sever network connection, interface generation was particularly slow (taking around 10 seconds round-trip time for generation of a simple HTML page, for example).

The use cases, and the ways in which it was envisioned that a model-based architecture could support them, were seen as realistic (although necessarily simplified for research purposes) and of benefit to process-oriented system builders. The application of the architecture in supporting rapid exploration of design ideas with end-users was seen as particularly useful. However, several people were less convinced of the approach in supporting process evolution, positing that most current major WFMS implementations are model-driven and as such can 'evolve' unaided already. This is true of course, but in defence it was argued that the workflow helper is a specific demonstration of a general systems development approach. Hence, the fact that it can exchange information and co-evolve with commercial WFMS implementations is a benefit of the model-based approach, not a duplication of development effort.

5.4 Results of follow-up interviews with potential users

Synergies with the IBS approach

A primary advantage of the model-based approach was seen as the ability to quickly integrate services, providing the role of *value-added middleware* within a dynamic enterprise. A number of users remarked that it was useful in 'bringing people further into the business process environment,' enabling at least partial reconciliation of process-oriented and *ad hoc* collaboration approaches to supporting users within CSCW scenarios. Using the model-based approach within Agent-based Workflow architectures was seen as feasible, particularly in providing flexible user interfaces that enhance human interaction and collaboration within the workflow environment. Several interviewees asked whether the architecture could be used to provide *information visualisation* services within the ABW architecture e.g. creating 3D representations of inter-agent negotiation. Informal experimentation within the model-based architecture following these questions concluded that such services could be provided, but would require development of a specialist set of builtin objects.

Linkage with commercial systems

The adoption of industry standards for integrating services was viewed as important to the credibility of the research approach working within a large enterprise. The use of X.500 (directory services) and CORBA query services (information management) was seen as useful, in that a wide range of other enterprise information sources could be rapidly integrated using similar approaches. The assumptions made about access to build-time and runtime WFMS appeared to be sensible within the approach, although it was noted that significant further development would be required to bring the implementation to production functionality. One perceived advantage of the distributed systems architecture, through which the model-based approach is realised, was that it enables computing power to be leveraged at a wider range of points within the enterprise. Many existing workflow users have terminal-type equipment whereas others have sophisticated PCs and workstations. The model-based approach could enable basic interfaces to be provided to terminal users and more sophisticated interfaces to be provided to users of more powerful clients, with no distinction made at the applications level.

Prototyping versus evolution

The utility of the model-based approach in enabling rapid exploration of design choices between scenario stakeholders was noted. A key advantage of the approach was viewed as the fast turnaround time on iterations through model modification, rather than code modification and recompilation. It was envisaged that this could enable some degree of *real-time* prototyping to take place with users, improving upon existing techniques in which there is a necessary delay of hours/days between prototype iterations. The IBS research emphasis is towards dynamic evolution of production systems, rather than iterative prototype-based construction of new systems. It was noted that, because the implementation system components were themselves of a prototypical nature, it was difficult to assess claims about the applicability of the approach within production environments. However, it was further noted that there was a good informal indication from the experimental architecture that such claims could be true, but substantiation would require development of the architecture implementation to more robust production-quality level. It was felt that the model-based approach could, in principle, support both prototyping of new applications and evolution of production systems at the modelling layer. That is, the role of prototyping would be to quickly derive a good user interface model, but the application components driven through this model during prototyping would ultimately be discarded. The model would then serve as a specification from which a production quality system could then be engineered, either using conventional development processes or a robust version of the model-based architecture.

Information integration and abstraction

The element of the implementation which attracted most criticism during demonstration and interviews was the application information model. The initial position taken during development of the model-based approach was that all information sources could be mapped into a simple E-R graph representation. In retrospect, this view proved to be somewhat naive and several consequential problems were encountered during implementation of the workflow helper application. The most serious of these problems was that information *abstraction* issues were not addressed adequately. For instance, within the workflow helper, several abstract *tiers* of information may be identified e.g. business process class (quotation processes), business process (network services quotation), process instance (quotation job 1003), process instance status (job 1003 awaiting legal advice). The workflow helper application attempted to cover several of these tiers, but the separation between layers become confused during implementation e.g. process meta-descriptive information and instance information was incorrectly mixed within PIF files. Although these criticisms were towards the workflow helper application, not the model-based development approach, they did serve to identify that the domain information model was much more complex than originally thought. Whilst the basic architectural mechanisms proposed for information management within the research appeared to be valid, the methodological emphasis upon information modelling was underestimated.

User-initiated adaptivity

Several interviewees noted that the research approach seemed restrictively focused upon supporting system-initiated user interface adaptivity. For example, all the use cases for the proof-of-concept implementation demonstration assumed that an expert operator would be available to make applications changes within the architecture. It was asked if, through appropriate use of the information manager and metaphor definitions, support for user-initiated interface adaptivity might be supported. In response, a quick experiment was constructed using the architecture in which a simple user profile was stored as a persistent information object. A metaphor was then written that enabled menu-based updating of this object using the form interaction subsystem. This enabled a user to, for example, select a default help level from a menu and for this to be kept and used over several sessions. The basic mechanisms of the architecture appeared adequate in supporting persistent user profiles to enable user-initiated interface adaptivity. It was noted that, as a future work package, it would be useful to package up these features into a reusable module as they would be useful in many scenarios. One interviewee also stated that this basic mechanism could feasibly be used to provide location-driven interface adaptivity, a manner similar to home/remote locations in mobile telephony. This would enable different profiles to be created for different computing clients from which a user connects to the application server (e.g. desktop PC, laptop, dial-up access, PDA). Providing that metaphor sets were created for each type of client computing device, the profile could be used to guide dynamic selection of the appropriate metaphor set at login.

Web user interface consistency

There were a number of questions relating to the use of Web user interface techniques in supporting multi-user applications. Firstly, it was asked how context switching within the Web interface was handled i.e. how the system could cope with a user browsing elsewhere and then returning to the application at some future point. Within the current proof-of-concept implementation there is no explicit support for session control. The problem has, however, been tackled successfully by other researchers developing session-oriented applications within Web infrastructure (e.g. the Stanford KSL ontology editor). When constructing the proof-of-concept implementation it was assumed that, as with security, session control measures would be present in a production environment but did not warrant explicit research investigation. The question asked was, "given that the model-based approach supports a service provider in specifying how an artefact should be manifested at the user interface, yet also supports a service consumer in customising their interface to the artefact, is it not likely that semantic inconsistencies will be introduced e.g. a feature deemed critical by the service provider is demoted or even ignored by the consumer?"[4] This problem was not explicitly addressed within the research approach, in that it was assumed that the application designer would be responsible for ensuring that such inconsistencies do not arise.

However, it was accepted that in supporting interface customisation for both 'producers' and 'consumers', there was more chance of problems occurring. There appears to be no simple solution to this problem. However, some support for *component maintenance* could be provided within the model-based architecture; as a well-defined set of metaphor definitions for an application forms an inheritance hierarchy, it was suggested that a token with the same semantics as `final` in (e.g.) Java could be used within the `add` construct to ensure that a component metaphor was not redefined or overridden in subclassed metaphors. Whilst this would not guarantee consistency, it would go some way towards supporting preservation of important interface features. Finally, the use of the

[4]This is an example of what is referred to as a *feature-interaction* problem within the telecommunications community.

model-based approach to provide an interface abstraction layer was identified as being of benefit. Large enterprises must typically support a heterogeneous set of client hardware and software, over which the model-based approach could provide an abstract user interface generation layer. The phenomenal pace at which Web technologies are advancing was also noted as a reason for adopting a model-based interface generation approach. It was felt that the approach would be a useful structured approach in bringing new Web technologies *on-stream* whilst still supporting existing baselines across an enterprise computing environment.

5.5 Assessment within the VWS group

The majority of the research work reported within this thesis was conducted at Leeds, within the CVWE. Most of the research assessment was conducted with BT however, in order to obtain more objective results. Although of a less formal nature, assessment was also carried out within the CVWE at Leeds in order to measure the potential benefit of the research to development of VWS solutions. As the project neared completion, a number of discussions were held with members of the engineering team within VWS Ltd. (the commercial exploitation arm of the CVWE) to elicit feedback on the research approach. The comments provided during these discussions are summarised as follows.

The main application for ParaDiMe, and hence the research approach, within VWS was viewed as supporting rapid generation of customised VWS solutions for customers. Although most existing VWS implementations integrate a wide range of core services, several clients have expressed interest in acquiring a VWS system that is tailored to their specific requirements. It was felt that ParaDiMe could help bring together core services into customised VWS *clones* in a rapid yet controlled manner. A customer would be able to select required services, perhaps through a special demonstration environment, and then ParaDiMe would assist in the generation of a VWS supporting the required features. The reduced maintenance costs associated with the model-based approach were also thought to be of significant benefit with respect to cloning VWS services. It was felt that, without a unified development and maintenance approach, supporting a heterogeneous base of VWS implementations would be extremely difficult.

A further benefit of the ParaDiMe approach, noted by a VWS engineer, was in supporting a spectrum of Web clients using an abstract interface layer. The VWS engineering team support a large number of users using a variety of client computing platforms and Web browsing software to interact with implementations. A number of inconsistencies have been observed between different versions of browsers and between platforms, meaning that it becomes difficult to guarantee usability levels without adopting a low baseline. It was envisaged that a practical role for ParaDiMe within VWS solutions would be in supporting heterogeneous browsers and platforms through specialised metaphor sets.

The feedback on the model-based approach from VWS was generally positive. However, it was noted that it would be unwise to adopt ParaDiMe as a core component of the VWS software architecture unless it was developed to commercial product quality.

5.6 Assessment from a software engineering perspective

The research reported within this thesis has investigated a model-based approach towards iterative construction and evolution of integrated internet CSCW applications. The broad hypothesis under consideration is that a model-based approach can decrease cycle times during iterative prototyping and maintenance activities. Construction and demonstration of a prototypical application through a toolkit supporting the model-based approach provided useful feedback on the research. In addition, it is beneficial to consider the value of the approach in comparison to existing software engineering guidelines and methods.

It was initially assumed that the model-based approach was equally applicable to prototyping and evolutionary systems development. As the research progressed, it became apparent that this assumption was somewhat misconceived. The subtle differences between prototyping and iterative development of production systems were not fully understood during the early phases of the work, in that prototyping was compared incorrectly to Rapid Applications Development (RAD) [70]. The goal of prototyping is to improve requirements elicitation in order to derive a more accurate specification which can then be used to drive development. In contrast, RAD attempts to deliver a complete production quality system as quickly as possible Prototyping may, of course, be used within a RAD cycle but it is not essential and prototype code may often be abandoned after it has served it's requirements engineering purpose [78]. There are good reasons for not propagating code developed through prototyping into production systems e.g. it may not be possible to retro-fit important system services ignored during prototyping and the system structure will typically degrade through *ad hoc* modification during prototyping [107]. In analysing the approach developed within this research, it is thus useful to consider its application towards prototyping and evolutionary development of production systems separately;

Prototyping

The model-based approach proposed in this thesis was validated through experimental implementation of the architecture, described in Chapters 3 and 4. Investigative emphasis within the research was placed upon the use of the architecture within prototyping phases of software development projects [111]. In [15], Boar defines a number of *candidacy factors* by which an assessment of the extent to which an application is amenable to prototyping may be made; application area, application complexity, customer characteristics and project characteristics. As a application domain, CSCW is not a good candidate for prototyping because of the baseline level of system complexity required to demonstrate basic cooperative working functionality. The model-based approach promotes rapid prototype implementation through reuse of basic building-block system components however, helping to overcome much application complexity. With the architecture available to system developers, all competency criteria for prototyping defined by Boar are satisfied; if complexity is reduced to an adequate level through a model-based approach, CSCW applications become good candidates for prototyping because of their high levels of user interactivity. Customer and project characteristics may respectively be met through willingness to participate in evaluation and provision of the architecture environment within the system development method. As a prototyping tool, the model-based architecture developed using ParaDiMe appears to mirror Somerville's four heuristics for reducing prototype cycle times [107];

Use a high-level language: Although system-oriented, DMSL may be viewed as a high level executable specification of user interaction;

Relax non-functional requirements: The current architecture implementation does not address performance, security or other non-functional issues (although these may be required in a production quality architecture implementation);

Ignore error conditions: Error-handling within the current architecture has not been fully developed—it is assumed that models used to drive the architecture are correct;

Reduce reliability and quality: Reliability and quality within the architecture are dependent directly on the quality of component service implementations.

Of course, the fact that the current architecture implementation exhibits these properties is influenced by the compressed timescale over which it itself was developed. Whilst identifying that the architecture *is* appropriate in prototyping Web-based cooperative systems, this does not imply that the approach is hence *not* appropriate in development of production systems.

Evolutionary development and maintenance of production systems

It is well known that software engineering based upon modular reusable software components ultimately improves product reliability and quality [57]. This was a central motivation in applying such an approach to development of internet-based cooperative systems. The approach developed within this thesis is not intended to facilitate a seamless transition from prototyping through to production system development and maintenance. Rather, the approach appears to be useful during prototyping and, given production quality architectural components, the same approach could offer several advantages in the ongoing evolution of production systems.

Maintenance is of major concern for software builders, accounting on average for some sixty-percent of product lifetime cost [97]. The object oriented nature of the model-based architecture developed within this research helps reduce maintenance effort as system changes are required. Firstly, the emphasis placed upon reuse of existing service components reduces the *target-area* in which errors may occur (assuming that the reusable components are themselves error free). Secondly, the *side effects* of maintenance activities [90] are reduced through the highly object-oriented nature of the architecture. These properties were observed experimentally during construction of the workflow helper proof of concept implementation. ParaDiMe forces developers to delineate user interface, information model and application-specific code. It was initially thought that as requirements changed, significant modification to application-specific code would be required. However, after a number of corrective and perfective maintenance iterations, the compiled code portion of the workflow helper remained fairly static. The majority of subsequent changes could be affected at the modelling layer e.g. the use case which demonstrated evolution of the workflow helper application did not require any code changes, rather reconfiguration of the information, process and user interface models used to drive the application. The model-based interface would thus seem useful during maintenance activities, provided that the basic system architectural components were designed to cope with reconfiguration at the model definition level. That is, implementation correctness following modelling reconfiguration could be guaranteed. A major aim within this research project was to develop architecture components that are resilient to model-changes. However, it was noted that development of components with intrinsic reuse and change resilience properties requires much more effort than tactical solutions. For example, it was informally noted during development of the workflow helper that creation of a *reusable* metaphor took (on average) over twice as long as development of a similar *hard-wired* example. However, this cost is a one-off cost and should be recouped through reduced maintenance costs.

Although it is noted that prototyping and evolutionary maintenance are different problems, the model-based approach can be beneficial during both phases of the software lifecycle as described above. Furthermore, there appear to be advantages in linking the two phases through the approach. Prototyping using ParaDiMe derives a model reflecting user requirements which can then directly be used to engineer a production system (possibly re-implementing services but retaining the bulk of the derived model). Thus, there is less scope for requirements to be mis-translated or lost within the implementation phase. A further interesting linkage between the use of ParaDiMe during prototyping and maintenance phases is analogous to *spare-part* replacement in hardware maintenance. When faults occur in harware systems, it is often less expensive to replace a defective subsystem rather than repair a specific fault e.g. replacement of a defective computer memory module. A spare parts strategy could also be used to reduce maintenance effort within software projects [36]. Prototyping often generates a number of candidate feature implementations from which one is finally selected. The model-based approach directly supports *hot-swapping* of components following change requests and could, thereby, feasibly be used to rapidly integrate replacement modules. However, the caveat to the spare parts strategy (as noted by Pressman [97]) is that there is a tendency for the same mistakes to be made within independent implementations of the same module.

Chapter 6

Conclusions and future work

6.1 Conclusions

The research reported in this thesis has been concerned with structured development techniques for integrated CSCW applications that are delivered via internet infrastructure. The research investigated issues surrounding architectural support for CSCW applications, with an emphasis towards service reuse and integration (Chapter 2). An internet CSCW system development approach was proposed which integrates reusable services and application specific code through a *modelling* language which specifies user interaction with functional components (Chapter 3). The perceived advantage of this approach is that it enables faster prototype iteration and structured system maintenance. These claims were explored through construction of a proof-of-concept implementation of an application based upon the ParaDiMe architecture (Chapter 4). The research approach was primarily assessed through demonstration of the architecture in supporting prototyping and maintenance activities within a *workflow helper* application (Chapter 5). Within the broad goal of contributing to the understanding of structured development techniques for CSCW applications, three specific hypotheses were tested through the research. In assessing the results of the work, it is thus useful to retrospectively consider each of these original hypotheses.

"*To investigate how common requirements in cooperative working scenarios are met by reusable CSCW services and how such services can be brought to together in a structured manner which promotes their integration and reuse within a model-based architecture.*"

This objective arose from an informal observation of cooperative working scenarios, which noted that requirements for basic CSCW services were often duplicated across applications and application domains. Many other researchers have also investigated frameworks for CSCW services of course (e.g. [85, 45, 58]). The investigation reported within this thesis is distinctive as it specifically addresses reuse and integration of services within dynamic enterprise environments. A number of CSCW frameworks were surveyed, from which basic requirements for coordination, collaboration and information management services were identified. Existing research and development efforts within these areas were then considered, in order to identify mechanisms that would enable services to be implemented as reusable components within a model-based architecture. This investigation concluded that many CSCW services *can* be reused across applications, and proposed a simple layered framework within which integrated applications may be built from component services. However, it was also concluded that service reuse was often limited by the visibility of an application programming interface (API) to the service implementation. The investigation also realised that integrated CSCW systems would typically require integration with legacy services within large enterprise computing environments. Hence, the investigation focused upon mechanisms through which existing software services (e.g. databases, workflow systems) could be integrated within the model-based environment using standard techniques, rather than requiring redevelopment.

"*To demonstrate and assess the benefit of a model-based approach towards system development through rapid prototyping, to test the hypothesis that the approach significantly reduces prototype development cycle times thereby enabling a higher level of user participation in the design process.*"

The model-based approach proposed within this thesis appears to be highly appropriate in prototyping internet-based integrated CSCW applications. The major barrier that currently prevents such prototyping activities is the necessarily complex structure of these systems, which is reflected in the large amount of coding required to demonstrate basic functionality using conventional software engineering approaches. The model-based development approach hides much of this complexity, allowing the reusable building blocks of cooperative applications to be assembled and reconfigured very quickly during prototyping. Within the case study, it was possible to demonstrate scenarios of use in which *real time* system changes were explored with users. This level of iteration cycle time would be significantly more difficult, if not impossible, to achieve using conventional software engineering approaches. In addition to successful demonstration of the model-based approach through proof-of-concept implementation, a comparison to established software engineering prototyping guidelines was performed. The comparison revealed that the approach was consistent with requirements for a good prototyping architecture and, through simplification of application complexity, enabled integrated internet CSCW systems to meet prototyping *competency criteria* whereas they would not do so through traditional approaches.

"*To demonstrate and assess the benefit of a model-based approach towards maintenance of systems as user requirements evolve within live applications, to test the hypothesis that the approach reduces the software effort required to affect changes thereby enabling evolving user requirements to be more efficiently fed back into systems.*"

This hypothesis could not be substantiated, for several reasons. Firstly, in order to realistically assess claims about maintenance within production quality systems, one must create such a system and assess it's evolution naturally over time. However, the time constraints upon PhD research are such that this level of development and analysis was infeasible. Secondly, the approach was investigated through implementation of the architecture to a proof-of-concept level, within which the research approach was embodied. As this implementation was itself of prototypical quality, assessment of it's ability to support production systems would be of little value. But this does not invalidate the hypothesis, it serves merely to acknowledge that it was overly ambitious to attempt to test it within this research project. There was however, strong informal evidence to suggest the suitability of the model-based approach in supporting evolutionary systems construction and maintenance. It would certainly be wrong to assert that the approach could seamlessly flow from prototyping through implementation to maintenance. The well-known characteristic of system structure degradation during prototyping was strongly observed during case study assessment. Hence, the use of prototype system architecture components within production systems would not be recommended. However, the models used to drive the prototype could serve as useful design specification templates during implementation and maintenance activities. A production quality implementation of the model-based architecture would also appear to be of benefit during software maintenance cycles, reducing the target area for maintenance and reducing system change side-effects. However, because of the difficulties stated above, it must be concluded that it was not possible to provide substantive evidence with which to bolster these claims.

Generality of the model-based approach

The model-based approach to development of internet CSCW systems proposed in this thesis is based upon definition of reusable CSCW services and a specification language which describes user interaction with those services within an application context. At runtime, the specification is used to drive a Web user interface generator that dynamically integrates access to required CSCW services. Although the DiMe DMSL scripting language was selected as the modelling component of the ParaDiMe architecture, the model-based approach is itself not dependent upon DMSL. That is, DMSL could be replaced by other modelling representations for use in other contexts. Conceptually, the modelling component of the ParaDiMe architecture requires a frame-based knowledge representation [77]. Object oriented modelling languages would appear to be particularly appropriate as these map well to the ParaDiMe object model. However, models used by ParaDiMe are specifically designed to serve as runtime specifications used to integrate system services and generate user interfaces. Although more general HCI conceptual modelling frameworks (such as TKS [55]) could feasibly drive the ParaDiMe architecture, it appears likely that some model reformulation would be required to map conceptual interactional requirements onto system interaction mechanisms.

6.2 Future work

The research presented in this thesis has investigated basic system architectural mechanisms through which internet-based cooperative systems may be integrated, built and maintained. The results appear promising, giving rise to a number of directions within which future research and development activities might progress.

6.2.1 Modelling language and reasoning

The DiMe architecture and DMSL scripting language were chosen as an experimental development vehicle within this research and have proved to be useful in exploring the research approach. DiMe was designed primarily to solve a pragmatic system development problem; the automated construction of large scale Web sites that integrate information access within HTML and VRML interaction styles. Although it served to inspire the work presented within this thesis, DMSL is beginning to get pushed beyond its intended use. As a modelling language, DMSL suffers from the same problem as HTML in that it mixes content and structural semantics. The World Wide Web Consortium (W3C) are introducing a variety of technologies to repair this problem, such as style sheets, extensible markup language etc. The Display Metaphor Scripting Language requires similar re-design if it is to remain useful. However, it is not actually clear that DMSL is needed at all.

As noted during Chapter 3, in which the system architecture for the model-based architecture was introduced, DMSL is in essence an object oriented frame language. A native-code implementation was chosen to maximise interactive server-side user interface generation performance. But for conceptual modelling, a conventional knowledge representation language may be a more appropriate vehicle through which further work can be progressed. The main motivation for this statement is that knowledge representation languages are generally designed to enable automated reasoning engines to operate over them. Knowledge-based approaches towards user interface generation are well-established (e.g. See [104] for a survey of intelligent user interface research), and there are several reasons for adopting such an approach with ParaDiMe. The point of system-initiated interface generation within ParaDiMe is the metaphor selector component. This currently operates using a simple multi-attribute decision strategy in which a DMSL command is matched against a metaphor index listing target object classes, viewer contexts and style keywords for each metaphor known to ParaDiMe. Although useful, this mechanism is simplistic in comparison to most model-based interface generators. This was known during development of ParaDiMe and it has always been the intention to *outsource* metaphor selection to a more sophisticated architectural component at some point. A declarative frame-based modelling language to replace DMSL would ease this transition significantly.

Importantly, an AI approach would also widely increase the representational capability of DMSL and allow new concepts to be introduced in a uniform manner; as more and more constructs have been pragmatically added to DMSL it has lost some perspicuity and gained some unnecessary complexity. For example, once an internal model has been built within ParaDiMe from a DMSL script, subsequently modifying that model without reparsing the initial DMSL model can produce unpredictable results. In contrast, knowledge-based systems typically provide excellent built-in support for dynamic assertion and retraction of information. There are also good reasons for adopting such an approach in embracing other future research directions. For example, distributed agent and task-driven interface generation (as described in the following text) would both benefit from a more general knowledge representation language within ParaDiMe.

6.2.2 Distributed and agent-based solutions

A major feature of the model-based architecture developed within this work is the support for per-user interface customisation through personalised metaphor definitions and DMSL interpreters. ParaDiMe is a client-server environment in which interfaces are generated at server-side and passed over the network to thin Web clients as target language streams (e.g. HTML or VRML). This approach was adequate for this research in demonstrating the model-based approach for a small set of users, but does not scale well for more ambitious applications.

The proof-of-concept research implementation was built using the CORBA distributed object architecture which eased the scalability problem significantly. DMSL interpreters are allocated dynamically by the runtime ParaDiMe user interface generator as independent CORBA objects. With appropriate server management functionality (as provided in major ORB implementations), these objects may be launched over a pool of machines providing support for distributed load-balancing.

A more ambitious area for future research and development work is to leverage the increasing power of Web client machines, by moving towards client-side interface generation techniques. The current CORBA implementation of ParaDiMe provides an excellent starting point for this research, especially given the integration of the Visigenics ORB within the popular Netscape Communicator browser. Some initial experiments have achieved promising results in investigating client-side approaches. Notably, Gareth Bottomley produced a working Java applet implementation of a small subset[1] of DMSL as his final year undergraduate project [16]. This work used a Java version of PCCTS (as used in the major ParaDiMe implementation) and connected the tree-walker to a VRML external application interface (EAI) running within an applet. This enabled simplified DMSL constructs to be dynamically downloaded via a network connection and be applied directly to the VRML model running within the browser. A client-side interface generation approach is the logical next step forward for the model-based interface generation technique (and hence ParaDiMe), providing two key benefits;

Firstly, it will enable richer interaction modalities (such as the next generation of Internet interactive virtual reality standards) to be provided more efficiently than through conventional server-side techniques. For example, representation of a detailed *office* scene may require several megabytes of VRML and component models, textures etc. This must be generated and packaged on the server, sent over the (increasingly bandwidth-constrained) Internet and parsed into the client VRML browser. The client-side model-based approach would be to cache a library of basic reusable interface components on the client and use a controlling applet (or other code) to synthesise interfaces on-the-fly, in response to DMSL commands sent from the application server. This would provide massive bandwidth reduction over VRML transmission, even with scene-compression techniques.

The second advantage of moving towards a client-side approach is in more effectively supporting individual user interaction preferences. Mechanisms currently exist within ParaDiMe to support user initiated interface adaption through user profiles stored in the information space and appropriate references to these profiles within metaphors. For example, a profile might maintain a user-skill-level attribute through which the user might be classified as a novice or expert This information may then be read into ParaDiMe through metaphors and used to guide metaphor selection e.g. by dynamic assignment of the *style* metaphor context descriptor. However, this mechanism is somewhat inelegant and difficult to maintain in practise. A better solution would be to investigate ParaDiMe *agents*, which would manage presentation services on a per-user basis. It is envisioned that the agents would be tightly integrated within the client browser, with access to the internal document model API enabling dynamic interface generation capabilities. The

[1]Five constructs.

user interface customisation services provided through this approach could potentially integrate with other personal-assistant agent functions on client machines, cf. [67].

6.2.3 Task and interaction modelling

The 'models' that drive the model-based approach to cooperative systems development investigated within this research are user interface component models, rather like widgets in conventional user interface toolkits. These widgets encapsulate form and function into some useful interface abstraction e.g. a telephone metaphor that establishes an audio conferencing call upon selection within a scene. The ParaDiMe model-based interface generator provides the basic representational and operational mechanisms for building user interfaces to multi-user cooperative working systems. The research reported within this thesis has attempted to raise the level at which these systems are created closer to the user domain. However, there is still a wide gap between scenario requirements modelling and system implementation.

A future research direction for the model-based techniques developed within this project (and in related work within the CVWE) would be to investigate interface generation for cooperative systems from task-oriented conceptual models, extending existing theories and methods such as Task Knowledge Structures (TKS) [55], which has proved useful in rapid prototyping of systems through task models [56]. The objective in this research would be to derive a formal task-interaction model of cooperative work describing users, the shared work context to be provided by the system, and the interactions which must be supported within that work context (e.g. user-to-system, user-to-artefact within the system, user-to-user mediated by the system). The ambitious research goal would be to investigate user interface specification or generation directly from such a model (using an architecture such as ParaDiMe), although it is currently not clear that such an approach is either feasible or desirable. A potentially more tractable objective would be to use a task-oriented CSCW model to identify commonly occurring tasks and interactions; construct reusable system services and interfaces that embody these requirements; and then use a model-based interface generation toolkit to create integrated cooperative applications through rapid integration of these basic cooperative working *building blocks*.

6.2.4 Visual system construction tools

Although the model-based approach developed within this project attempts to further involve end-users within design and prototyping of cooperative systems, operation of the current user interface generation architecture (built using ParaDiMe) requires a relatively high level of technical expertise to accomplish simple tasks. Although this is perhaps more a development exercise rather than research direction, there is a need to derive visual tools that support the functionality of DMSL whilst hiding the language behind a graphical integrated development environment (IDE). Products such as Microsoft's Visual InterDev suite have significantly simplified the task of creating Web-based information systems, and a similar approach seems highly appropriate for involving non-technical stakeholders directly in the model-based systems development process. A starting-point in moving towards a visual toolkit would be to augment the functionality of a standard Web design tool (e.g. Microsoft FrontPage), with a toolbox of cooperative working widgets which could be 'drag-and-dropped' into interfaces as required. A control function (perhaps similar to the *publish* feature in FrontPage) could then generate out a set of runtime architecture components that would implement the functionality specified within the visual design tool. This work would take a significant amount of time and resources to implement successfully, but moving at least some way towards a visual toolkit is essential in investigating the wider use of model-based design techniques.

6.2.5 Richer 3D interaction styles

This research has been developed over a four year period between 1994 and 1998, during which time internet technologies have advanced greatly. One area that has attracted increasing interest during the project is internet virtual reality technologies, such as VRML. The research reported within this thesis has not attempted to contribute towards understanding in this area. An existing VRML interface generation module was adopted from a related CVWE research effort and was used primarily to demonstrate the utility of a model-based approach in seamlessly supporting both 2D and 3D interaction modalities. The decision not to pursue VRML interface generation as research investigation within this PhD was made primarily because of the poor support for scene interactivity in the VRML implementations available during the project (i.e. VRML 1.0). Whilst it was possible to generate VRML offices for example, it was not possible to populate or animate them with real world interactive objects e.g. people. Hence, whilst 3D presentational interfaces looked attractive and helped to contextualise system services for neophyte users, their actual contribution to the functional operation of system was of negligible benefit or even detrimental to usability.

Internet VR technology is advancing at a fast pace however, largely fuelled by the influential entertainment and leisure markets. Technologies such as VRML 2.0, Java 3D and more recently Microsoft Chrome are moving towards an Internet VR interaction infrastructure. The benefits of ParaDiMe are readily apparent within 3D interaction styles, where there are currently very few products supporting automated construction of interactive Internet-based VR worlds. Fewer still, if any, enable dynamic integration of CSCW services within these environments. Hence, there is potential research in investigating model-based construction of cooperative working environments using the emerging interactive Internet VR technologies. The emphasis in this investigation would be in task-based interaction modelling as described earlier, differentiating the research from existing work such as that of Benford et al. [11]. It would be particularly interesting, given ParaDiMe's flexible interface generation capabilities, to study the integration of support for 2D and 3D interaction modes within such environments (e.g. switching from a 3D style used to browse a virtual library shelf to a 2D style used to interact with a particular document).

6.3 Closing remarks

This thesis takes a software engineering perspective upon supporting cooperative work within dynamic internet environments. Software engineering methodologies provide well-established modelling and design abstractions for the functional and behavioural characteristics of such systems. However, direct representational and tool support for HCI-informed systems engineering is at best peripheral and at worst non-existent within current software engineering methods (See e.g. [59]).

In the same way that the *patterns* work of Gamma et al codified and encapsulated best-practise knowledge to promote reuse of object-oriented software concepts [32], there is a real need for HCI design patterns to be fed into mainstream software engineering. This firstly requires that HCI principles be embraced within widely-used software engineering methods; not just making a token appearance in non-functional requirements or style guides. Secondly, there is a requirement for these patterns to be embodied within CASE tools so that software engineers and usability engineers may share a common interlingua that more formally injects HCI knowledge through into implementations.

The opinion advocated through this thesis is that the World Wide Web will be a primary factor in narrowing the gap between HCI and software engineering. As client-server implementations, the user interfaces of Web information systems are necessarily separated from functional components. Unlike conventional client-server systems however, the HTML user interface delivery mechanism is ubiquitous and externally adaptable by non-programmers. As Gayna Williams of Microsoft's usability group notes, HTML interfaces are now becoming important design specification and prototyping tools that enable participatory design iterations to occur extremely late in the product development process [118]. The ultimate goal of the research presented within this thesis is to move towards CASE tools and methods which directly support this prevailing compressed form of software product lifecycle.

Bibliography

[1] George Almasi, Anca Suvaiala, Ion Muslea, Calin Cascaval, Ted Davis, and V. Jagannathan. Web*: a technology to make information available on the Web. In *Proceedings of the Fourth IEEE Workshop on Enabling Technology: Infrastructure for Collaborative Enterprises: WET-ICE '95*, Concurrent Engineering Research Centre, West Virginia University, 1995.

[2] American National Standards Institute, *Standard X3.135-1992. Database Language-SQL*, January 1993.

[3] Architecture Projects Management Ltd. *The ANSA Reference Manual*, 1989.

[4] Lowell Arthur. *Rapid evolutionary development : requirements, prototyping and software creation*. Wiley, New York, 1992.

[5] A. Bäcker and U. Busbach. DocMan: A document management system for cooperation support. In *Proceedings of the 29th Hawaii International Conference on the System Sciences*, volume 3, pages 82–91, Maui, January 1996.

[6] L. Bannon and K. Schmidt. Issues of supporting organisational context in CSCW systems. COMIC Project Deliverable 1.1, Ocober 1993.

[7] Liam Bannon and Kjeld Schmidt. CSCW: Four characters in search of a context. In J. M. Bowers and S. D. Benford, editors, *Studies in Computer Supported Cooperative Work*. Elsevier Science Publishers, North Holland, 1991.

[8] M. Barbuceanu and M. Fox. The architecture of an agent based infrastructure for agile manufacturing. Technical report, Enterprise Integration Laboratory, University of Toronto, 1995.

[9] M. Barbuceanu and M. Fox. The information agent: Building intelligent information infrastructures for enterprise integration. Technical report, Enterprise Integration Laboratory, University of Toronto, 1995.

[10] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18:323–365, 1986.

[11] Steve Benford, John Bowers, Lennart Fahlen, John Mariani, and Tom Rodden. Supporting cooperative work in virtual environments. *The Computer Journal*, 37(8):653–668, 1994.

[12] Steve Benford, Adrian Bullock, Neil Cook, Paul Harvey, Rob Ingram, and Ok-ki Lee. From rooms to cyberspace: Models of interaction in large virtual computer spaces. In *Interacting with Computers*. Butterworth-Heinmann, 1993.

[13] R. Bentley, T. Horstmann, K.Sikkel, and J. Trevor. Supporting collaborative information sharing with the World Wide Web: The BCSCW shared workspace system. In *Proceedings of the Fourth International World Wide Web Conference*, Boston, December 1995.

[14] Gordon Blair and Tom Rodden. The impact of CSCW on Open Distributed Processing. In De Meer et al. [23], pages 143–153.

[15] B. Boar. *Application Prototyping.* Wiley-Interscience, 1984.

[16] Gareth Bottomley. *A client-side Java VRML generation language.* BSc final year project dissertation, School of Computer Studies, University of Leeds, UK, 1998.

[17] Adrian Bullock. Visualising organisations. In John Bowers, editor, *A Conceptual Framework for Describing Organisations*, chapter 10, pages 229–240. COMIC Project Deliverable D1.2, October 1994.

[18] R. G. G. Cattell, editor. *The Object Database Standard: ODMG-93 v1.2.* Morgan Kaufmann, 1994.

[19] P. P. Chen. The entity relationship model–towards a unified view of data. *ACM Transactions on Database Systems*, 1(1), March 1976.

[20] M. F. Costabile, D. Malerba, M. Hemmje, and A. Paradiso. Building metaphors for supporting user interaction with multimedia databases. In *Proceedings of 4th IFIP*

2.6 Working Conference on Visual DataBase Systems (VDB 4), L'Aqulia, Italy, May 1998.

[21] T. Crowley, P. Milazzo, E. Baker, H. Forsdick, and R. Tomlinson. MMConf: An infrastructure for building shared multimedia applications. In *Proceedings of CSCW '90*, Los Angeles, October 7–10 1990.

[22] S. Das, K. Kochut, J. Miller, A. Sheth, and D. Worah. ORBWork: A reliable distributed CORBA-based workflow enactment system for METEOR$_2$. Technical report, LSDIS Laboratory, University of Georgia, 1997.

[23] J. De Meer, V. Heymer, and R. Roth, editors. *Proceedings of the IFIP TC6/WG6.4 International Workshop in Open Distributed Processing*, Berlin, Germany, October 1991.

[24] P. M. Dew, C. M. Leigh, R. S. Drew, D. T. Morris, and J. M. Curson. Collaborative working systems to support user interaction within a virtual science park. *Information Services and Use*, 15:213–228, 1995.

[25] Richard Drew. *Integrated Information Directory Services to Support the Innovation Process.* PhD thesis, School of Computer Studies, University of Leeds, UK, June 1997.

[26] Rae Earnshaw and John Vince, editors. *The Internet in 3D: Information, Images and Interaction.* Academic Press, 1997.

[27] C. A. Ellis, S. J. Gibbs, and G. L. Rein. Groupware: some issues and experiences. *Communications of the ACM*, 34(1), January 1991.

[28] S. Elrod, R. Bruce, R. Gold, F. Halasz, W. Janssen, D. Lee, K. McCall, E. Pedersen, K. Pier, J. Tang, and B. Walsh. Liveboard: a large interactive display to support group meetings, presentations and remote collaboration. In *Proceedings of CHI '92*, pages 599–607. ACM Press, 1992.

[29] D. Engelbart and H. Lehtman. Working together. In *Byte*, pages 245–252. CMP Media, December 1988.

[30] K. Fischer, J. P. Müller, I. Heimig, and A. W. Scheer. Intelligent agents in virtual enterprises. In *Proceedings of the First International Conference on the Practical*

Applications of Intelligent Agents and Multi-Agent Technology (PAAM '96), pages 205–223, 1996.

[31] Fernando Flores, Michael Graves, Brad Hartfield, and Terry Winograd. Computer systems and the design of organizational interaction. *ACM Transactions on Office Information Systems*, 6(2):153–172, April 1988.

[32] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1994.

[33] M. Genesereth and R. Fikes. Knowledge interchange format: Version 3.0 reference manual. Technical report, Computer Science Department, Stanford University, 1992.

[34] J. Giarratano and G. Riley. *Expert Systems: Principles and Programming.* International Thomson Publishing, 1994.

[35] S. J. Gibbs. LIZA: An extensible groupware toolkit. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, Austin, Texas, 1989. ACM Press.

[36] T. Gilb. Software spare parts. In G. Parikh, editor, *Techniques of Program and System Maintenance.* Winthrop Publishers, 1981.

[37] S. L. Goldman. 21st Century Manufacturing Enterprise Strategy: An Industry-Led View. Iococca Institute, Lehigh University, 1991.

[38] S. L. Goldman, R. N. Nagel, and K. Preiss. *Agile Competitors and Virtual Organizations.* Van Nostrand Reinhold, January 1995.

[39] P. M. D. Gray, A. Preece, N. J. Fiddian, W. A. Gray, T. J. M. Bench-Capon, M. J. R. Shave, N. Azarmi, M. E. Wiegand, M. Ashwell, M. Beer, Z. Cui, B. Diaz, S. M. Embury, K. Hui, A. C. Jones, D. M. Jones, G. J. L. Kemp, E. W. Lawson, K. Lunn, P. Marti, J. Shao, and P. R. S. Visser. KRAFT: Knowledge fusion from distributed databases and knowledge bases, database and expert system applications. In *Database and Expert System Applications (DEXA' 97)*, Toulouse, 1997.

[40] Saul Greenberg. Personizable groupware: Accommodating individual roles and group differences. In L. Bannon, M. Robinson, and K. Schmidt, editors, *Proceed-*

ings of the Second European Conference on Computer Supported Cooperative Work (ECSCW '91). Kluwer Academic Publishers, 1991.

[41] I. Grief, editor. *Computer Supported Cooperative Work: A book of readings.* Morgan Kaufmann, 1988.

[42] Shishir Gundavaram. Web Gateways: Increasing the Power of the Web. *World Wide Web Journal*, 2(2):191–202, Spring 1997.

[43] P. Gust. Shared X: X in a distributed group work environment. In *Proceedings of the Second Annual X Conference*, MIT, January 1988.

[44] S. Harrison and S. Minneman. The media space: A research project into the use of video as a design medium. In *Proceedings of Conference on Participatory Design*, pages 51–58, Seattle, WA, March 1990.

[45] P. Hennessy, P. Harvey, and H. Smith. Support for enterprise modelling in CSCW. Technical report, NEXOR Ltd., 1994.

[46] David Hollingsworth. The workflow reference model. Workflow Management Coalition, Document TC00-1003 Issue 1.1, November 1994.

[47] M. Huhns, N. Jacobs, T. Ksiezyk, W. M. Shen, M. Singh, and C. Tomlinson. Enterprise information modeling and model integration in Carnot. In Petrie [95].

[48] Neil Hunter. *Secure, User Centred Conferencing for Virtual Working Systems.* PhD thesis, School of Computer Studies, University of Leeds, UK, December 1997.

[49] International Business Machines Corporation, Vienna, Austria. *FlowMark Workflow Modeling*, 1994. Release 1.1.

[50] ISO/IEC 10646. Extensible Markup Language (XML). Standard specification, 1997.

[51] ITU and ISO/IEC 9594-1. X.500–The Directory: Information Technology, Open Systems Interconnection. Standard specification, 1992.

[52] Y. Jayachandra, editor. *Re-engineering the Networked Enterprise.* McGraw-Hill, New York, 1993.

[53] N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O'Brien, and M. E. Wiegand. Agent-based business process management. *International Journal of Cooperative Information Systems*, 1996.

[54] R. Johansen. *Leading Business Teams.* Addison-Wesley, Reading, MA, 1991.

[55] P. Johnson and H. Johnson. Task Knowledge Structures: psychological basis and integration into system design. *Acta Psychologica*, 78:3–26, 1991.

[56] P. Johnson, H. Johnson, and S. Wilson. Rapid prototyping of user interfaces driven by task models. In J. M. Carroll, editor, *Scenario-based design for human computer interaction.* John Wiley and Sons, Inc., 1995.

[57] T. C. Jones. Reusability in programming: a survey of the state of the art. *IEEE Transactions on Software Engineering*, 10(5):488–494, September 1984.

[58] T. Kalin, H. Lubich, and J. Rugelj. A proposal for the architectural model for CSCW. Technical report, Co-Tech Project (COST 14) WG3, 1990.

[59] Elizabeth Kemp and Chris Phillips. Extending support for user interface design in object-oriented software engineering methods. In May et al. [71].

[60] G. Lakoff and M. Johnson. *Metaphors We Live By.* University of Chicago Press, 1980.

[61] Kenneth Lantz. *The prototyping methodology.* Prentice-Hall, Englewood Cliffs, N.J, 1986.

[62] Peter Lazar and Peter Holfelder. Web database connectivity with scripting languages. *World Wide Web Journal*, 2(2):203–219, Spring 1997.

[63] J. Lee, M. Gruninger, Y. Jin, T. Malone, A. Tate, and G. Yost. The PIF process interchange format and framework. Technical report, PIF Working Group, University of Hawaii, May 1996. Version 1.1.

[64] J. Lee and T. Malone. Partially shared views: A scheme for communicating amongst groups that use different type hierarchies. *ACM Transactions on Office Information Systems*, 8(1):1–26, 1990.

[65] W. Litwin and A. Abdellatif. Multidatabase interoperability. *IEEE Computer*, 19(12):10–18, December 1986.

[66] J. D. Mackinlay, G. Robertson, and S. Card. Cone trees: Animated 3D visualizations of hierarchical information. In *Proceedings of the ACM SIGCHI '91 Conference on Human Factors in Computing Systems*, pages 189–194, 1991.

[67] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, July 1994.

[68] T. W. Malone and K. Crowston. Toward an interdisciplinary theory of coordination. Technical Report 120, Center for Coordination Science, MIT Sloan School, 1991.

[69] Marilyn Mantei. Observation of executives using a computer supported meeting environment. In *Decision Support Systems 5*, pages 153–166. Elsevier Science Publishers, North Holland, 1989.

[70] James Martin. *Rapid Application Development*. Macmillan, 1991.

[71] Jon May, Jawed Siddiqi, and Julie Wilkinson, editors. *Adjunct Proceedings of BCS Human Computer Interaction '98*, Sheffield, UK, September 1998.

[72] D. McCarthy and S. Sarin. Workflow and transactions in InConcert. *Bulletin of the Technical Committee on Data Engineering*, 16(2), 1993.

[73] R. McCool. The common gateway interface. Technical report, NCSA, 1995.

[74] R. Medina-Mora, T. Winograd, R. Flores, and F. Flores. The ActionWorkflow approach to workflow management technology. In *Proceedings of CSCW '92*, pages 281–288. ACM Press, 1992.

[75] P. Merle, C. Gransart, and J. Geiß. CorbaWeb: A WWW and Corba worlds integration. In *Second COOTS Workshop on Distributed Object Computing on the Internet*, Toronto, June 1996.

[76] D. Miers. The use of technology within business process redesign initiatives and the future of information systems. In *Proceedings of the CCTA Emerging Technology Showcase*, pages 98–108, London School of Economics, January 4–6 1995.

[77] Marvin Minsky. A framework for representing knowledge. In P. Winston, editor, *Psychology of Computer Vision*. McGraw-Hill, New York, 1975.

[78] Ian Mitchell. *A CASE Supported Approach to Object-Oriented Rapid Prototyping*. PhD thesis, Rapid Prototyping Laboratory, University of Sunderland, UK, 1997.

[79] D. T. Morris, G. Lajos, P. M. Dew, R. S. Drew, and D. Willows. DiMe: An object oriented scripting language for the automatic creation of virtual environments. In *Proceedings of Eurographics UK Chapter Conference*, April 1997.

[80] National Industrial Information Infrastructure Protocol Consortium (NIIIPC). *NIIIP Reference Architecture: Concepts and Guidelines*, Technical Report NTR95-01 Cycle 0, January 1995.

[81] National Industrial Information Infrastructure Protocol Consortium (NIIIPC). *Task and Session Objects: Common objects for enabling virtual enterprise resource sharing and collaboration*, NIIIPC OMG Business Objects RFP Response, January 1997.

[82] National Institute of Standards and Technology. *Integration Definition for Function Modeling (IDEF0)*, Federal Information Processing Standards Publication 183, December 1993.

[83] National Institute of Standards and Technology. *Integration Definition for Information Modeling (IDEF1X)*, Federal Information Processing Standards Publication 184, December 1993.

[84] National Institute of Standards and Technology. *Framework for NII Services*, Report NISTIR 5478, December 1994.

[85] L. Navarro, M. Medina, and T. Rodden. Environment support for cooperative working. Technical report, Universitat Politëcnica de Catalunya and University of Lancaster Computing Department, 1994.

[86] A. Obaidi, R. Drew, and P. M. Dew. Generic approach to agent systems with application to adviser project. In *Proceedings of the ECSCW '97 Workshop on Social Agents in Web-based Collaboration*, Lancaster, UK, September 1997.

[87] Object Management Group. *CORBAservices: Common Object Services Specification,* OMG Document 97-12-03, November 1997.

[88] Object Management Group. *The Common Object Request Broker: Architecture and Specification,* OMG Document 97-09-01, Revision 2.1, August 1997.

[89] D. Palaniswami, J. Lynch, I. Shevchenko, A. Mattie, and L. Reed-Fourquet. Web-based multi-paradigm workflow automation for efficient healthcare delivery. In *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-art and Future Directions*, July 1996.

[90] G. Parikh, editor. *Techniques of Program and System Maintenance.* Winthrop Publishers, 1981.

[91] Terrance Parr. *Language Translation Using PCCTS and C++*. Automata Publishing Company, 1996.

[92] Encarna Pastor and Jonny Jager. Architectural framework for CSCW. In *Cooperation Among Organisations: the potential of Computer Supported Cooperative Work*, ESPRIT Research Reports, Project 5660. Springer, 1993.

[93] J. F. Patterson, R. D. Hill, S. L. Rohall, and W. S. Meeks. Rendezvous: An architecture for synchronous multi-user applications. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW '90)*, Los Angeles, 1990. ACM Press.

[94] Mark Pesce. *VRML: Browsing and Building Cyberspace*. New Riders, 1995.

[95] Charles Petrie, editor. *Enterprise Integration Modeling: Proceedings of the First International Conference*. MIT Press, 1992.

[96] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey. *Human Computer Interaction*. Addison-Wesley, 1994.

[97] Roger Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2nd edition, 1989.

[98] Wolfgang Prinz and Paola Pennelli. Relevance of the X.500 Directory to CSCW Applications. In D. Marca and G. Bock, editors, *Groupware: Software for Computer Supported Cooperative Work*, pages 209–225. IEEE Computer Society Press, 1992.

[99] Dave Raggett. Client-side scripting and HTML. In *Scripting Languages: Automating the Web,* World Wide Web Journal, volume 2, pages 29–37. O'Reilly, 1997.

[100] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. *HTML 4.0 Specification*. World Wide Web Consortium, April 1998. Recommendation REC-html40-19980424.

[101] A. S. Rogers and S. Gray. Virtuosi- supporting collaboration in design and manufacture. In *Procedings of Telecom '95 Technology Summit*, Geneva, October 1995.

[102] S. Rowett, S. Saunders, P. M. Dew, C. M. Leigh, and E. J. Foster. A virtual science park to support work-based learning. In *Proceedings of World Conference on Educational Telecommunications*, Calgary, Canada, June 1997.

[103] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorenson. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

[104] M. Schneider-Hufschmidt, T. Kühme, and U. Malinowski, editors. *Adaptive User Interfaces*. Elsevier Science Publishers, 1993.

[105] Amit Sheth. Workflow automation: applications technologies and research. SIGMOD Conference tutorial notes, May 1995.

[106] Amit Sheth. From contempory workflow process automation to adaptive and dynamic work activity coordination and collaboration. Technical report, Large Scale Distributed Information Systems Lab, University of Georgia, 1997.

[107] Ian Somerville. *Software Engineering*. Addison Wesley, 3rd edition, 1989.

[108] Guy Steele. *Common LISP: the language*. Digital Press, 2nd edition, 1990.

[109] M. Stefik, G. Foster, D. Bobrow, K. Kahn, S. Lanning, and L. Suchman. Beyond the chalkboard: computer support for collaboration and problem solving in meetings. *Communications of the ACM*, 30(1):32–47, January 1987.

[110] M. A. Swaby. Research assessment results: BT interview transcripts. Internal PhD project report, School of Computer Studies, University of Leeds, June 1998. Unpublished manuscript.

[111] Michael Swaby, Peter Dew, David Morris, and Gyuri Lajos. System support for rapid prototyping of collaborative internet information systems. In May et al. [71].

[112] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The Enterprise Ontology. Technical report, AIAI Enterprise Project, June 1995.

[113] F. Vernadat. CIMOSA: Enterprise modelling and enterprise integration using a process-based approach. In H. Yoshikawa and J. Goossenaerts, editors, *Information Infrastructure Systems for Manufacturing*, volume B-14, pages 65–79. Elsevier Science, North-Holland, 1993.

[114] Norman Walsh. An introduction to cascading style sheets. In *Advancing HTML: Style and Substance,* World Wide Web Journal, volume 2, pages 147–156. O'Reilly, 1997.

[115] J. Wernecke. *The Inventor Mentor: Programming Object Oriented 3D Graphics with Open Inventor.* Addison Wesley Publishing Company, 1994.

[116] Gio Wiederhold. Intelligent integration of diverse information. In T. Finin, C. Nichola, and Y. Yesha, editors, *First International Conference on Information and Knowledge Management*, Baltimore, November 1992.

[117] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.

[118] Gayna Williams. Usability process challenges in a Web product cycle. In May et al. [71].

[119] D. Woelk and C. Tomlinson. The InfoSleuth project: Intelligent search management via semantic agents. In *Proceedings of the Second International World Wide Web Conference*, October 1994.

[120] M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practise. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[121] World Wide Web Consortium. *Document Object Model Specification, TR-WD-DOM-971209 (Working Draft)*, December 1997.