



uOttawa

L'Université canadienne
Canada's university

**FACULTÉ DES ÉTUDES SUPÉRIEURES
ET POSTDOCTORALES**



**FACULTY OF GRADUATE AND
POSTDOCTORAL STUDIES**

Robin Tropper

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical and Computer Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

**Architecture and Programming Paradigm for a Scalable, Metamorphic and Cloud-Collaborative User
Environment**

TITRE DE LA THÈSE / TITLE OF THESIS

D. Ionescu

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

Liam Peyton

Chris Joslin

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Architecture and Programming Paradigm
for a Scalable, Metamorphic and
Cloud-Collaborative User Environment.

By:

Robin Tropper

B.A.Sc., B.Ed., B.Mus.

A thesis submitted to the
Faculty of Graduate and Post-doctoral Studies
In partial fulfilment of the requirements for the degree of

Masters of Applied Science
(Computer Engineering)



uOttawa

Faculté de génie
Faculty of Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
University of Ottawa

© Robin Tropper, Ottawa, Canada, 2008-2009.



Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 978-0-494-65508-5
Our file *Notre référence*
ISBN: 978-0-494-65508-5

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

■+■
Canada

Short Abstract

A growing number of enterprise applications on the Internet ranging from banking transactions to business management make use of real-time collaboration. Simultaneous access from any device to any set of applications shared among many users is a hot area of research and development.

This thesis designed a thick-client for real-time collaboration supporting the applications development and interoperability. It introduces a new programming paradigm, algorithms and protocols to bring real-time collaboration to a web-based platform. Its component-oriented metamorphic architecture supports a run-time scalable multi-desktop environment connecting client applications through automated remote procedure call and the object request broker pattern while providing new mechanisms for dynamic resource loading.

The new architecture supports unsolicited server control actions on the client using an event model to simulate interruptions and sustained user-activity during network failure. Results obtained validate the correctness of the approach and the feasibility of an extensible web-based platform for real-time collaboration.

Abstract

There is a growing trend of enterprise applications on the Internet ranging from banking transactions to business management to legal activity where Real-Time Collaboration and Unified Communication are dearly needed. Real-Time Collaboration from anywhere, at any time and with simultaneous access to any set of applications shared among many users is both in big demand and a hot research area.

This thesis focuses on the design and implementation of a new thick-client user environment for Real-Time Collaboration based on computing "in-the-cloud" which also supports applications development and interoperability. Achieving this required the introduction of a new programming paradigm, new algorithms and new protocols to bring Real-Time Collaboration to the cloud. Thus a Web-based Component-Oriented Architecture supporting a Nomadic Desktop Platform was designed and implemented.

The novel architecture embodies Metamorphic Object-Oriented JavaScript, a new Browser-To-Browser RPC, a new Recovery from Failure protocol and new mechanisms for Dynamic Resource Loading just to name a few. The new architecture supports server activity based on a model of event based interruptions and sustained user-activity during network failures.

Results obtained with collaborating in-the-cloud on a selection of applications and on various use-cases are given to prove the correctness of the approach and the feasibility of an extensible web-based platform for Real-Time Collaboration.

Acknowledgements

I would like to sincerely thank my graduate supervisor, Professor Dan Ionescu. His far reaching vision was a constant prod to my creativity.

I would like to kindly thank my NCCT Laboratory team mates for their contributions to the project in which this thesis is grounded. Miqdad Jaffer and Michael Lynn implemented the server and communications protocols for the AJIP project. Rabih Dagher designed and built the version of the server application (JAWS) on which the prototype underlying this thesis was built. Bogdan Ionescu, Marcel Ionescu and Mircea Trifan implemented various back-end services for AJIP and the first prototype generation as well as lending their hand to network-intensive use-cases of the second generation. Shahidul Islam built the first video conferencing prototype used in the projects underlying the thesis and also driver for gestural manipulation of client-side RIA which uses multiple pointing devices developed as part of this thesis work. He and Rabih Dagher built support for sending pictures taken from wireless devices. Rabih Dagher, Cristian Gadea and Bogdan Ionescu implemented a map application for hand-held devices with Real-Time Data Markers and asynchronous picture reception which were integrated in the RIA based on results obtained in this thesis work. Lukasz Koszanski adapted the project's user-environment libraries for a smart-phone and was the main developer for the second iteration of the prototype video conferencing suite used in the project. Bogdan Solomon extended the slide-show application with a collaborative YouTube player which includes time-stamped interactive comments using the Real-Time Collaboration mechanisms that resulted from this thesis.

Finally, and most importantly, I emphatically thank my beloved wife, Marjan, for having supported and encouraged me in many ways throughout a considerable part of my undergraduate studies and my entire master's program. It is difficult to overestimate the importance of her role as well as her patience with the time and effort I had to devote to the process.

Contents

Abstract.....	i
Acknowledgements.....	ii
Table of Figures.....	vii
Equations.....	xi
Code Listings.....	xi
Acronyms and Abbreviations Used in the Thesis.....	xiii
Chapter 1 - Introduction.....	1
1.1 Research Objectives.....	4
1.2 Main Contributions of this Thesis.....	5
1.3 Thesis Organization.....	6
Chapter 2 - Related Work.....	8
2.1 Unified Communications (Computing) and Mobility.....	8
2.2 Mashups.....	9
2.3 Drag and Drop (DnD) and Object Association.....	10
2.4 DDE, OLE and ALE.....	11
2.5 Metamorphic Objects.....	12
2.6 Telepresence and Collaboration.....	13
2.6.1 General Architecture for Collaboration.....	13
2.6.2 Design of Collaborative Applications.....	14
2.7 Web-Based Solutions.....	15
2.7.1 Browsers as Support for Distributed Computing, Pros and Cons.....	15
2.7.2 Life Cycle.....	16
2.7.3 Web-Based Client-Server Connection Protocols.....	18
2.7.4 AJAX for Messaging, Data and RPC-based Collaboration.....	18
2.7.5 AJAX Servers.....	20
2.7.6 File and Data Formats.....	21
2.7.7 Web-Based Collaboration and Social Networking.....	21
2.8 Web-Based User Environments.....	21
2.8.1 Windowing Toolkits.....	22

2.8.2	JSR 168: Java Portlet Specification.....	22
2.8.3	Vendor UI tools.....	23
2.9	Client-Generated: Various AJAX libraries.....	24
2.10	Social-Networking Applications.....	25
2.11	Full Web Desktops (WebOSes).....	25
2.12	Web-based Office Productivity Applications.....	27
2.13	Remoting and the Virtualization of Applications and the Workspace.....	28
2.14	Web-Based Programming Support.....	29
2.14.1	MVC : the Key to Scalability and Reusability.....	29
2.14.2	Object-Oriented JavaScript (OOJS).....	30
2.15	Summary of Deficiencies in Previous Approaches.....	31
2.15.1	Appearance and Disappearance of Related Projects.....	31
2.15.2	Sharing and Real-Time Collaboration Issues.....	32
2.15.3	One-to-Many = Incomplete Collaboration.....	33
2.15.4	RIA: a Backwards User Interface Paradigm.....	34
2.15.5	Reuse is Absent from the Design.....	35
2.15.6	No Iterative or Hierarchical GUI Building.....	36
2.15.7	Unidirectional Method Invocability.....	36
2.15.8	No Technology Independent API.....	37
2.15.9	Absent Combination of Supported Technologies.....	37
2.15.10	Not True RT Collaboration, Not on Any Application.....	38
2.15.11	Mashups don't Collaborate.....	39
2.15.12	Inter-Dependent Applications are Full-Screen Singletons.....	39
2.15.13	Programming Language and Software Design Issues.....	40
2.15.14	Conceptual Problems with Popular AJAX Libraries.....	42
2.15.15	Multi-Touch Support $\neq$ Multiple Mice.....	46
2.16	Chapter Summary: Conclusions Drawn from the Related Work.....	46
Chapter 3	- Collaborative WebOS Architecture.....	48
3.1	Overview.....	48
3.1.1	Requirements.....	49
3.1.2	Client-Server Communication Protocols.....	51
3.1.3	Overall Architecture.....	51
3.1.4	Thick Client for a Rich User Environment.....	53
3.1.5	Recovery from failure.....	56
3.1.6	Application Life Cycle for Nomadic Computing.....	59

3.2 General Use-Cases Supported.....	62
3.2.1 End-User Cases.....	64
3.3 Specific Architecture of the Client Side.....	65
3.3.1 Components of the User Environment.....	66
3.3.2 Components for Environment Management and Communication.....	67
3.3.3 Components of the Windowing Toolkit Proper.....	69
Chapter 4 - Adjusting the Paradigm of Web-Programming.....	71
4.1 Complete Object Orientation in JavaScript.....	71
4.1.1 Improving JavaScript Inheritance.....	72
4.1.2 Static Members and Classes.....	77
4.1.3 Multiple and Removable Inheritance.....	77
4.2 Standardized MVC.....	82
4.2.1 Communicate Data, not HTML Content.....	83
4.2.2 Thick Client: Roles.....	84
4.3 Component Orientated Programming.....	86
4.3.1 Scope Decomposition of Sub Components.....	87
4.3.2 Hierarchical HTML.....	90
4.4 Contract Object.....	91
4.5 DnD: Data Transfer & Event Firing.....	92
Chapter 5 - Collaboration and Interoperability: Architecture and Algorithms.....	94
5.1 States of Application Visibility (Collaborating, Shared, Public).....	94
5.2 Access and Indexing.....	96
5.3 Browser-to-Browser Client-Centric Collaboration.....	97
5.3.1 Relay Architecture.....	97
5.3.2 Many-to-Many-on-Many Relationship Collaboration.....	98
5.4 Observer Pattern: Cornerstone in Collaboration.....	99
5.4.1 Notification Object.....	99
5.4.2 Single Data, Multiple Displays (SDMD).....	100
5.4.3 Remote Observer Pattern Between Web Clients.....	101
5.5 RPC Between Browser-Contained Clients.....	101
5.6 Deferred Synchronicity and Sequential Continuity.....	104
5.7 Event Replication or Echoing Actions.....	105
5.7.1 Automation of Action Replication.....	106
5.7.2 Mashups, Aggregation and Real-Time Collaboration.....	108
5.8 Browser-to-Browser ORB.....	117

5.9	Sets of Inter-Dependent Applications.....	118
5.10	Dynamic Properties.....	120
5.10.1	Dynamic Resource Loading and Application Definition.....	120
5.10.2	Persistence & Metamorphism: Dynamic Data.....	121
5.11	Scalability to Multiple-Technologies.....	124
5.11.1	Interoperability Between Technologies and DnD.....	124
5.11.2	IFRAME injection.....	127
Chapter 6 - Results: Validation of the Design.....		130
6.1	Answer to the Deficiencies in the Literature Review.....	130
6.2	Features of NIWT and CSOS.....	137
6.2.1	Three Kinds of NIWT.....	137
6.2.2	Containers, Layouts and Mashup.....	138
6.2.3	Utilities.....	139
6.2.4	SDMD & DnD.....	140
6.2.5	Advanced DnD Use-Cases.....	140
6.2.6	Performance of Dynamic Resource Loading.....	141
6.2.7	Recovery From Failure.....	141
6.3	Applications Developed.....	143
6.3.1	Applications in Different Technologies.....	143
6.3.2	Applications and Component Oriented Programming.....	144
6.4	Sets of Inter-Dependent Applications.....	144
6.5	Comparison with the Current State-of-the-Art.....	146
6.5.1	Feature Comparison.....	146
6.5.2	Performance Comparison.....	149
6.6	Third-Party Integration.....	151
6.7	Interoperability and Mashup.....	152
6.8	Sharing and Collaborating.....	155
6.8.1	Applied Real-Time Collaboration, Locally Synchronized Copies and OLE.....	158
6.9	ORB Between Remote Web-Pages.....	161
6.10	Hand-Held Devices.....	163
6.11	Unified Communications.....	166
6.12	Drawbacks with the Results.....	166
6.12.1	Caching and Memory Leaks.....	166
6.12.2	Cross-Domain Interoperability.....	167
6.12.3	Reusing Third-Party Software	167

6.12.4 Browser Compliance Issues.....	168
6.12.5 Web MVC not Automatic.....	168
6.12.6 Weak OOJS.....	169
6.12.7 The Set Manager is Needlessly Complex.....	170
6.12.8 RPC Returns and Type-Constraining.....	171
Chapter 7 - Conclusion.....	172
7.1 Concepts Addressed in this Thesis.....	172
7.2 Contributions of this Thesis.....	173
7.3 Future Research.....	175
Summary of Appendices:.....	176
References.....	179

Table of Figures

Figure 2.1: GUI for Virtual Collaboration Room.....	13
Figure 2.2: Architecture for Virtual Collaboration Room.....	13
Figure 2.3: Using single-user applications for collaborative purposes. [46].....	14
Figure 2.4: Legacy web service life cycle. From 179.....	17
Figure 2.5: AJAX web services Life cycle. From [2].....	17
Figure 2.6: Example layout for portlets.....	22
Figure 2.7 : Bindows full WIMP in COP design.....	26
Figure 2.8 : Widgets and Utilities in Bindows.....	26
Figure 2.9: Screenshot of DesktopTwo.com.....	27
Figure 2.10: Sun VDI allows remote access to any Windows or UNIX based OS from virtually any device.....	28
Figure 2.11: Collaboration Relationships.....	34
Figure 2.12: "AJAX" libraries call external functions, neither the reverse nor event notification.....	36
Figure 2.13: Google Maps: three applications in one.....	40
Figure 2.14: High network traffic from tight knit client-server coupling. 179.....	43

Figure 3.1: UC-IC for nomadic and ubiquitous computing.....	48
Figure 3.2: Server and client side modules of UC-IC.....	52
Figure 3.3: Building-blocks of the UC-IC client and server.....	53
Figure 3.4: Application for collaboration.....	54
Figure 3.5: Ideal package organization for the client side.....	54
Figure 3.6: Communications Channels Between the Major Components.....	56
Figure 3.7: FSM for connected/disconnected activity.....	58
Figure 3.8: Life cycle of a NIWT application.....	60
Figure 3.9: Use-case driven deployment.....	62
Figure 3.10: Administrator revoking ongoing permission for using an application.....	63
Figure 3.11 : Classes of End-User Use-Cases.....	64
Figure 3.12 : Standard Application Use-Cases.....	64
Figure 3.13: Communication Use-Cases.....	64
Figure 3.14: Principal components of the user environment.....	67
Figure 3.15: Components for environment and use-case management.....	68
Figure 3.16: Most important NIWT components.....	70
Figure 4.1: Inheritance and implementation.....	72
Figure 4.2 : Prototypal inheritance.	73
Figure 4.3 : Factory Pattern Inheritance.....	73
Figure 4.4: Plug Object and JS class (closure).....	79
Figure 4.5: Plugging augments a class with its own members.....	79
Figure 4.6: Unplugging removes the Plug's members from a class.....	79
Figure 4.7: An instantiated object using a Plug on another instantiated object.....	80
Figure 4.8: Active Plug.....	80
Figure 4.9: Active Plugs allow execution of specified actions.....	81
Figure 4.10 : Sequence for changing a superclass.....	82
Figure 4.11: Clean MVC separation in facilitates communication with and control from the server.....	83
Figure 4.12 : Content that would likely be sent as HTML, not XML.....	84
Figure 4.13 : Composition tree of a typical dot.NET office application.....	87
Figure 4.14: Inherited global-scope arrays of pointers to objects.....	88
Figure 4.15: FSM for DnD.....	93

Figure 5.1: Application States.....	95
Figure 5.2: FSM for Application Updating.....	95
Figure 5.3 : Modules involved in BTB collaboration.....	98
Figure 5.4: Client-side architecture to support the M3 relationship.....	98
Figure 5.5 : Observer Pattern using the NotificationObject.....	100
Figure 5.6 : SDMD Class Diagram.....	101
Figure 5.7 : Locally synchronized copies of one application shown on all desktops.....	109
Figure 5.8: OLE support through LSC and Container Independence.....	109
Figure 5.9: Example use-case map for BTB collaboration on a composition tree.....	111
Figure 5.10 : Cascading Notification.....	112
Figure 5.11 : Broken Notification.....	112
Figure 5.12 : Full Notification.....	112
Figure 5.13 : Hybrid Notification causes redundancy.....	112
Figure 5.14: Event Notification Classes.....	114
Figure 5.15 : Infinite notification loop.....	115
Figure 5.16: UC-IC's Algorithm for synchronizing both locally and remotely avoiding endless loops.....	116
Figure 5.17: Sets of Inter-Dependent Applications.....	120
Figure 5.18: Algorithm to load application definitions on demand only.....	121
Figure 5.19: Dynamic data defined by inheritance and composition.....	123
Figure 5.20: Dynamic Reconstruction of Metamorphic Components.....	123
Figure 5.21 : Event model for DnD with embedded technologies.....	126
Figure 5.22 : The state of mouse buttons allows assumptions on DnD.....	127
Figure 5.23: IFrames are external pages merely embedded in the main page.....	128
Figure 5.24: Injecting code for interoperability with IFRAME RIA.....	129
Figure 6.1: Workspace layouts: tiled and cascading desktops.....	138
Figure 6.2: Three uses of the Draw Utility.....	139
Figure 6.3 : Using DnD delegation for cross-desktop mashup.....	140
Figure 6.4: Multi-Book: 3-level sub-composition.....	144
Figure 6.5: Google maps compared with two sets of inter-dependent applications	145
Figure 6.6: NIWT Pointers showing DnD states.....	152

Figure 6.7: Gestural Manipulation with NIWT Pointers.....	153
Figure 6.8: Gestural Mashup 1: pictures integrated to map.....	154
Figure 6.9: Gesture Mashup 2: integrating XML data.....	154
Figure 6.10: Gestural actions and interoperability.....	154
Figure 6.11: Sharing and public applications.....	156
Figure 6.12: Starting collaboration on a word processor.....	157
Figure 6.13: Real-time collaboration between two users on a word processor.....	157
Figure 6.14: Slide Show Application: Collaboration and Interoperability.....	158
Figure 6.15: Simultaneous locally synchronized copies and real-time collaboration with 3-level delegation.....	160
Figure 6.16: Other user's Application Manager using ORB.....	161
Figure 6.17 : Full user workspace on a smart phone.....	163
Figure 6.18: Sending geo-located picture to iPhone extracts of NIWT/CSOS.....	165
Figure 6.19: Image displayed on iPhone extractions of NIWT/CSOS.....	165
Figure 6.20: Cell-phone sending a picture to other UC-IC users.....	165

Equations

Equation 1: Event Contract.....	91
Equation 2 Event Replication.....	106
Equation 3: Event Redundancy.....	113
Equation 4: Redundancy Evasion.....	113
Equation 5: Multiple notifications received per unique event.....	113
Equation 6: Independence in Concurrent Application Sets.....	119

Code Listings

Listing 4.1: Finding a superclass definition of a function in NIWT components.....	76
Listing 4.2 : Scope decomposition for an item in a form in a panel in a window.....	89
Listing 4.3: 'this.someVar' will show 'undefined'.....	90
Listing 4.4: C++ Header-Style Closure Definition.....	90
Listing 4.5 : Building the Event Contract.....	92
Listing 6.1: Looping on a volatile variable.....	169
Listing 6.2: Contract with return and data typing.....	171

Acronyms and Abbreviations Used in the Thesis

AJAX: Asynchronous XML and JavaScript. A feature whereby the browser must support XML parsing, construction and edition as well as support for the standard on the XMLHttpRequest object [1][2][4]. The popular use of the word refers to libraries of graphical interface components built in DHTML and CSS.

ALE: AJAX Linking and Embedding.

API: Application Programmer's Interface.

CPU: Central Processing Unit.

CSOS: Client-Side Operational Structure.

CSS: Cascading Style Sheets.

DHTML: Dynamic HTML.

DnD: Drag and Drop.

DTD: Document Type Definition.

DOM: Domain Object Model.

GIS: Geographic (or Geodesic or Geodedic) Information Systems.

GUI: Graphical User Interface.

GWT: Google Web Toolkit.

HTML: Hyper-Text Markup Language.

JAWS: Java/JS Asynchronous Web-enabled System.

JEE: Java Enterprise Edition.

JSP: Java Server Page.

LIFO: Last one In, First one Out.

M3: the Many-to-Many-on-Many relationship.

MSDN: Microsoft Developer Network.

MVC: the Model View Controller software design pattern.

NIWT: Ncct Internet Windowing Toolkit.

OLE: Object Linking and Embedding.

OO: Object-Orientation (Object-Oriented, etc.).

OOP: Object-Oriented Programming.

OOJS: Object Oriented JavaScript.

ORB: Object Request Broker [124].

OS: Operating System.

P2P: Peer to Peer.

PC: Personal Computer (including MacIntosh, Linux etc.).

PDF: Portable Document Format.

RIA: Rich Internet Application.

RMI: Remote Method Invocation.

RPC: Remote Procedure Call. This will include RMI: Remote Method Invocation.

RSS: Really Simple Syndication.

RT: Real-Time.

SDMD: Single Data Multiple Display.

SOAP: Simple Object Access Protocol.

UC: Unified Computing (or Communications).

UC-IC: the project underlying the thesis. (Not an acronym but a rebus.)

UI: User Interface.

W3C: World Wide Web Consortium [5].

WIMP: the Windows Icons Menus and Pointers paradigm for user interfaces [6].

XML-RPC: a standard for Remote Procedure Calling using XML marshalling [7].

Chapter 1

Introduction

People and businesses are dearly seeking applications that bring together different activities in a collaborative space. For example, chat utilities should be able to display video clippings and serve to make phone calls. Geographic Information Systems (GIS) should accommodate pictures, sound-tracks, memos as well as tourism and security information. Mobility is an expected part of professional practice in most fields. Web-based solutions allow users to access their work in a nomadic fashion; that is, by using whatever computer is handy where they happen to be rather than needing to carry a laptop. Web-based solutions also open the door to connection with a vast pool of other people who are also using it while requiring nothing more than a mainstream browser application. The emergence of Social Networking and Collaboration Environments (extending Virtual Meetings) has spiked consumer interest and new products from trusted industrial players have started to become available.

On the perspective of application development, despite past promises, the engineering principle of reuse seems to have failed as much as it has succeeded in the industry for both distributed and local systems.

«The heterogeneity of hardware architectures, the diversity of OS and network platforms, and stiff global competition are making it increasingly infeasible, however, to build networked applications from scratch with the following qualities: Portability, [...] Flexibility, [...] Extensibility, [...] Predictability and efficiency, [...] Reliability, [...]» [10]

This makes a compelling argument for web-based systems, where the compliance to standards by hardware and Operating System (OS) vendors as well as the existence of browsers for the most important Personal Computer (PC) and hand-held device platforms seemingly nullify the remarks above. The success of distributed application platforms

depends on using well established and trusted component frameworks such as Java Enterprise Edition (JEE), using software design patterns, modelling and maintaining a close feedback loop between middleware development and application development. Rather than a top-down approach, generalizing applications into abstract components apparently results in better reusable components [10].

A ‘many-to-many-on-many’ (M3) relationship in real-time collaboration is a new model for aggregated servers which naturally emerged from work underlying this thesis. M3 describes a connection graph whereby many applications can be part of the same collaboration session while any individual application can be part of several sessions and where the sessions can link users from many vendors of real-time collaboration services.

Interoperability on the same environment implies that any component can respond to changes in the state of any other component be it the result of user interaction, a server-side process or even a routine originating from a remote host. In terms of Rich Internet Applications (RIA), this requires that any component be capable of handling events fired by another component, of enquiring on the value of its member variables and of calling its member functions. Because several technologies can make-up the web-page, an Application Programmer’s Interface (API) is required for such capabilities. Though Java Applets, Flash Objects and some media players provide libraries for communication with JavaScript, application developers must still define the actual API. There is currently no library available to provide inter-technology APIs for commonplace issues like event capturing and handling, Drag-and-Drop (DnD) or messaging, let alone any mechanism to automate them when definitions they on data and state.

Asynchronous JavaScript and XML (AJAX) is an important technology for the development of RIA, increasingly used and recommended by the most noteworthy players in the field of distributed and collaborative computing. It gives the browser the ability to support an application life cycle which more closely resembles that of applications on a desktop computer while opening the doors to providing services from multiple domains. Its emergence as a platform for the desktop metaphor, however,

requires work in the fields of programming paradigms in order to provide the power and flexibility required by more complex user-environments and applications.

Above simple message passing, collaboration between components on remote clients implies the ability to synchronize the state of a component on one host with that of the corresponding components on other hosts. Current protocols for Remote Procedure Call (RPC) and the like are geared towards clients calling server functions and Peer-to-Peer (P2P) implementations focus on data streaming. A new design of such protocols is therefore necessary to ensure that actions performed on one client are replicated on many others, most importantly the new protocol for Browser-to-Browser Remote Procedure Calling (browser-to-browser RPC). The automation of such replication currently does not exist for RIA; it is therefore presented here as the new implementation of the contract object.

In most Programming Paradigms, objects (instantiated classes) have a fixed nature with little room for change beyond the contents of any member array they may possess. But there is ever more mention of ‘mashable applications’, a principle which aims to allow users to create their own applications by assembling various parts. Taken to the extreme, this implies that applications can be created and modified dynamically at run-time. Any given application or component can exhibit different behaviour patterns in different contexts; for example, one Window might require DnD capabilities for a picture-book whereas another must prevent any modification of its contents. Another example: a rich text editor application in one instance be an independent window, while in another instance it is part of a more complex slide-show editor. All these changes are called ‘morphology’ and can be expressed in terms of run-time changes to an object’s structure, definition, and even inheritance. Architectures, class definitions and programming paradigms capable of supporting such run-time morphology are called ‘metamorphic’ [37].

1.1 Research Objectives

This thesis focuses on designing and implementing a new thick-client architecture that supports real-time collaboration as well as local interoperability for any application that can be contained in a web document while ensuring the protocols and the programming paradigm required to make it possible. Success implies a user-environment and a platform for the development of RIA from reusable components and tools whereby real-time collaboration, interoperability and control from the user environment for DnD as well as the persistence and restoration of the application state are built-in.

This requires careful study of:

1. Reusable architecture models and existing resources for RIA development in order to integrate solutions that answer the requirements or improve on deficiencies.
2. Real-time collaboration platforms, protocols and algorithms (web-based and non). Again, reuse is the goal, but lessons learned can also prepare a solution to the requirements.
3. Automation of algorithms and reuse by inheriting from the resources that implement them in order to avoid having to reprogram the same routines with every new application.
4. The theory of programming paradigms leading to support for scalable complex and metamorphic architecture.
5. Means to aggregate data fragments of various formats (txt, img, video, etc.) to support the persistence and restoration of the data and state of applications which combine the formats as well as automating collaboration mechanisms for them.

The client-side architecture proposed:

1. Supports the reusability of components by following a methodology of incremental augmentation as well as the persistence and restoration of components with tolerance for runtime morphologies.

2. Implements the client-side mechanisms to support collaboration with a M3 relationship in terms of linking sets of applications to sets of sessions independently of the collaboration service vendor.
3. Implements a web-based user-environment characterized by server autonomy while reacting to asynchronous, unsolicited messages and/or commands from the server.

1.2 Main Contributions of this Thesis

1. A new programming paradigm for the development of browser-contained applications bringing to them a Java-like, Object Orientation applying the Model View Controller design pattern.
2. New algorithms and the implementation of new support mechanisms for the run-time metamorphism of structure, interface and implementation in web-applications along with the automated persistence and restoration thereof.
3. The design and the implementation of a very thick client comprising a new web desktop metaphor which includes advanced features and use-cases from native OS desktop environments.
4. New APIs and protocols between the server and web-client (browser) for truly asynchronous control on any part of the user environment from the server, the detailed and full persistence and restoration of its state and recovery from failure.
5. A new definition of real-time collaboration, for client-to-client (browser-to-browser) interaction and extending it to applications of random purpose along with the mechanisms to render collaboration transparent to application developers.
6. A new design and the implementation of supporting mechanisms for automated interoperability between different applications built in the same or different technologies.

1.3 Thesis Organization

The research underlying this thesis centred on programming paradigms required to address collaboration protocols and sound design methodology while implementing middleware to build a prototype user environment. The organization of the remainder of this thesis will reflect a progression that explains the various steps in the underlying research.

Chapter 2 will provide an in-depth review of important related work, both academic and industrial. It will also draw attention to deficiencies in the previous approaches and conclude with a summary that justifies this thesis research.

Chapter 3 will provide the architecture and use-cases for the client-side (the topic of this thesis). It will also explain the life cycle of an application built with the NCCT Internet Windowing Toolkit (NIWT) and support for nomadic computing, network-independent user activity, asynchronous messaging and server-based environment control and recovery from network failure, thus laying the groundwork for the following chapters.

Chapter 4 will explain the proposed changes to the current programming paradigms in RIA development which better support reusability, scalability, robustness and run-time metamorphism. The above includes discussions on advanced topics concerning object-orientation and design patterns.

Chapter 5 will discuss the algorithms and components required to implement the protocols for real-time collaboration including browser-to-browser RPC, event replication, locally synchronized copies as well as user-environment concerns such as environment restoration, dynamic resource loading and interoperability. Specific concerns regarding distributed systems design in AJAX will also be addressed.

Chapter 6 will describe the results in terms of the prototype built and how it demonstrates the use-cases and requirements listed in chapter 3. Further experiments hand-held devices will also be described, showing the scalability afforded by the results

of this thesis work and drawbacks from the solutions proposed will also be discussed.

Chapter 7 will conclude the thesis and suggest areas for future research.

This thesis is supported by a number of appendices, listed after chapter 7, which are available on request. They provide additional details regarding the research including some explanations of obstacles overcome, longer code listings, expanded UML diagrams, more detailed use-cases, more examples of applications, a brief history of the UC-IC project and a description of future potential offered by this thesis.

Chapter 2

Related Work

Research in real-time collaboration has been ongoing for a considerable number of years on many platforms including full operating-systems, middleware appliances, specific application suites dedicated to real-time collaboration and even machine virtualization and remoting. The lessons learned from these are valuable whether they are web-based or not. This chapter will therefore review related work from both academic and industrial sources. It will also examine the state of the art for the various technologies on which the results of this thesis were built. Deficiencies found in the related work will then be discussed.

2.1 Unified Communications (Computing) and Mobility

Unified Communications (UC) and computing tools for multiple data formats is still in high demand as evidenced by ongoing calls for the design interfaces, environments and the required mechanisms to support the unification of data in communication [12][13].

According to Cisco:

Unified Communications [...] combines all forms of business communications into a single, unified system that provides powerful new ways to collaborate.

[14]

This idea has been expressed since 1998 [15] to assemble the audio-visual streams of teleconferencing with the familiarity and ease of telephony as well as static media such as formatted text and images all into one source of communication [16]. The reason for this being that people use devices for multiple purposes: a cellphone for telephony, a PDA for planning the activities, a television, sound system and radio for entertainment, fax for document copies and finally a computer for office productivity, messaging and e-mail. Carrying separate devices for each purpose becomes cumbersome and purposes also

overlap: telephony requires a phone book, e-mail requires an address book and web-surfing requires a list of bookmarks and so on. As the physical devices are becoming more powerful, they now assemble some of these functions together. Several different applications can run on the same device to accommodate the different media formats, but few applications really allow the combination of them together. For businesses to offer unified communications as a service is not trivial. There are two ways to deliver the service: client-based and server-based, from each of which more strategies can be derived to support specialized services such as security and message box abstraction [15].

Mobile computing is not limited to wireless hand-held devices, but also includes the ability to use a workspace on one computer in one location, exit and resume the work on another computer in another location. This kind of use is called ‘nomadic’ and has been equated with mobile computing in [17]. To provide this, a system must grant the user access to data and settings as well as the state of her/his last activities regardless of the host used

2.2 *Mashups*

Mashups aggregate information and services from a variety of sources on the Internet [18]. Though typically displayed and organized on a map, it can be any aggregation of data from one or more sources combined in any customized UI [19]. They are an important aspect of the ‘web 2.0’ phenomenon and comprise mostly RT data from RSS feeds. Their composition model can be based on either data-flow, layout design or events model. Data-flow is the case where several streams of data are aggregated (often in a pipe-and-filter engine) that can sometimes be customized with layout designs. The third model, event-based, is the rarer case where parts of a mashable application fire events that are caught by other parts which react according to their design [18].

Web-based mashups can be designed by either application developers or end-users. Yahoo Pipes [27][28], Microsoft Popfly [29], for example, create mashups according to user preferences for data extraction (by topic, location etc.). A Graphical User Interface

(GUI) allows users (end-users or web-site designers) to connect RSS feeds and the preferences configure pipe-and-filter mechanism. The final result is a mashup from various sources tailored to user's desires. The Google Mashup Editor [30] is similar, but provides a list of supported sources (Google Base) and links the mashup to Google Maps for a geo-centric information engine.

In all cases mentioned above, the creation of mashups can be automated by providing an Excel-type spread-sheet or XML document which describes the data in the specified format. Mozilla professes a kind of mashup which they call 'ubiquity' [31], though it more closely resembles Dynamic Data Exchange (see section 2.4) by extending the power of the Firefox browser itself. This proposed technology aims to empower users by enabling them to assemble existing web-services from different vendors into a single container (web-document).

2.3 Drag and Drop (DnD) and Object Association

In Java, DnD is all about transferring data from one object to another through mouse manipulations by the user [21]. Moving a given GUI component is only the means by which a user associates the drag-source (source of the data transfer) and the drop-target (receptor for new data).

AJAX libraries, on the other hand, focus almost exclusively on the ability to use the mouse for moving GUI components [22]. As the emergence of reusable functional libraries is relatively new to Dynamic HTML (DHTML), drag and drop must be (re-)coded every time by the web developer. The ability for a drop-target to accept or reject a given drag-source is supported by some libraries (e. g. [23]), but the whole process both depends on, and is defined in, the HTML itself.

Mozilla agrees with the data-transfer value of the DnD Experience [24] and implemented a 'dataTransfer' object accessible through mouse events fired. This object supports adding and removing information data and also provides default data according to the type of HTML element on which the mouse-down event is fired (e. g., an IMG tag will

provide the URL). However, this construction is still based on HTML elements which this thesis will show to be a hindrance to building sophisticated applications and environments.

Though the Microsoft Developer Network (MSDN) also defines DnD in terms of user-mouse actions, their tutorial describes mostly data transfer mechanisms [25]. Microsoft also provides mechanisms to use DnD for the creation of links between applications (see section 2.4 below). This improves the user experience by allowing the user to dynamically select pieces from one application and port them into another, sometimes launching a given routine[33].

2.4 DDE, OLE and ALE

Microsoft had started a powerful trend with its Dynamic Data Exchange (DDE) Protocol, whereby applications could exchange messages via shared memory on a discreet or streaming basis [32]. The user experience best known for this is the clipboard metaphor which supports the cut, copy and paste actions [33]. OLE continued this innovation by incorporating pieces of one application into another one by the transfer of links to objects [26]. This keeps in line with a trend started as early as 1995 to build new applications as aggregations of existing components [34]. This structure suggests that applications might include components from other applications, either local or distributed.

The main difference between OLE and DDE is that, rather than using a shared memory space with a limited number of functions, OLE creates a link between two applications whereby any set of specified actions can be taken on given events and conditions. The use of OLE has become so widespread, that vendors now provide libraries to enable OLE between applications built on the evermore popular Java platform and Windows applications [35].

- 、 The latest innovation in terms of object association needs to be mentioned for its web-

potential: ALE¹ (Ajax Linking and Embedding) is an attempt to replicate OLE in RIA [36]. It is a specification whereby a RIA from any domain can be embedded in an e-mail or editable document.

2.5 Metamorphic Objects

The notion of metamorphic objects is new and fleeting. It is not easily found in academia or in industrial literature. This thesis will show that it is nonetheless a crucial aspect in attaining all the goals in the project underlying the thesis. According to [37], metamorphism:

«[...] implies that an object, as identified by a unique object reference, can change its interface definition, class definition, implementation, even its location, and still remain the same entity.»

This metamorphism can be seen in how applications installed on a PC evolve over time, especially when they make use of services and resources found on the web. Work on the subject of dynamic reconfiguration of resources can be found in the field of distributed computing [135], notably where CORBA is involved [130][131], in OOP theory [132] and database design [133][134].

The focus of this thesis being applications whose composition, interface definition and implementation change at run-time as the result of user action on such applications, the model of metamorphic objects becomes powerful; real-time collaboration on these applications highlights the urgency in supporting metamorphic properties. If these morphologies can be expressed in the programming paradigm itself as well as in the protocols for collaboration, persistence and restoration, then many automated mechanisms can be abstracted and built-in to the platform in a way that greatly facilitates the development of new applications.

¹ Remark that there is a contention with the acronym: it also stands for Application Level Event which is apparently widely used and implemented. [36]

2.6 Telepresence and Collaboration

Telepresence, or virtual presence, is any technology that will enable users from remote locations to interact in a manner that gives the impression they are together. Collaborative applications, virtual meeting rooms, video conferences and even simple chat tools reduce the need for transportation, thereby reducing the costs of time and fuel consumption among others. They are thus being touted by vendors as an environmental solutions which also improve business practices [39]. This section surveys some of the most important endeavours in this field.

2.6.1 General Architecture for Collaboration

The Virtual Collaboration Room (VCR) is a web-based co-authoring environment that uses the room metaphor [45]. The UI (Figure 2.1) grants the use of a variety of applications concurrently with other users while organizing important information in panels. An important feature of VCR is ‘awareness’: RT updated data on the objects and users. Awareness of objects concerns its existence, size and other visual properties whereas user awareness concerns the online/offline status of other users as well as their actions and personal information on them.

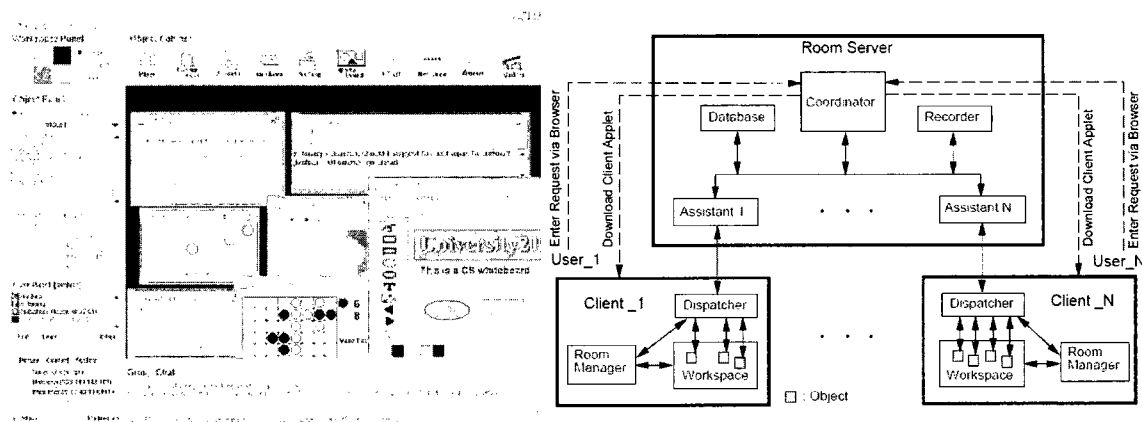


Figure 2.1: GUI for Virtual Collaboration Room. Figure 2.2: Architecture for Virtual Collaboration Room

To accomplish this, VCR implements an architecture of components (Figure 2.2) dedicated to collaboration on both the client and server sides. Server-side ‘assistants’

manage users' activities on objects (one assistant per user); the assistants are therefore the server-side collaboration managers on any given application. The entire VCR is implemented in Java and the UI is entirely contained in an Applet.

2.6.2 Design of Collaborative Applications

The coding requirements and available (or unavailable) tools for the implementation of collaboration software have significant impacts on the practice of application development itself. If a platform for collaboration aims to support the rapid development of new applications for random purposes, then an API must be provided for programmers who may not be used to considering all the implications. Ideally, coding collaborative applications should be the same as coding single-user applications [46].

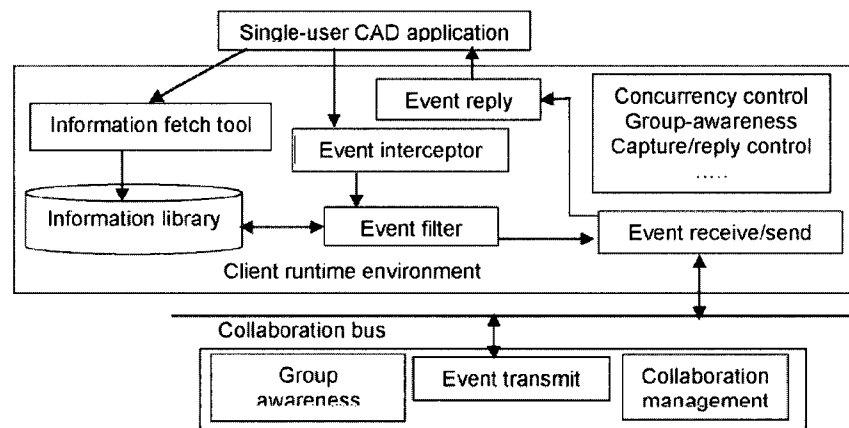


Figure 2.3: Using single-user applications for collaborative purposes. [46]

Figure 2.3 shows how applications designed for single-user can be turned seamlessly into collaborative applications. All the application must do is fire events that can be captured by an environment which has a collaboration manager (distributed over both client and server). The collaboration manager will provide all the requisite mechanisms and apply these events on the environments of the other users transparently to the programmer.

2.7 Web-Based Solutions

As the Internet is increasingly becoming the most important medium for collaboration environments, with the web-browser its *de facto* UI, this section will examine issues related to programming for it.

2.7.1 Browsers as Support for Distributed Computing, Pros and Cons

Browser applications are believed to be inherently ‘cross-everything’ applications and thus simplify the development of any distributed system [47]. In fact, each browser is responsible for defining its own interpreters of and layout engines for HTML, Cascading Style Sheets (CSS) and ECMA (the standard for JavaScript). The result is that developing a compelling GUI with powerful functionality is not straight-forward as might be expected from a platform such as Java etc. Many browsers claim full support of W3C standards, yet they still render differently, making web development increasingly challenging. The increasing popularity of new browsers only intensifies the challenge.

Browser-based systems are believed to be always easier to install and upgrade [47]. In fact, this depends heavily on the components required by the browsers to perform required tasks. Some applications depend on Java, dot.Net, Flash or other technologies which have all undergone significant changes in the past few years. There is no guarantee as to clients updating these technologies and backwards-compatibility is scarcely ensured. For example, programs developed with Action Script 2 are incompatible with others developed in Action Script 3 making reusability difficult.

It is believed that browsers have robust and well-considered security models [47]. This is naïve as a subscription to any technological mailing list will quickly testify. Using HTTPS will go a certain length in securing the communication lines [48], but it does not guard against misuse and inadequate design and programming.

All in all, there are use-cases that are well suited to browsers and others that are not.

Using browsers to access a web-based SOA has the enormous advantage of giving licensing control to the user. Whereas installed applications charge licensing fees per copy installed, browser access to web-based services do not. Such services can implement a per-use strategy monitored with access guards (user accounts and login tools), but it still leaves the control in the hands of the user.

Configuration management remains a good reason to use a browser-based client. Apart from client-side dependencies (e. g. JRE and Flash player versions), the browser acquires its resources at every system access, ensuring that the latest material is always used. In other words, (barring some caching issues) dynamic reconfiguration [48] is automatically supported.

Browsers less adequate when timely responses are required. Using browsers adds a layer of computation and handling, in particular in response to keyboard and mouse events as well as audio and video input/output [47][49]. Furthermore, speed optimization mechanisms such as Threads are much more difficult to develop. Where fast start-up times are required, browsers pose a problem: apart from the handshake with the web services, loading the various resources is time consuming, especially where dynamically generated content is involved. Where the system must communicate with and through multiple servers, protocols and browsers, system development becomes very heavy and prone to malfunction.

Finally, [47] suggests that careful attention to visual real estate is of the essence. Though this thesis will disprove the notion that complex GUIs are ill-suited to browsers, the paper does make a good point that portability to hand-held devices is not ensured. Other human factors, like a user's likelihood of using the browser's navigation buttons, can seriously infringe upon the system's expected state.

2.7.2 Life Cycle

The typical life cycle of any web-document follows the paradigm of user request and server response whereby a web page will call a server that will replace the entire page

with the contents obtained by the URL (Figure 2.4). This is a fully synchronous and stateless scenario where business logic is difficult to implement. [2]

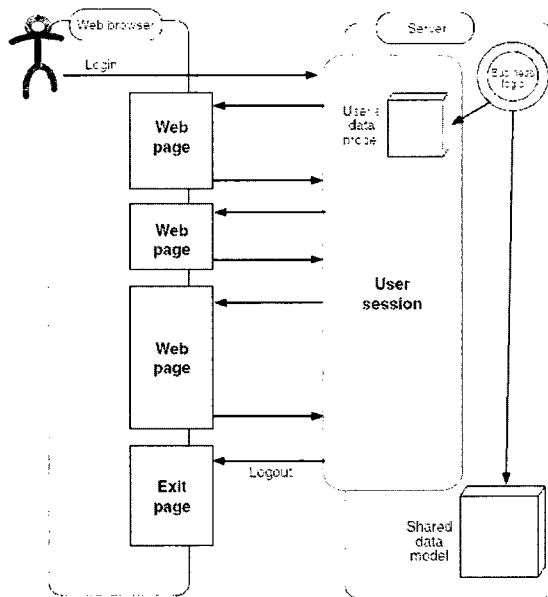


Figure 2.4: Legacy web service life cycle. From 178

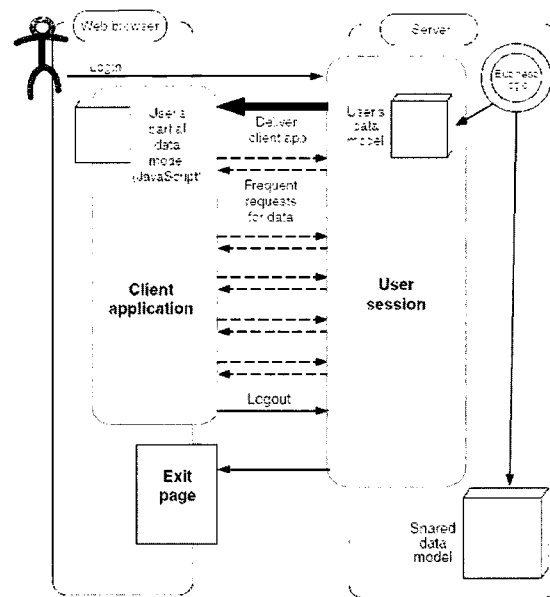


Figure 2.5: AJAX web services Life cycle. From [2]

The biggest attraction of AJAX is the ability to obtain a response to a server call and perform some random action with the contents obtained. The entire server call is kept in an XMLHttpRequest and the response is delegated to a handler function. In its simplest form, a server request will respond with HTML content that will be appended to or replace an existing element in the page. The life cycle is now more dynamic (Figure 2.5) and it becomes asynchronous insofar as the user can keep acting on the page and scripts can continue to execute while waiting for the server response. [2]

More advanced use of AJAX begins when XML data is built and sent along with the server call, the response from which is a string of characters or arbitrary content. When that string is HTML, it can be inserted directly into the document. If it is XML, AJAX compliance requires the browser's JavaScript engine to provide a Domain Object Model (DOM) parser and an XML serializer which afford JavaScript with the transformation of XML documents into strings and vice-versa. These capabilities pave the way to substantial functional power. [2]

2.7.3 Web-Based Client-Server Connection Protocols

The AJAX life cycle explained above allows a more independent and dynamic client-side to be built in a web document, but does not provide the server any ability to send unsolicited messages to the client. In other words, AJAX is only asynchronous in terms of the client making requests: the server must still wait for a request before sending anything to the client.

There are two protocols that overcome this limitation: HTTP Streaming and AJAX Push. In HTTP Streaming, the connection between the browser and the server is essentially kept alive until the client navigates away from the page. This allows the server to keep outputting data, commands, messages etc. to the web document. In the AJAX Push protocol², the client request and server response protocol is terminated as usual, but the client immediately sends a waiting state request to the server thus re-establishing the connection between them. This request/response sequence is suspended until messages to the server or other variable time business logic requires action to be performed by the server on the client [104]. Though AJAX over the standard HTTP request/response paradigm introduces the possibility of designing a stateful system on the client side, HTTP-Streaming and AJAX-Push allow the design of a stateful system on the server or client or both. This is the foundation of asynchronous communication and control between the server and the client as well as between clients (with the server as a relay) for the architecture and protocols presented in this thesis. Most importantly, AJAX Push facilitates the detection of connection failure which is a crucial part of the algorithm proposed for recovery from failure.

2.7.4 AJAX for Messaging, Data and RPC-based Collaboration

Collaboration can be achieved by applying routines to remote objects in order to mirror routines that occur locally. This is termed Remote Procedure Call (RPC) and is implemented by the Simple Object Access Protocol (SOAP), Java's protocol for Remote

2 Also known as 'Comet' or 'Reverse AJAX' [122].

Method Invocation (RMI) and other similar protocols. AJAX is a feature by which Browsers and Web-Servers can communicate using the eXtensible Markup Language (XML): a character-based, self-describing data format designed specifically to communicate data between applications [7][44] regardless of programming language or platform.

SOAP [136] has become an important player for its ability to provide the benefits of RMI in web-based distributed systems. It is, however, criticized for its large markup design and marshalling which are computationally intensive [40].

XML-RPC is a standard for transmitting execution instructions to remote entities [7]. As with AJAX, the standard details how the XML leaves implementation on the library developer. Most often, XML-RPC is used by a client to call a method on a server [40] [41], and most of those calls concern queries to databases, for example XRPC and XQuery [40].

Currently, all support for web-based RPC is defined as a client invoking a method on a server [40][41][42], no examples have been found where an application invokes a method on another client. Rather, collaboration implementations are mostly data-centric; that is, each collaborating client makes updates to the data using a shared-memory architecture with some control mechanism to ensure data integrity [102].

Because AJAX allows the content of web-pages to become dynamic and responsive, the browser has become a middleware for client-side applications [51]. Typically, the server's response to an AJAX call is to provide HTML content to be displayed in the page [51] [52] which is a sub-optimal use of the technology as the response is only a character string that can convey anything on which the client and server application agree. Thus, the server application can send data or even commands to be executed. The result of this is a client-side capable of performing its own tasks, thus lightening the load on the server infrastructure [52]. Unfortunately, the entire browser environment is event-based which makes concurrent programming (e. g. Threads) impossible; JavaScript being a script, it is incapable of interrupting a function and resume it later. Consequently, handling calls to

the server is done in a separate function, requiring great care when building complex behaviour [53].³

2.7.5 *AJAX Servers*

There are numerous systems called “AJAX Servers” that can be purchased and/or obtained freely. They claim that the software developers only do server-side coding and indirectly suggest that all computation happens on the server, never on the client. The claims continue that this increases security, robustness and so forth.

In fact, these are systems that resemble PHP and JSP whereby the server application generates the HTML and JavaScript. Every user interaction requires four steps:

1. computation on the client-side,
2. the client-side sends an AJAX call to the server,
3. the server application performs more computation on the call received and
4. DHTML code is returned to the client in response.

One important example of a so-called AJAX servers needs to be mentioned here. DWR (Direct Web Remoting) is a major step forward in web-based distributed computing and describes a technology combining AJAX and Java libraries and promoting client-side programming more similar to server-side JAVA [54].

The name is subject to interpretation as remoting could mean “remote login”, “remote desktop” or “remote control”, none of which are directly supported by this technology. Their major claim is to provide Object Request Brokering (ORB) between client and server such that any AJAX can make an object-oriented method call to a server-side class without any special syntax and to provide cross-domain filter⁴.

3 See section 5.5 for important corollaries of developing a distributed environment in AJAX.

4 Before Firefox version 3 in 2008, the HTTP security sandbox prohibited calls from a page originating in one domain to be made to another domain.

DWR is the first vendor technology to openly use various streaming models (e. g. HTTP Streaming, Ajax Push etc.) . It also provides Java-like programming of client-side, thus promoting better software architecture in RIA. It integrates the DOJO user interface library [55] for easier, expectable results. However, RPC is limited to communications with the server – even that remains sub-optimal and the sophistication creates a steep learning curve for developers.

2.7.6 File and Data Formats

Any given RIA can be developed in different technologies (e.g DHTML, SVG, Flash, Silverlight or Java Applets). Different environment that offer the same services must somehow cope with the different data formats produced by these different technologies [41], especially where hand-held devices are included. Collaboration between different companies requires compatibility between the different environments, which is currently not the case with WebOSes [49]. Standards in data formats are not always followed, but the impact can be alleviated by the thesis work of [56], on XML schema transformations.

2.7.7 Web-Based Collaboration and Social Networking

In the early and mid-1990's social Networking was mostly a matter of Discussion Groups that could be found by List Servers and Bulletin Board Servers (BBS). A modern extension of this is now found in Blogs and Personal Pages like Facebook, MySpace and Twitter. HTTP is increasingly popular for applications requiring human collaboration as the port for internet communication is much less strictly guarded by firewalls [42]. Also, the explosion of IRC, picture-sharing and other related social applications favour web-browsers as their GUI, slowly abandoning locally installed applications (e. g. MSN Messenger, ICQ etc.).

2.8 Web-Based User Environments

This section will examine what resources are available to re-create the familiar desktop

environment and productivity software in a web browser as well as how RT collaboration can be supported.

2.8.1 Windowing Toolkits

Though not intended for browser-contained GUI environments, Java's awt and swing [62] [63], Win32 [57], dot.NET [58], Delphi [59], QT [60] and GTK [61] have all set the standard on building WIMP environments which define the desktop metaphor. What they share in common is that they provide reusable visual and logical components that can be aggregated and that provide all-way communication with other components.

The big attraction with Java is that it provides a uniform way to create GUIs regardless of the platform on which it finally runs; hence the name Abstract Windowing Toolkit (awt) which provides (along with its derivatives) everything from frames (the Java equivalent of windows) to buttons to content layout managers and a vast array of user-interaction components. Another interesting feature is that Java's incorporation into web browsers (either using Applets or directly using JavaScript) allows the full array of Java's GUI to be used in web-based applications.

2.8.2 JSR 168: Java Portlet Specification

Home-pages with customizable content have become increasingly popular since the late 1990's. Yahoo, AOL, Google and various ISP's have given their subscribers the ability to define what news contents are displayed in their home page and how they are arranged. This has given rise to the notion of 'portals'

whereby a web-page is only a container for an aggregate of sources, 'portlets' [64].

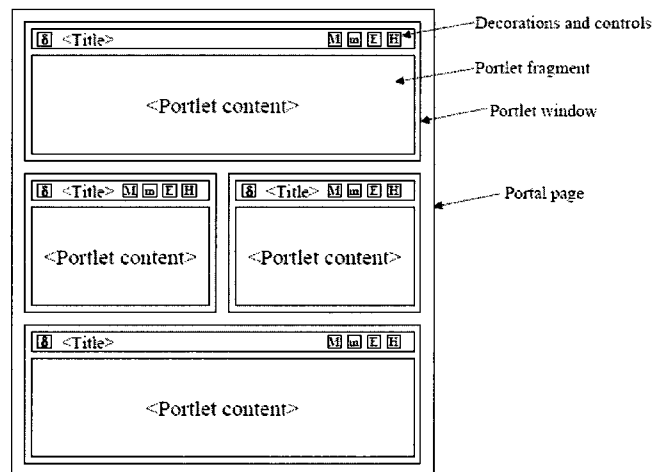


Figure 2.6: Example layout for portlets.

Dynamically creating DHTML to reflect a selection of sources is a difficult task. Java defined the specification for portlets whereby an XML description will arrange the content into simple windows with reduced functionality (Figure 2.6). The specification allows for runtime redefinition of the aggregate (portal) and even interoperability between the portlets. In practice, however, such interoperability is difficult to achieve and the dynamic aspect requires the entire web-page to be reloaded.

2.8.3 *Vendor UI tools*

The following web UI tools provide everything from reusable widgets to full-blown utilities. Their main advantages are that they are already developed and ready to use, usually ensuring cross-browser compatibility. Their main disadvantages are that they are often services hosted on and accessed from their own servers with the data also kept on their own servers while offering no persistence of the state of the user-environment. New applications using these tools are usually difficult to develop and do not integrate between vendors.

YUI (Yahoo User Interface) [65] is an rich set of GUI components ranging from stylized buttons to data grids to tree-views as well as a plethora of core elements such as DOM, Event models and connection management. This is a serious attempt to turn the web-browser into a full-featured GUI designer and process manager.

GWT (Google Web Toolkit) [66] has attained notoriety by allowing programmers to develop in Java then use their compiler to generate the web-browser code. Though it is touted as being a web-programming IDE reminiscent Java, it in fact only support a small sub-set of Java Widgets (buttons, grids and so on). GWT is apparently attempting to emulate Java's full OO coding style and power in browser-executable scripts. Google Wave [67] is an endeavour to extend the real-time collaboration paradigm of Google docs to random applications made with GWT and deployment capabilities on anyone's server. Java Server Faces (JSF) has long been touted as a means to reuse components that have proven their worth [68], but like GWT, it is mostly limited to form widgets.

Programming in JSF is somewhat complicated and requires using Taglibs (imports from servers on the web). The life cycle described in [68] shows very heavy network traffic as each and every user interaction results in a server query.

2.9 Client-Generated: Various AJAX libraries

Whereas many important technologies (e. g. PHP, JSP and Ruby) are often concerned with generating client-side execution code on the server, more and more tools are available to develop RIA and user environments directly where it is used: in the browser itself. These are usually known as AJAX libraries or toolkits because of their built-in support for server communication through AJAX. In fact, they are entirely coded in JavaScript and CSS, relying greatly on the DOM of the web-document.

Because of the tremendous success of the Eclipse IDE for Java, C++ and UML, all AJAX toolkits integrated into an Eclipse-based IDE have become major contenders. ZEND [69] is one such distribution of Eclipse for the development of both servers in PHP and user interfaces via the Dojo AJAX library [55].

Several libraries have emerged as the foundations of bigger toolkits. Prototype.js [129] is doubtless the most important of these providing core-functionality: pre-defined functions and short cuts to facilitate JavaScript programming. Several of its derivatives, notably OpenRICO [23] rely on Prototype.js to provide reusable animations (drag-and-drop, graded resizing over time, etc.). These in turn provide tools for other powerful resources, ExtJS [70] being one such example that brings web-programming into the realm of component-oriented programming. It offers the basic form and windowing widgets and is used by a large number of serious enterprises for their portals and most importantly, it is compatible with GWT. Another noteworthy toolkit is Dynarch [71] which includes containers with managed layouts from menus to slide-shows and full applications ranging from mail clients to chess games. It is currently becoming a core library with support for AES encryption.

The Laszlo platform can be purchased and also exists in open (free) versions. It is a very

popular RIA development kit that offers rich stylistic appeal using both AJAX and Flash. The commercial version showcases a ‘dashboard’ [72] which is essentially a portlet layout with some communication applications including RSS feeds, messaging, address books as well as some personal administration utilities. Every reaction on client side is dependent on a response received from the server application. They also offer server application that can be deployed on most mainstream server technologies. Alternatively, a full server-client version is compiled for JEE. As with JSF, Laszlo user-interaction generates high network traffic.

2.10 Social-Networking Applications

Applications have been built on the technologies mentioned above, many of which have evolved into platforms for social networking: the research underlying this thesis has found too many to be listed here, the most prevalent are detailed in [73]. These include, but are not limited to, collaborative office productivity to flow-chart and drawing to computer and telephony.

2.11 Full Web Desktops (WebOSes)

When work on this thesis started, the very notion of recreating the desktop in a web-browser was experimental and unproven. Now, numerous full-featured desktop environments are available, sometimes supporting social networking. Some of them can be purchased for the development of an enterprise's software requirements.

An evaluation of full browser-contained desktop environments will be based on the following criteria:

1. **Scalability:** applications must be support multiple independent instances.
2. **Compatibility:** must support various applications & integration of technologies.
3. **Reusability:** ease of developing new applications.
4. **Sharing and Collaboration:** the strength of the mechanisms used.

YouOS was one of the first and most functional AJAX desktop workspaces found during the research on existing solutions. The project, now defunct, was a joint venture by several students from MIT, Stanford and CalTech [74].

Bindows [75] is probably the most expansive toolkit, providing a full desktop metaphor in DHTML (Figure 2.7) and a wide array of interactive GUI components including (Figure 2.8 clockwise from the upper-left corner) slider-selectors, a map utility that supports vector path tracing and other layers, Visio-like flow-chart and complex dials for controls, monitors and clocks, all of which can be customized for contextual reuse. It supports component oriented programming along the lines of Javax.Swing and can therefore be used to develop new applications with coding done in a Java-like syntax. These utilities maximize their graphic power and animation ability by using SVG.

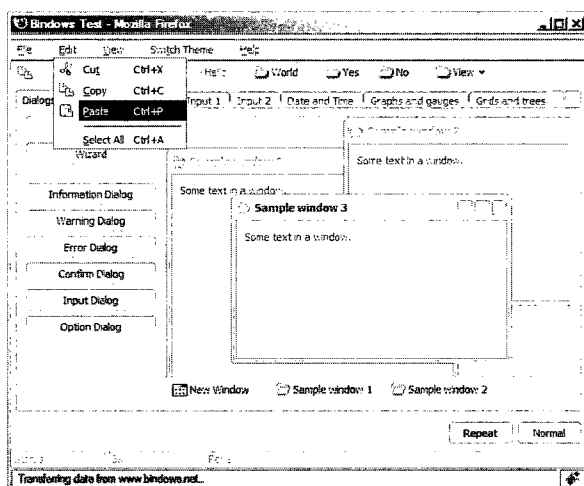


Figure 2.7 : Bindows full WIMP in COP design.

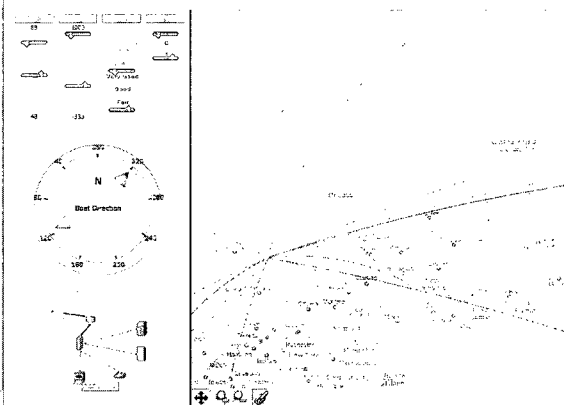


Figure 2.8 : Widgets and Utilities in Bindows.

Cloudo is a DHTML web desktop with support for mobile devices that provides an API for application development in a language resembling Adobe FLEX. Support for mobile is limited to viewing documents (read-only) but porting applications to the mobile platform is apparently done seamlessly from developing for the PC. [76]

DesktopTwo is entirely built in Flash and Java [77]. It provides access to real versions of OpenOffice and Acrobat using VNC (Figure 2.9) in pop-up browser windows. It also claims support for collaboration. Its community includes an application development

team, but its purpose is to provide static applications intended to be hosted exclusively on their own servers.

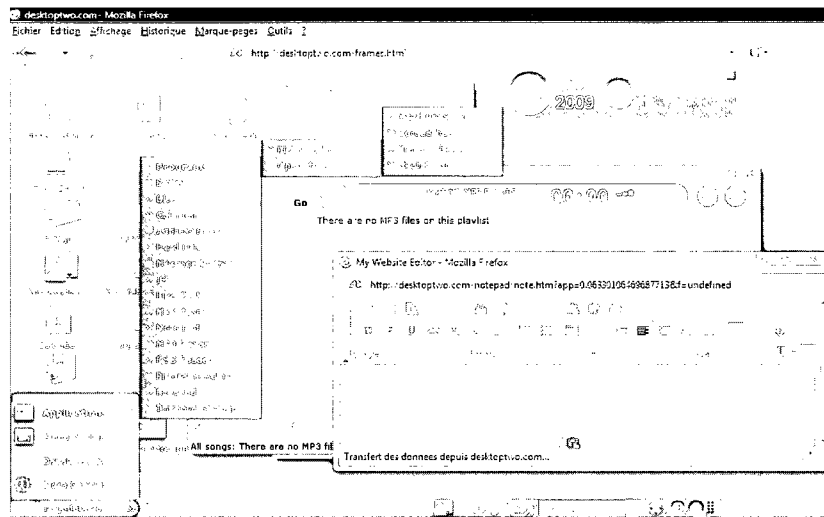


Figure 2.9: Screenshot of DesktopTwo.com

2.12 Web-based Office Productivity Applications

A natural assumption on WebOSes is the support for office productivity software [137] [138]. This section will examine suites that can be used from within a web-browser.

Google Docs [78][139][140][142], is a set of RIA including a word-processor, spreadsheet and presentations. It can import Microsoft Formats but saves everything on their own servers. In other words, to access work done with it, a user must log-on to Google Docs. However, it does allow a user to download (save locally) in generic formats (HTML, RTF) and Open Office formats. Its main advantage is nomadic computing and the ability to co-author in pseudo RT. It achieves the latter by synchronizing every few seconds with the server. The server-side resolves editing conflicts and returns the result to all participants.

Zoho [79][142] is a suite of web-based office productivity software has versions that can be managed either on their own servers or anyone else's ("remote version"). Either way, all applications are designed for collaboration from the start. It must be noted however:

«Since there is no user account created in the Zoho service when using the Remote API's, there are some inherent limitations in the editors, these include: no collaborative editing, document sharing, pagination, document version, and document history..» [79]

It was originally a remote connection to OpenOffice veritabily running on their servers but it is now programmed in AJAX. It handles most office productivity and performs full imports/exports of Microsoft document formats as well as OpenOffice, Tex, PDF and others. It also has plugins for additional functionality in MS Office, Internet Explorer & Firefox, iPhone, Windows Mobile, Facebook and others.

2.13 Remoting and the Virtualization of Applications and the Workspace

End-user applications can be installed on a central host and streamed to users. That is, applications are delivered to users on a per-need basis. Microsoft touts this technology as a means to cost-savings through a reduced number of licenses [80]. This partially revives the notion of “dumb terminals” where client hosts need do little more than provide enough memory and computation power to access remote applications.

Several architectures and applications give people the ability to access and manipulate their own PC from another one [81]. VNC provides both a native interface replica as well as a browser-based replica of a host that runs the server application [82][83]. The client provides full functionality of the replicated desktop environment which is contained in a window.

Sun Microsystems and Mitel have joined forces to offer a different business approach. They also revive the notion of dumb terminal but have enhanced its power by enabling them to connect to a wide variety of OSes running on servers [101]. Mitel has integrated its communications expertise to enable devices of different sizes (Figure 2.10) to connect to servers, giving remote users access to Windows XP, Linux or Solaris. [84][85]

Citrix [86] is an important player in remote computing providing commercial services that cover the whole spectrum of remote access and collaboration. The distinction between Citrix products and a true distributed OS becomes blurred as the products bridge remote control and hardware access. GoToMyPC [87] is a remote PC server and client very similar to VNC. GoToMeeting

[88] is a virtual meeting platform that boasts Macintosh compatibility plus management tools and being accessible from any Microsoft Office product while also being able to record meetings and play them back later. Go2Assist [89] is a remote customer-support platform that includes screen-sharing without software pre-installation. XenDesktop [90] provides roaming access to a user's desktop environment. XenApp [91] claims to isolate any Windows or UNIX application and make it available remotely to the major PC Platforms (Windows, Macintosh, Linux) and also to smart phones.

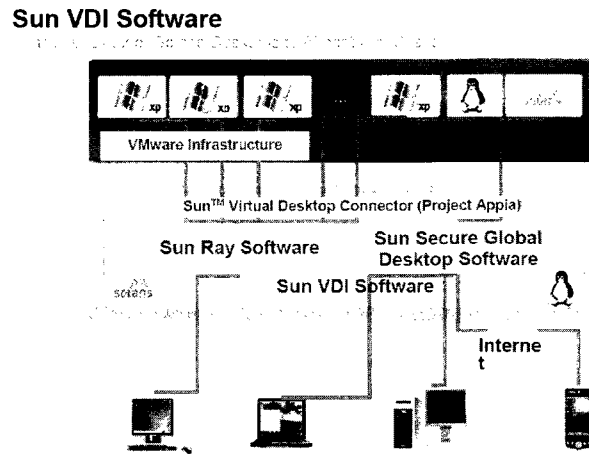


Figure 2.10: Sun VDI allows remote access to any Windows or UNIX based OS from virtually any device.

2.14 Web-Based Programming Support

This section will examine programming considerations required to support all the above in a web-based environment.

2.14.1 MVC: the Key to Scalability and Reusability

The Model View Controller (MVC) pattern decouples data (Model) from the View (or presentation or display), both of which interact through the Controller. This decoupling affords greater reusability. Because the display can change according to conditions in the model, letting the controller define (generate) the display affords better management and RT modification [92]. Most references speak of data models in terms of database

structures rather than component attributes [92][93]. ‘Dual-MVC’ describes the case where either the server or the client produces the data-model. Components are essentially entities that provide a given view in relation to a defined set of control mechanisms on a known model for the data [94].

2.14.2 Object-Oriented JavaScript (OJS)

In OO theory [44][96] a class is defined in terms of a template and an object is defined as an actual instance or ‘running copy’ of the class. Strictly speaking, there are no classes in JavaScript as objects in this language cannot be instantiated: they are essentially singleton variables. Instead, JavaScript provides ‘closures’: instantiable functions that resemble classes as independent entities, respecting their scope and identity for as long as any pointer exists on them⁵.

Inheritance is a key benefit of OO Programming (OOP), though its definition can be the cause of some disagreement [96]. It includes the cancellation and restriction of previous properties as well as optimization of existing functionalities. Simply put, the derivation of a sub-class from its superclass will result in the former inheriting from everything afforded to the latter. The advantages in terms of reusing code and algorithms as well as the modularity afforded as a system grows in complexity should be self-evident.

JavaScript supports two types of inheritance. The first, prototype programming, is designed into the language itself [97], and the other makes use of the declarative nature of JavaScript by implementing the Factory Pattern. The Factory Pattern can be used to implement virtual inheritance (or class nesting) by the iterative augmentation of an instantiated object [98].

[96] mentions selective inheritance (or ‘mixin’) whereby one instance of a class will own a member object (abstract class) to contain required variables and functions. Different

5 The difference between class and closure is outside the scope of this thesis and the two words will be used interchangeably with ‘class’ referring to the OO model and ‘closure’ where it is necessary to recall the JS distinction.

instances of the class can instantiate different subclasses for that object which allows them to respond differently to the same situations. Alternately, the same instance can change the object by instantiating a different subclass in different circumstances, though this misrepresents the very definition of inheritance. Rather, what the paper is referring to is the Command Pattern [2] (or Strategy Pattern [99]) as the variables and functions do not belong to the class proper.

Another important feature of inheritance is the ability to call a function defined by a superclass when it has been overridden [96] (e. g. `super()` in Java). Prototype programming in JavaScript has this built-in.

2.15 Summary of Deficiencies in Previous Approaches

The literature review above is merely a fragment of academic and industrial research and development in the fields of UC, real-time collaboration and web-based user environments that are capable of supporting them. Despite the evident depth and quality, several deficiencies can be noted in the literature thus forming the foundation for the work presented in chapters 4, 5 and 6. All of the work presented by this thesis consists of novel solutions as well as innovative extensions or adaptations of existing design patterns and protocols in order to answer the deficiencies described below and bring real-time collaboration to the cloud in new and expanded manners.

2.15.1 Appearance and Disappearance of Related Projects

When work on the project underlying this thesis began, web-based windowing toolkits and the like were scarce to be found. Those that were available did only supported modification with substantial effort. Since then, dozens of related projects and resources have appeared – most of them after the architecture presented in this thesis was well established – and many of them have gone defunct.

None of these resources, no matter how well (or poorly) designed, supported the metamorphic architectural requirements nor the mechanisms for real-time collaboration

or interoperability which means that they must be implemented for each application by the developer. Considerable re-definition of their components is necessary and is usually made impractical by the coding style of the libraries. This means that the final goals of this thesis cannot be achieved by existing web-desktop libraries and a new one needed to be built from the ground-up.

2.15.2 Sharing and Real-Time Collaboration Issues

Current research in real-time collaboration focuses on ensuring data integrity during real-time collaboration transactions [102]. Though this consideration is crucial, it assumes workspaces with single applications and single domains for collaboration management which leaves the matters of multiple concurrent real-time collaboration sessions and cross-domain collaboration unaddressed. Furthermore, concentrating on the data omits the generalizing of an algorithm that will replicate unknown actions (e. g. sharing controls) from unknown sources (e. g. user action or programmed routines or other software intervention) in unknown applications.

(1) Current Sharing is User Activity Intensive

On a user's perspective, sharing in current social networking trends consists of placing files on a server and accessing them via a user interface (e. g. YouTube and Flickr). Some services provide notification of changes in the shared content, but this is little more than enhanced e-mail or messaging as users must:

1. have the specific application,
2. be notified,
3. find open the corresponding message,
4. download the document,
5. find the downloaded document and
6. choose the appropriate application to open it.⁶

⁶ Some steps can be merged through OS defaults, but there still remain many actions to be explicitly performed by the user.

(2) Real-Time Through Polling is Inefficient

Support for RT performance depends on the timeliness of message passing [103]. Periodic polling a shared resource⁷ is the current standard for web-based collaboration. There are two obvious consequences of this:

1. Where changes in the state of the resource is not periodic, there is much useless network traffic.
2. Where changes in the state of the resource are faster than the polling frequency⁸, critical values can be missed if they are changed before the next poll.

Asynchronous communication [103] describes a system built on event handling resembles hardware interrupts. This would cause network traffic only when desired conditions occur and precisely at the time they occur. Even though AJAX implies asynchronous communication, the HTTP request-response paradigm remains synchronized as no server action will be taken until a user makes a request.

2.15.3 One-to-Many = Incomplete Collaboration

Solutions like remote desktop, desktop sharing and most collaboration environments examined above offer, at best, a one-to-many relationship. That is, a single location for data and/or applications is shared with many users (Figure 2.11 a, top). Google Docs provides a many-to-many relationship insofar as documents from a variety of users can be shared with different sets of users and some sets can overlap. However, Google Docs still has the inconvenience of editing only one document per real-time collaboration session (Figure 2.11 a) bottom). Several documents can be opened concurrently via browser tabs which does not provide concurrent sessions in a single workspace. This thesis will also show that this inhibits interoperability.

7 The shared resource could be a data base, a service or any number of devices as long as the overall process resembles the shared memory model.

8 Network lag will be assumed acceptable to the real-time parameters.

A solution where a user's workspace can contain and control different sets of applications from several different users (Figure 2.11 b) has yet to be found. The promises of Google Wave [67] are apparently far from being delivered. Even if they were, cross-domain real-time collaboration would still be unavailable.

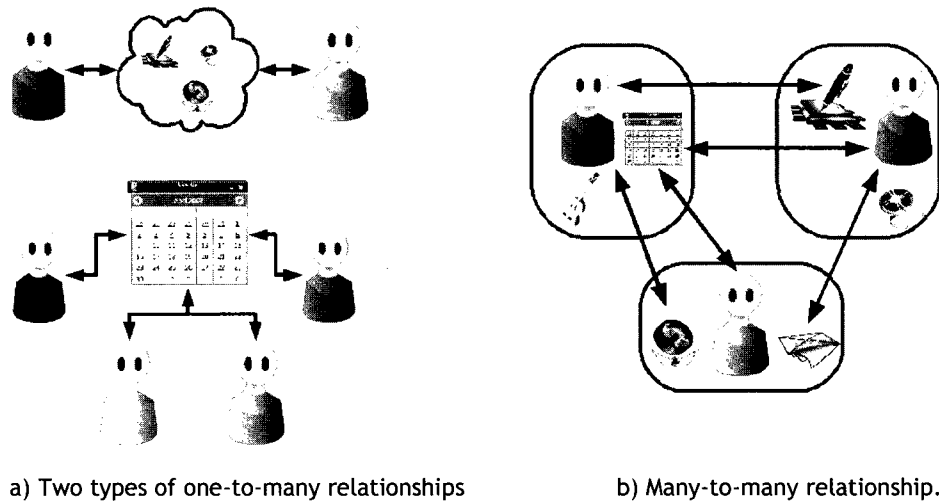


Figure 2.11: Collaboration Relationships

2.15.4 RIA: a Backwards User Interface Paradigm

Less than twenty years ago, PC applications running on DOS 6.2.2 (and often also on Windows 3.1) would require the entire visual real-estate (the whole computer screen). The impact of introducing the desktop metaphor needs no discussion here except to say that it has become an expectation in the mass population, having found similar visual metaphors for every device from entertainment and gaming centres to small cellphones.

Despite this, mainstream RIA is rarely more than form-filling (e. g. financial management and on-line shopping) and message posting (e. g. blogs). At all levels of complexity, RIAs take the entire visual real-estate making navigation from one application to another, or even one document to another, difficult (e. g. [78]). Compatibility between similar applications remains an issue and unifying data remains elusive. Better results in better real-time collaboration and UC are obtained by some commercial products that require installation and native execution on limited platforms

(e. g. [105][106]).

2.15.5 Reuse is Absent from the Design

Until recently, all RIA development was designed and developed on a per-case basis. Reuse and combination of libraries from different vendors remains rarely possible, especially for web desktops. Because these components are actually scripts that turn an HTML view into a full application, it becomes very difficult at runtime to change its content or its layout, let alone both.

«It's easier and more cost effective to develop and evolve networked applications by basing them on reusable distributed object computing middleware, which is software that resides between applications and the underlying operating systems, network protocol stacks, and hardware. Middleware's role is to functionally bridge the gap between application programs and the lower-level hardware and software infrastructure [...]»
[10].

The entire environment being a web-document, the easy development of any new user interface using HTML should be expected. In fact, [107] was found with the very first experimentations and proves the feasibility of using DHTML as a client-side middleware in keeping with the quotation above. Unfortunately, this expectation is not fulfilled by most WebOSes. In fact, a striking drawback of both collaboration environments and most WebOSes is the apparent lack of means to develop new applications, be they built with Flash or DHTML. Where they do, entire communities devote much time and effort to produce only simple applications compared to similar communities for application development on Windows, MacIntosh or Linux.

There is very little room for generalization or modification of existing RIA applications and component libraries [47]. For example, one would want to reuse a rich text editor to build a slide-show designer. But to substitute expected subcomponents with different ones is rarely supported and insertion into other components often fails. Though there are

CASE tools available to produce RIA [83], current developers are not using them. In fact, an inspection of the source-code of existing libraries shows that it is often difficult to abstract them into coherent models. This is unsettling: if RIA is to become more than a populist curiosity and enter as a serious contender for sophisticated use and for collaboration, data complexity and size will increase, and a component model will become increasingly useful [83].

2.15.6 No Iterative or Hierarchical GUI Building

The ability to reuse components is key to scalability in terms of creating new applications without restricting their purposes and construction. To be able to inherit and iteratively augment GUI components (as is done in Java, dot.NET, Delphi, etc.) has a major impact on reuse in DHTML: this is not found in the current state of the art. In other words, building the HTML code must be fully done by the application developer for each application.

2.15.7 Unidirectional Method Invocability

The vast majority of AJAX libraries under-utilize the potential of OOJS. When a callback function is required as a parameter in another function call, library and resource developers will define that function inline. Standard practice makes this even

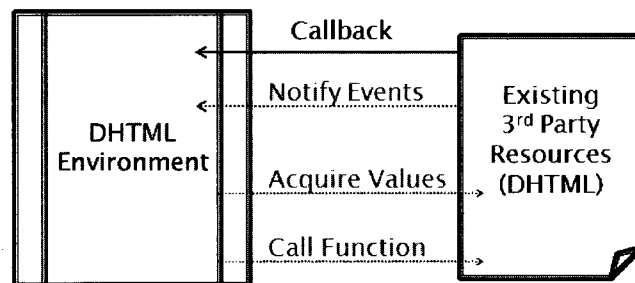


Figure 2.12: "AJAX" libraries call external functions, neither the reverse nor event notification.

more confusing when these callback functions require an object for an argument: this object is also usually defined inline of the inline callback function definition. Each and every time a similar operation is required, the function must be redefined again. Such definitions are static (hard-coded) and fail to make entities reusable or shareable.

A problematic corollary of callback coding can be seen in [65] and similar toolkits is that

their objects can be said to have “Unidirectional Method Invocability” (Figure 2.12). That is, they are capable of calling functions given to them as callbacks, but they neither provide getters and setters for their environment nor do they fire events that can be captured and handled by the environment.

2.15.8 No Technology Independent API

Several libraries often replicate the definition of a given component. Typically, the different libraries require a vastly different style of programming to achieve the same result. An API is therefore desirable to abstract the library chosen such that the programmer will be assured a standard and consistent style of constructing and coding. Furthermore, completely different technologies can be used to realize all or part of the client side. For example, some existing platforms are entirely HTML, while others are entirely Flash, while others are a mixture of both. Therefore, an architecture that makes abstraction of the technology would ensure better portability, allowing a change of basic technology or combining them by using different technologies for different parts.

2.15.9 Absent Combination of Supported Technologies

Web-documents have long provided the capability of embedding components built in browser-independent technologies; though usually plugins, the most prolific of them (e.g. Flash Player, Java, Real Player and Windows Media Player) have become so common that browser vendors include them with the installer, thus requiring no extra action from the user. This thesis will therefore refer to components built with these technologies as ‘embedded’ (as opposed to software embedded in hardware).

Very few of the existing resources combine the various client-side technologies. The most promising ones are Flash-based desktop environments that allow HTML definition of applications. These use the IFRAME⁹ method of defining applications which are subject to attacks from both hackers and malfeasant developers [108][109]. IFRAME application

9 Floating frame element in HTML.

containment also makes interoperability with the containing web-document much more difficult.

JavaScript, JRE libraries, Java Applets and Flash/Flex technology provide APIs to enable communication between them and perform operations on each other through the web-document. The benefits of such communication should be self-evident in terms of a total product built by assembling parts, each developed in the technology that best answers its purpose. This is rarely found, let alone generalized into reusable algorithms and resources.

2.15.10 Not True RT Collaboration, Not on Any Application

Promising true real-time collaboration on any application requires a platform to support it. Single applications can be endowed with special features that give it the capabilities for communicating and so on, but in the current state of the art, the onus is on the programmer. [46] agrees that programming a collaborative application should require little effort from the developer than to produce an application for single-users. The “little effort” means complying with conventions such that an independent entity can provide collaborative functionality where none exist *a priori*.

Though at first glance Google Docs, Mobwrite and the still unreleased Wave [67][110][111] appear promising, recall that:

1. They only work with GWT based applications.
2. They require the server-side to manage the collaboration.

Consequently, their collaboration can be summarized in two words: hampered scalability. It cannot be used on random applications built on random web technologies, it does not support local synchronization of the type OLE would require, using the server for the entire process creates much (often needless) network traffic while increasing the bottleneck on the server and disconnected use is fully unsupported.

2.15.11 Mashups don't Collaborate

The most common mashup services are data-centric views (first and second models from [18], section 2.2), manipulating web-service data [113], and consume the browser's entire viewing real-estate. Existing mashups are static and cannot currently be defined, let alone modified, at run time by the end-user without reloading the environment [20]. Even though they might be initially defined client-side, the vast majority of mashups are organized on the server [112][27].

There are a number of browser-based mashup editors, most notably Google Mashup Editor [30], Yahoo Pipes [87] and Microsoft Popfly [88]. The two latter offer extra ease of use by providing a graphical environment where the designer can link data source nodes and set properties in order to build a pipe-and-filter aggregation. These and several other tools exist which can perform mashups, but these are geared towards developers for the aggregation of server operations [112][27]. Very few (if any) mashup mechanisms are centred around the end-users' needs [27]. None of them aggregate data produced by the end-users and none of them even consider applications as items that can be aggregated into a mashup (which is admissible by the definitions in [18][27][113][114]).

2.15.12 Inter-Dependent Applications are Full-Screen Singletons

From a user's perspective, applications such as Google Maps can be understood as a set of different applications. Figure 2.13 shows that the same search engine is used for Maps, images, groups, news, websites and others not shown. The search results appear textually on the left in a results viewer and also graphically on the map as pins. Clicking any of these pins will show detailed information on the item in a viewer (the bubble).

Applications of this nature are set in terms of the number of sub-applications and their layout. It is not possible to work on more than one copy in the same visual real-estate nor is it possible to rearrange its parts (which would enable the user to organise it for concurrent work on other applications). It is not possible to take one of these applications

and share it or collaborate on it directly with another person.

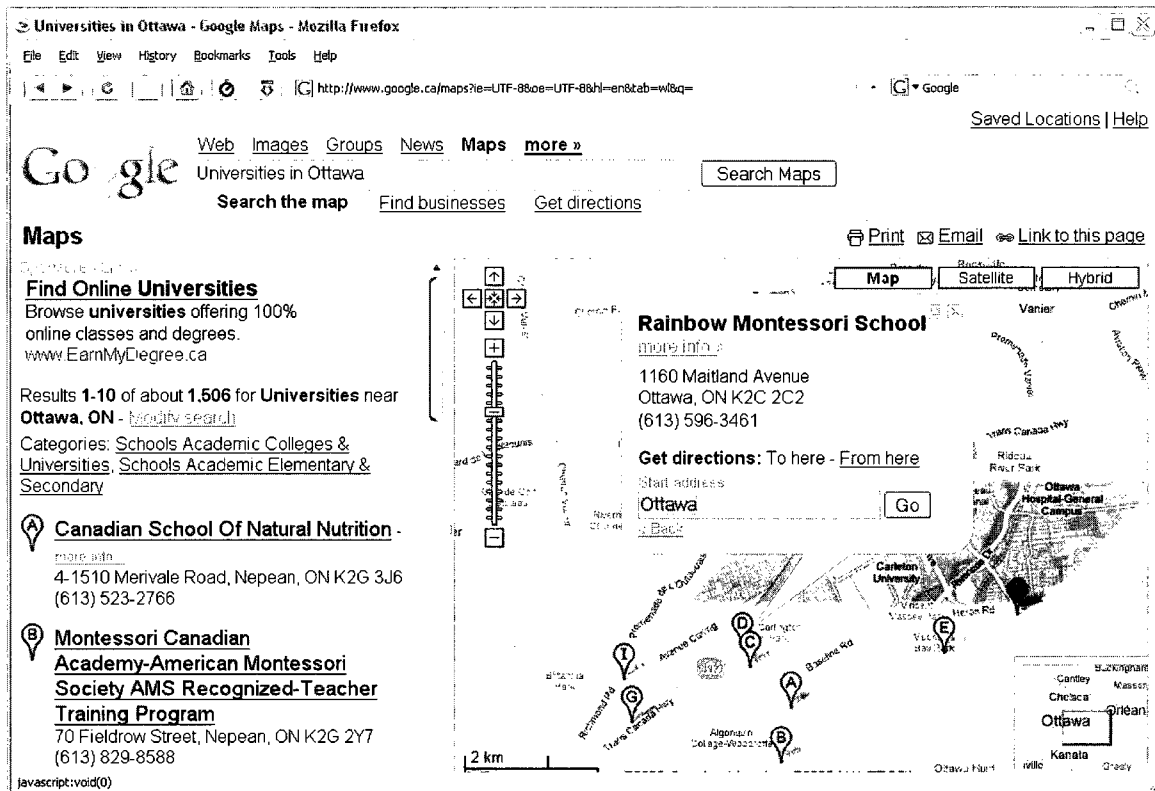


Figure 2.13: Google Maps: three applications in one.

2.15.13 Programming Language and Software Design Issues

The literature review raised some deficiencies with respect to programming languages and design patterns as they are applied to RIA. Early experiments in mobile browsers with existing AJAX libraries that use prototype programming would fail, no matter how simple or small.

(1) Misconception Leads to Performance Degradation

Claim upon claim can easily be found from vendors of various AJAX libraries concerning what JavaScript can do. Though components provided by these libraries often demonstrate impressive animation effects and user interaction possibilities, the difficulty in reading the code is equally frustrating. The reason is often that they attempt to

reproduce features of compiled programming languages in JavaScript. The following examples are not cause for great alarm, but they do suggest a lack of adherence to computer science models in the production of important technologies in the web-application industry.

A first blatant example is the claim that there can be private variables in JavaScript. According to [115], there are ways to define variables visible only to the class: these variables exist only for the duration of the constructor's execution. This is actually called 'scope resolution' and it is very different from a variable that continues to exist even after a given function (e. g. the constructor) has terminated its execution. Only the declaration of a member ensures persistence for as long as the object exists, but in JavaScript they all have public visibility.

Another important misconception, as evidenced by the source-code of [116], is that Threads can be produced in JavaScript by using a timer-based function repetition routine. A looping mechanism can be established between two functions such that they perform together an endless loop until some condition stops the timers. Though the Listener Pattern is based on this idea, endless loops are not an integral part of Threads which are actually concerned with context switching: JavaScript timers do not suspend and resume the execution of a routine neither do they implement the life cycle that define Threads [117]. JavaScript is an event based scripting language: OO as it may become, a web document's runtime environment still only provides a sequential execution of instructions and was never designed for parallel computing in the manner of Multiple Instruction Multiple Data (MIMD) in computer architecture. Though there is currently work on an add-on engine for running Threads within JavaScript [53], concurrent programming is as yet not directly possible.

(2) Multiple Inheritance

The strategy pattern or 'selective inheritance' proposed by [96] (see section 2.14.2) can substitute for Java's ability to implement multiple interfaces and can be attractive academically, but in practice, the work leading to this thesis has found it confusing to use.

A means to change implementations directly in the class – dynamically changing definition – would afford much clearer programming.

(3) RPC and ORB do not Connect Clients

The notion of remote objects on the Internet is almost always a matter of the client calling the server. The notion of servers calling objects on remote servers is left to other design topics and the notion of a client calling an object on a remote client appears nowhere to be found. Notwithstanding the potential benefit to collaboration from a client-to-client protocol, web-based implementations RPC do not support the kind of OOP such as Java RMI provides.

Recall DWR [54] from section 2.7.5. In reality, ORB in DWR is sketchy at best: the client-side has a wrapper that turns a method call into an AJAX call. The server-side interprets the call to instantiate a new object of the required class each time the call is made; this is a fully stateless transaction.

2.15.14 Conceptual Problems with Popular AJAX Libraries

Some AJAX libraries have become cornerstones in RIA development; most are JavaScript-based whereas some are Flash based (called AFLAX) and most of them exhibit the same important properties examined below.

(1) Server Generated Client-Side and Locality of Reference

All but the most advanced web desktop environments use server generated AJAX content. That is, the server application actually builds HTML (or Flash) which the client then inserts where it is required. An inspection of their source-code shows that they are all centred around calling functions on the server-side where client-side execution is perfectly feasible and acceptable (also remarked in [2]). In consequence, and considering the four steps of communication mentioned in section 2.7.5, it is quite clear that each instance of such RIA increases network traffic significantly when it is often not necessary to do so.

Locality of reference is a well known property of computer hardware and OS design [44] and software engineering theory promotes a modular approach where roles and responsibilities are clearly defined and contained [11] keeping inter-dependent routines and objects local while resorting to remote resources only when it is clearly advantageous to do so. This thesis will show that, though it is hardly used in the current state of the art, such a design is feasible for building RIA environments.

(2) *Network Dependence*

By depending on network connections for their very definition (Figure 2.14 [2][51][52]), ‘web-apps’ ignore a major issue: network connections can get lost. A person working on a laptop on a bus, a train, a ferry boat and so on can lose connection. Power settings on a pocket PC can turn off the wireless card when the battery gets low. Domestic ISP services can be unreliable. In such situations, important work in progress should not get interrupted, let alone lost. A mechanism for failure detection and recovery that affords disconnected work would be of real benefit to employees where reliability is essential.

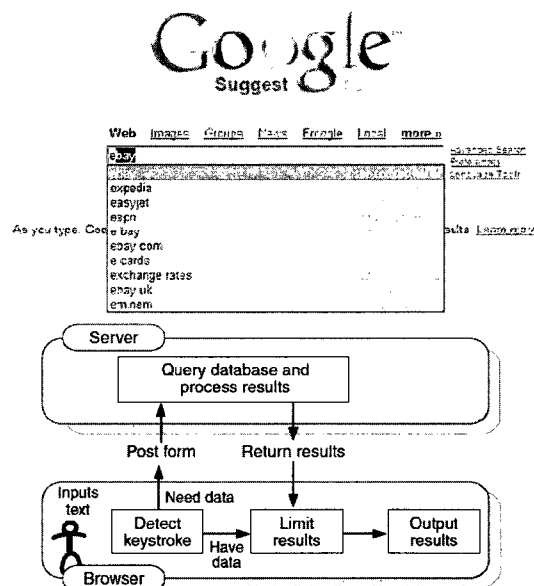


Figure 2.14: High network traffic from tight knit client-server coupling. 178

Even with the best of web environments, working independently from the server is currently unavailable in existing web-based solutions. A user who must disconnect from the network for any reason, is incapable of continuing to use the workspace.

(3) *Inadequate Object Orientation and Improper MVC*

The prevalent means for Inheritance in AJAX-based RIA is achieved with prototype

programming. [97] shows that independence of instances can only be achieved if they inherit from a template that does not change. This is not the case with prototype inheritance as all changes to the prototype result in changes to all instances of the class (see section 4.1.1(1)).

Popular AJAX makes extensive use of the web-document's DOM tree to do almost everything; in other words, basic HTML elements are transformed into RIA by executing a JavaScript script which often modifies the tree and adds predefined functionalities. These functionalities are set and cannot be modified without a lot of effort; such modifications tend to affect every instance of the application, making customisability difficult.

It must be noted that in almost all AJAX resources, animations, user interactions and even the management of server communications are an interplay between HTML elements and CSS properties with the functionality in JavaScript doing little more than a 'piggy-back ride'. Some element of MVC is provided in a heavy library of functions which must be set-up directly in the HTML code after having inserted those elements. In popular AJAX, HTML is both the model and the view, leaving the JavaScript controller fully dependent on it. The MVC design pattern requires the opposite to be true. Consequently, it is difficult to:

1. develop multiple independent instances,
2. produce interoperable applications,
3. externally acquire information on the components.

Where multiple instances of a given class must contact the server at each user interaction, some means of identifying the exact instance must be provided. By grafting functionality onto previously coded HTML, dynamically adding and removing copies of this component very difficult and identifying exact instances from the environment is nearly impossible.

The source-code in the most popular libraries is based on global function calls and static

objects: OO control is thus made very difficult. Using callback programming certainly enables the GUI component to cause a state change in a given controller, but if the controller needs to acquire information from that object asynchronously, it cannot do so.

(4) CSS DnD and Server Communications

DnD for DHTML applications currently focuses on moving visual components with a mouse: a typical illustration can be found in [23] which works by running a script on existing, hard-coded HTML. Acceptance or rejection of a drag-source is done by identifying the drag-source's (HTML element's) CSS class name. In popular AJAX, CSS class names are also used to filter sources to be accepted or rejected when communicating with the server. This is effectively using display styling features used for control – a complete violation of the MVC pattern – and severely limits RIA's capability for scaling and reuse. Visual styles should not be the basis for class identification and different instances of a same RIA component need not necessarily conform to a static set of visual characteristics.

(5) Memory Leaks

When engineering a framework with a long user life cycle (i.e. several applications created and destroyed several times within the platforms use), memory management is an important consideration. Web browsers are even more sensitive to memory problems because of their very poor memory management and garbage collection and that is where the least attention is being paid by programmers. Three kinds of memory leaks are described in detail by [118] which are the effects of functional cycles, policies for the creation and destruction of closures (classes) and the creation and destruction of DOM elements in the HTML document.

There is a growing use of animations in AJAX where timers determine the gradual changes on the animation with recursion. Debugging tools show that this iteratively creates new function calls that consume both memory and CPU activity.

2.15.15 Multi-Touch Support $\neq$ Multiple Mice

Though Microsoft's multi-touch SDK is commercially the closest thing to having multiple independent pointing devices, it still requires applications to register and define their own actions to corresponding events. This means that more than one person cannot simultaneously work on different parts of the same environment. Furthermore, existing applications that were not designed to register for and handle the multi-touch events are unable to benefit from this technology.

2.16 Chapter Summary: Conclusions Drawn from the Related Work

The desire among the general public and need within the industry for real-time collaboration on any application is real and growing. Platforms and protocols have been designed which successfully demonstrate the feasibility of the concept as well as the feasibility of using browsers, more specifically the web-document, as a portable platform for the client side user-environment for real-time collaboration. The web-document supports and can benefit from the co-occurrence of different UI and functional technologies, especially where such technologies provide APIs to communicate with the document (e. g. Flash, Java and JavaScript).

In terms of use-cases that can be derived from the above, the potential of web-based collaboration environments is currently under-estimated:

1. Existing real-time collaboration platforms currently fail to assemble many people with overlapping sets of different applications originating from different locations (i.e. users from different domains and applications from different vendors).
2. Despite the existence of web-based full desktop environments, mainstream real-time collaboration environments continue to cling to full-screen layouts for one-at-a-time individual applications thus making concurrent or intermittent use of several applications as well as the navigation between them very difficult.

3. Web-based solutions hardly integrate the different supported technologies, show no sign of supporting the combination of components from different vendors.
4. Through naïve programming practices the development of new applications is made very difficult. Though a large extent of the range of functional expression from compiled languages can be approximated in AJAX, the needless re-invention of design patterns [9] and programming paradigms has severely limited its potential.
5. Despite the evidence of sub-optimal performance with server-centred generation and despite the potential of client-side autonomy, existing solutions still shy away from a modular design based on locality of reference in favour of so-called “Server Generated AJAX”.

Regardless of the platform (web-based, native OS or virtual-machine), solid engineering practices in terms of modelling, using time-proven design patterns and correct programming paradigms are required to produce a scalable, maintainable and robust solution; the review of both literature and state-of-the-art has shown that a lack thereof is causing stasis, the limitation of supported use-cases and sub-optimal performance.

This thesis therefore concludes that, to fulfil the objectives discussed in chapter 1, it should re-design the entire architecture for the user-environment from the ground-up, including the programming paradigm used to build it and provide reusable mechanisms and algorithms integrated into basic components used as building blocks for the remainder.

Chapter 3

Collaborative WebOS Architecture

This chapter introduces the main aspects that drove the research and prototyping which culminated in this thesis. An overview of the solution proposed will be analysed in terms of requirements, overall architecture, use-cases and the client-side architecture.

3.1 Overview

The research was done in the context of project UC-IC, a web-based server-client architecture for nomadic computing, ubiquity, RT sharing, RT notification and RT collaboration which also provides a platform for application development and runtime re-definition to foster unified communications (Figure 3.1).

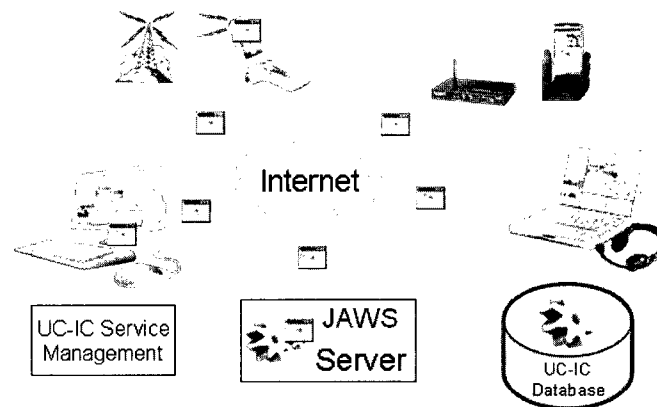


Figure 3.1: UC-IC for nomadic and ubiquitous computing.

Specifically, the research deals with the client side of UC-IC. This client-side is made up of two major packages: the NCCT Internet Windowing Toolkit (NIWT) and the Client-Side Operational Structure (CSOS). Frequent reference will be made to UC-IC's server application JAWS (Java/JavaScript Asynchronous Web-enabled System) as it was a concurrent project which implemented a compatible API. Although by design it was not necessary to do so, in practice the development of NIWT/CSOS and JAWS proceeded

hand-in-hand.

In the context of this thesis, nomadic computing includes the persistence of user activity and the restoration of the state of the user environment at every login just as the user left it when logging out. This means that all applications that were running will be restored with the data, visual aspects and preferences.

The user environment consists of a multi-desktop metaphor. The more sophisticated use-cases are made more intuitive, where possible, by the use of DnD. Collaboration offers a M3 relationship (see section 1.1). UC is supported by providing a UI (the multi-desktop) which contains, manages and displays applications that perform different business communication services such as video conferencing and RT control of multi-media documents; it is further enhanced by affording the user with the ability to combine parts of or whole applications which perform the different business communication into new, integrated and interoperable applications.

3.1.1 Requirements

The client-side must consist of a web-based user environment which will exhibit the following characteristics:

1. It must be built on web-document technologies that require no plugins other than the most commonplace (i.e. Java, Flash Player) where communications and control commands can arrive asynchronously from the server via event-based protocol that approximates hardware interrupts.
2. It must provide a multi-desktop metaphor which allows the end-user to maximize the use of DnD for object association, process launching and application re-definition.
3. It must also act as a framework for application development where support for real-time collaboration, interoperability, DnD (object association and reactivity) are built-in, requiring little or no extra effort from the application developer.
4. The applications framework must provide independence of domain such that a M3 relationship is implicitly supported.

5. The framework must provide a reusable component oriented windowing toolkit similar to Java's awt and swing with a clearly defined MVC construction which promotes the ability to substitute equivalent components (e. g. replace a tool bar with a menu) by implementing full introspection features (see section 4.1.1).
6. The user environment must support user redefinition of applications during real-time collaboration with metamorphic changes of structure (sub-components), interface and implementation changes echoed to all participants; this must include 'mashable applications' built by the run-time addition or removal by the end-user of pertinent components of required complexity.
7. Design and implement a dynamic data structure that reflects, responds to and re-creates the metamorphic attributes of any given component by delegating the acquisition of information, and the restoration from data, to the levels of inheritance and sub-components concerned.
8. The user-environment will assume sporadic communication failure with the JAWS server and enable the user to continue working on all available applications during such failures¹⁰. A recovery from failure protocol will be designed and implemented accordingly.

10 Note that sporadic failure to reach any given server does not mean full network failure. Some services may continue while others fail and vice-versa.

3.1.2 Client-Server Communication Protocols

Section 2.7.3 explained the difference between the standard HTTP Request-Response protocol, HTTP Streaming and AJAX Push. Early experiments found that HTTP Streaming was capable of supporting asynchronous messages to the client from the server by maintaining objects related to the connection. However, it lacked in robustness: disconnection states cause the browser to react to time-out events and navigate away from the web-document serving as the user environment. AJAX Push is essentially a refinement of HTTP Streaming and was found by the work on the JAWS server to support a stateful system while facilitating the detection of connection failure on both the server and client sides.

For the reasons above, AJAX Push serves as the basis for all communication protocols between the server and the client sides. Above parsing, interpreting, sanitizing and reacting to formatted messages from the server, early experiments found that AJAX is also capable of executing commands ‘written’ to it by the server; this is the basis of server-based administrative control over the user-environment.

3.1.3 Overall Architecture

The general architecture for both the client and server sides consists of the modules shown in Figure 3.2. Exclusively server modules pertain to data persistence (including application management) as well as control of user activity¹¹. Exclusively client-side modules pertain to generating the GUI display: visual containers, appearance and customization, visual interactive widgets, specific application behaviour and functionality (where autonomy from server action is possible) as well as the building blocks for an Integrated Development Environment (IDE) which includes customizing the user environment.

¹¹ Server architecture belongs to the scope of another thesis. It remains important to briefly mention it here as the underlying project remains a continual interplay of the server and client sides.

A number of modules can be considered shared by both the server and the client as their purpose and activities entail management on both ends. The server manages the data base in which applications persist their data and obtain the data which allows them to restore their structure, content and preferences. Protocols and services are by nature an agreement between the server and the client, using components which give access to the communications platform between them (AJAX with specific implementation, messages and signals).

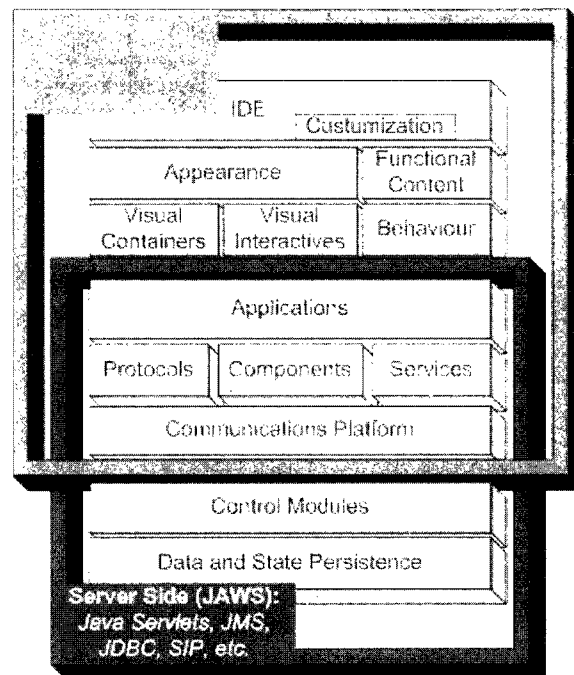


Figure 3.2: Server and client side modules of UC-IC.

The JAWS server application is responsible for persisting and restoring users' activities and application state. It is also the relay and control node for RT communications and collaboration and thus is responsible for the implementation of SIP, broadcasting and filtering legitimate activity such that the system will block and respond to illegitimate activity. The NIWT and CSOS client provides the user interface including layers and controller counterparts for RT communications and communications as well as the API and supporting routines to fire events on the server-side while handling events fired by the server application.

3.1.4 Thick Client for a Rich User Environment

The client side architecture was designed to benefit from the integration of technologies supported by web-documents and is built using AJAX and Java Applets. New applications are built by augmenting NIWT components with application-specific content implemented in either AJAX (DHTML), Flash, Flex, Java Applets or a combination thereof and may include compatible media players and PDF files.

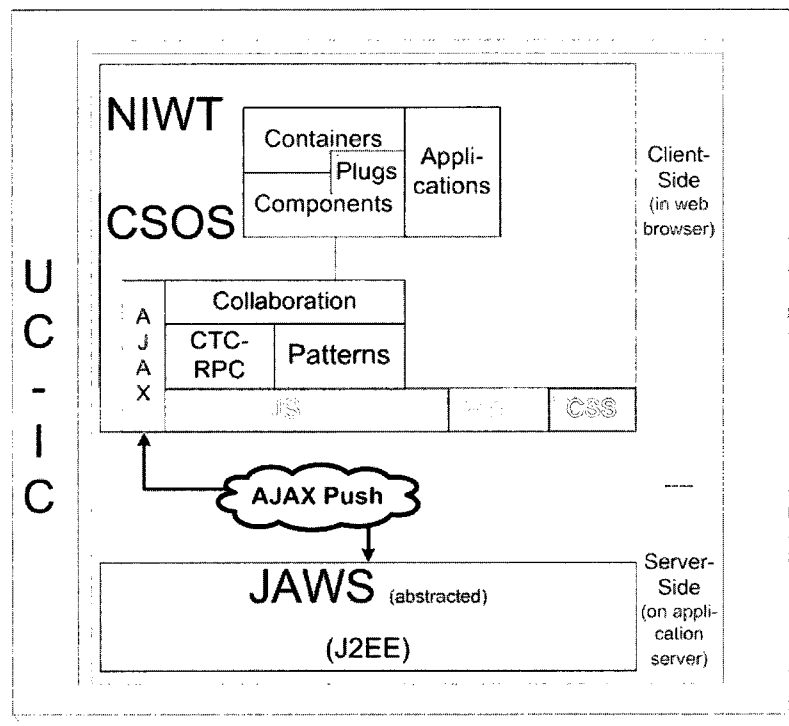


Figure 3.3: Building-blocks of the UC-IC client and server.

Figure 3.3 shows that NIWT provides the basic building blocks for the of the user environment as well as applications whereas CSOS provides the management and communication tools, including state persistence, sharing and collaboration. NIWT applications consist of visual container objects (GUI components) that organize existing applications (reused) and visual interactive components (e. g. sliders, calendars, text editors and colour choosers) as well as functional components that manage user interaction (e. g. DnD). All GUI components are Object Oriented JavaScript (OOJS),

generate HTML, fully introspective and capable of adding and removing functionality and interface definition at runtime through the use of components called ‘Plugs’ (see section 4.1.3).

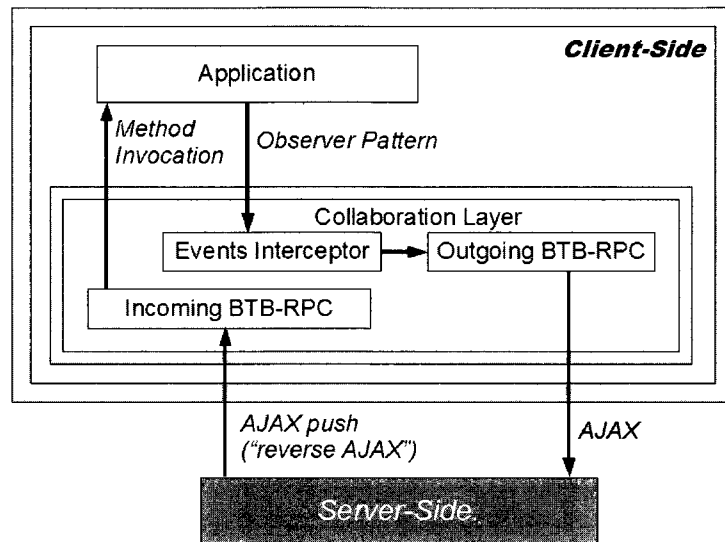


Figure 3.4: Application for collaboration.

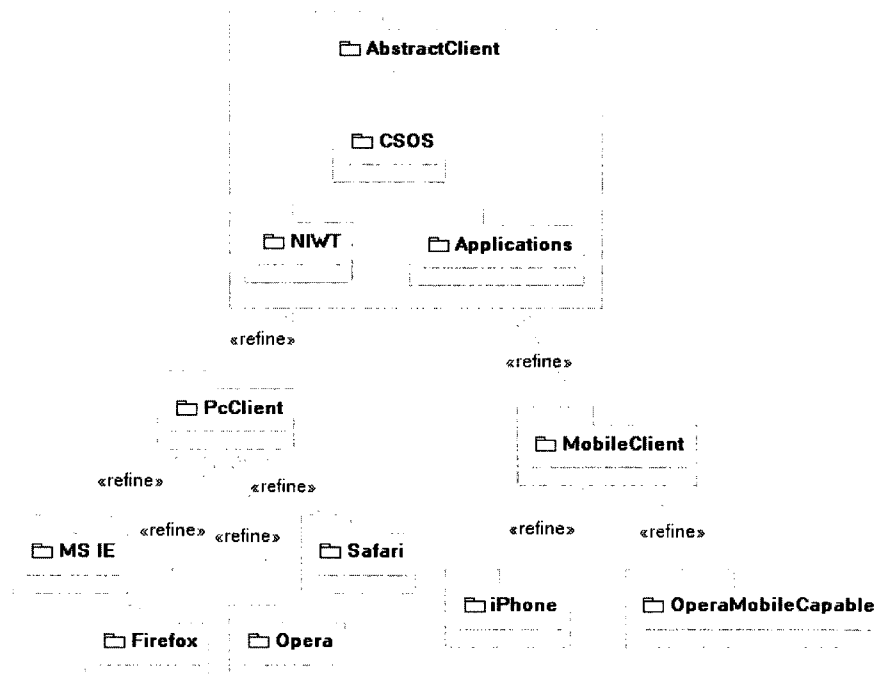


Figure 3.5: Ideal package organization for the client side.

The general architecture for collaboration in UC-IC is given in Figure 3.4 which approximates the model from [102] (Figure 2.3). A collaboration layer on the client-side observes events fired by applications in order to replicate them on the corresponding application running in the environments of all users participating in the collaboration session using browser-to-browser RPC which uses the server as a broadcast relay.

The JAWS server is capable of identifying the client hardware as well as the browser that is making the call. Therefore, the package organisation separates abstract functionality from specific implementation for the various clients connecting to the server. Figure 3.5 shows an organization of the client side that imitates Java's package inheritance. This organisation allows the server application to parse its file system, iteratively choosing the sub-tree corresponding to the specific client.

(1) Signals Between Major Modules

Figure 3.6 shows the major components involved and interfaces for the signal lists they must handle. The server-side has implemented generalization whereby the same AJAX to a small number of 'ports' (URLs) are reused with changed parameters for specific purposes. Four types of signals are sent from the client to the server:

1. **User Activity:** login/logout of multiple users and recovery from failure.
2. **Component Activity:** changes in the data or state of any component (typically applications).
3. **RPC and ORB:** these are browser-to-browser messages that wrap function calls to components on the environments of other users.
4. **Messaging:** special communication messages that are not managed by RPC or ORB can be wrapped in a unicast or multicast message that will be delivered by the server application to other clients.

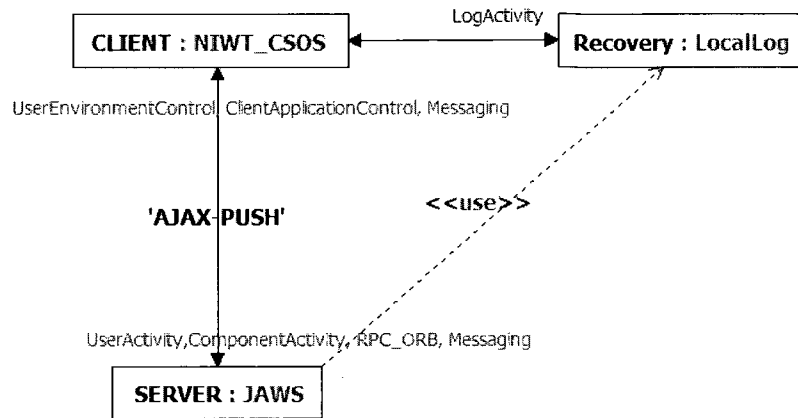


Figure 3.6: Communications Channels Between the Major Components.

Communication signals from the server application to the client are filtered by a small list of commands that can be executed by the AJAX client itself. These are:

1. **User Environment Control:** adding/removing/modifying elements of the multi-desktop environment and executing random functions.
2. **Client Application Control:** calling functions on specific important components. This includes updates to information relevant to other user.
3. **Messaging:** includes browser-to-browser RPC and browser-to-browser ORB but is intended to send random messages to be handled by an application identified in the message.

Finally, the recovery manager logs user activity in a file on the local host when network failure is encountered.

3.1.5 Recovery from failure

Almost all web-based technologies including JSF, Ruby and almost all self-styled “AJAX platforms” are built the assumption that the network connection is dependable. Though it is true that HTTP ensures delivery of packets, the rising popularity of mobile devices brings to light the importance of assuming the contrary: the Internet connection will fail and it may do so at critical moments. For example, energy-managers to favour battery life might disconnect the WI-FI, signals can get lost between base stations, etc.

The browser's reactions to disconnection and time-outs are unpredictable insofar as they differ from one vendor to another and from one version to another. Furthermore, even if browser configurations can be made by the client to help overcome this hurdle, it requires a certain amount of both expertise and effort: a demand that might be unreasonable to make. It seems trivial that a complex client-side serves the user better if it is capable of continuing its work during network failure, when activity can conceivably be designed as independent of server participation. When the connection is re-established, a synchronization routine can make-up for any discrepancy incurred by such autonomous work.

Clearly, the more the client-side is autonomous from the server, the faster its performance becomes and the less loss of connection is disruptive. Loading required JavaScript (and similar linked resources) has proven sufficient to ensure application definition and management where communicating with the server is not absolutely necessary.

Recovery from failure implies three aspects:

1. Disconnections must be detected in order to enter the failure mode (or "fail state").
2. A mechanism for saving the state of the environment and each application must be provided and be kept up to date of all changes.
3. When connection is re-established, synchronization with the server application must ensure integrity of data and state.

Figure 3.7 shows that any failed AJAX call to JAWS results in the client-side entering the fail state and a log of activity is kept locally to synchronize with the server once connection is restored. Communications to the server are re-directed by the Recovery Manager to a file on the local hard-disc serving as an activity log.

The state "LogActivityLocal" performs two important activities. The first happens as soon as the client-side enters the "JawsFailure" state: the entire user state must be persisted in order for subsequent changes to have any meaning. The second activity consists of logging the changes as they occur. The process of restoring the state from this

log then consists of just applying the last changes to each application.

Many such communications are intended to acquire resources, data, or start processes, which serve little or no purpose in disconnected mode. However, each attempt at contacting JAWS re-evaluates the connection status. When a successful connection happens in disconnected mode, the log is sent to JAWS for synchronization, and operations proceed normally.

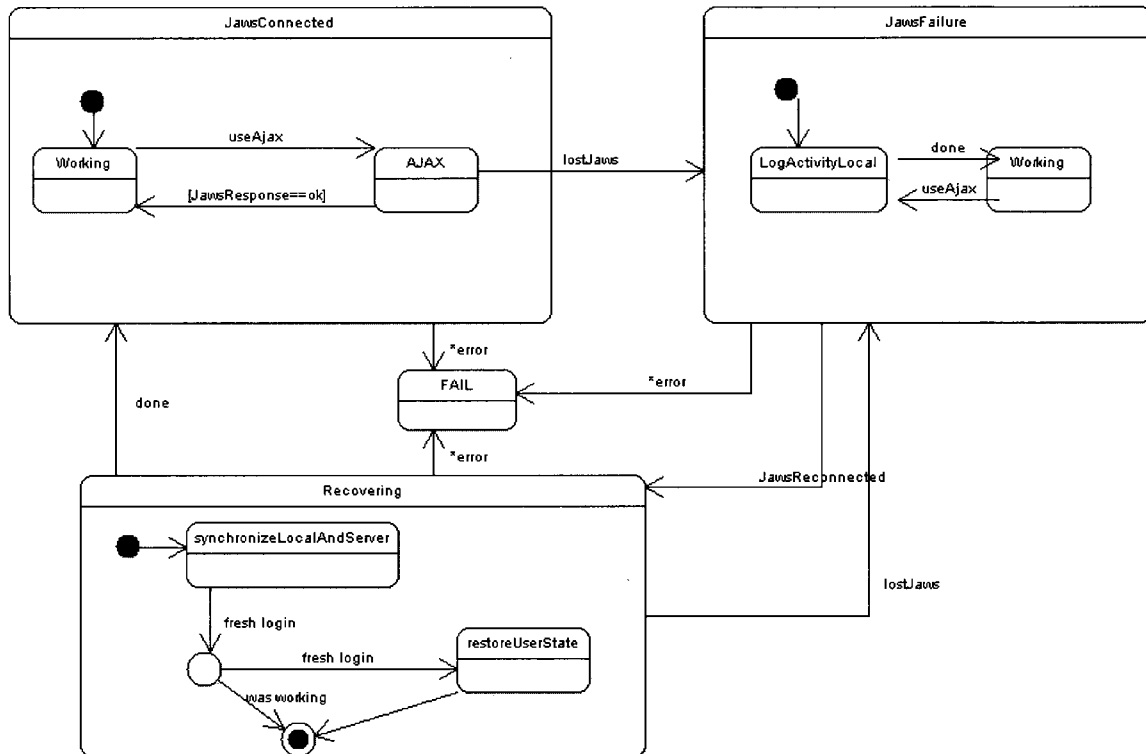


Figure 3.7: FSM for connected/disconnected activity.

In the case where a user logged-out while in the fail-state, synchronization is required at the next login. The client-side will verify if this log file exists or not. If it does, then synchronization will proceed (see section 6.2.7).

Disconnection detection is implemented in AJAX and supported when using the NIWT AjaxObject (introduced in section 3.3.2). The AjaxObject extends XmlHttpRequest object which has a status member that defines the state of the request. Figure 3.7 shows

how the handler for the AJAX call can inspect this status and on any error code¹² will put the client-side in failure mode and begin server-independent activity.

3.1.6 Application Life Cycle for Nomadic Computing

In web-based computing, running applications only have real meaning inasmuch as they are entities in the database managed by the server application. Means to persist and reproduce these entities are required for:

1. **Nomadic computing:** under the assumption that logging in anew from any location will restore running applications and their data to the state of the last logout.
2. **Shared and public resources:** they also require the restoration as described for nomadic computing.
3. **Real-time collaboration:** under the assumption of Application Spawning¹³, users will acquire the application and its data restored to the state of all other participants.

All three use-cases above are summarized by the server-side calling the client-side application creation process. It must be noted here that the difference between the server-side restoring an application or a user launching it for the first time is limited to:

1. The server application provides the required data for full restoration of data, preferences and all other features.
2. The user calls one of many possible routines that will initiate the application to pre-determined settings which may be different in distinct routines.

The difference in these settings are reflected in the data and so, in the restoration process, the server application need not differentiate which routine was initially called by the user. Figure 3.8 shows the steps required to this effect.

12 e. g. 404:page not found, 403:forbidden, 401:unauthorized, etc.

13 Application Spawning: an existing application is replicated (spawned) somewhere else (e. g. a remote user workspace) with its data, preferences and other states faithfully mirroring the original.

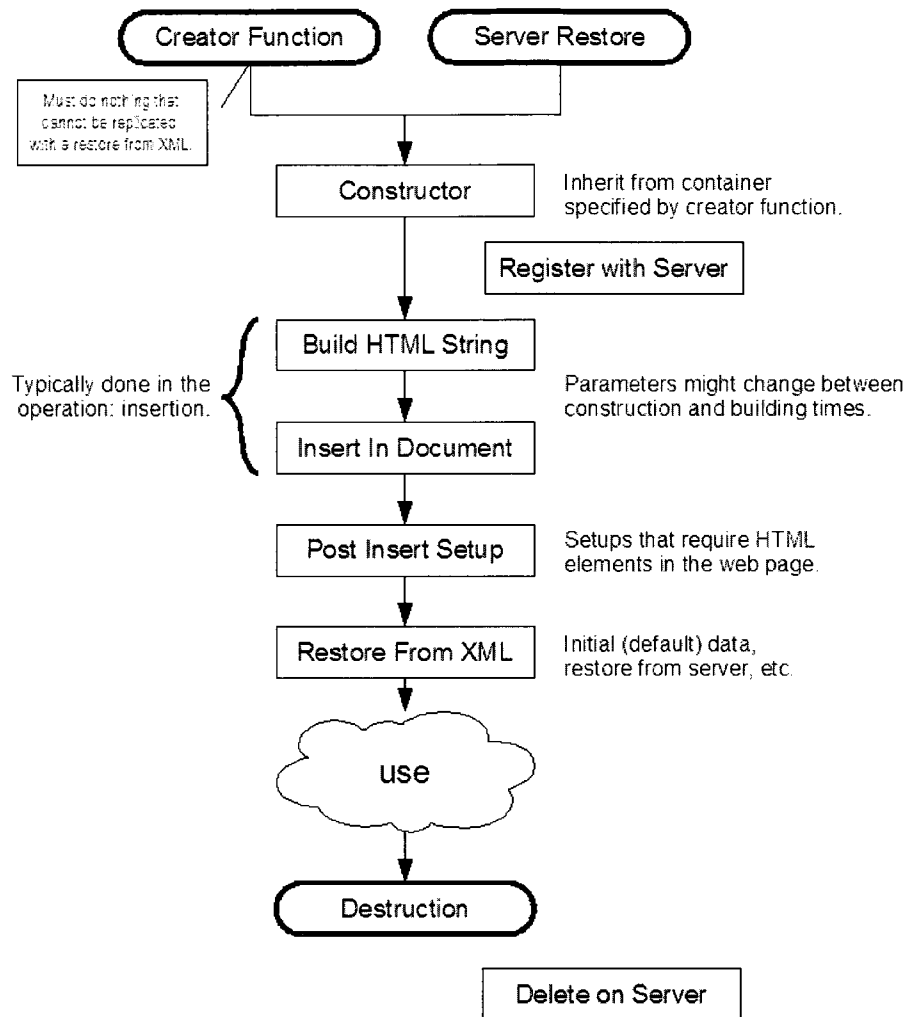


Figure 3.8: Life cycle of a NIWT application.

The constructor should not normally be called directly: applications are created asynchronously by users randomly choosing menu commands, clicking launchers, receiving shared files and answering collaboration invitations, to name only a few. Different creator functions can therefore initiate an application to different settings according to their needs in context.

Unlike Java, the JavaScript closure serves as both class definition and constructor, but like constructors in Java or C++, this is where it should stop. It was observed that building and inserting the GUI in the constructor reduced much needed flexibility. The exact make-up of the HTML code will often depend on settings expressed as member

variables which can be modified after the constructor. The separation between application construction (the logical controller) and the generation of its GUI code therefore promotes a dynamic definition. Instantiating the JavaScript object that controls an application (even its GUI) should therefore not automatically insert it in the page.

Some applications require special setups to be performed after the main GUI has been inserted into the document. A considerable number of third party AJAX resources (e. g. OpenLayers.js [119] used by the underlying project for GIS applications, see section 6.3.1) require one or more specific HTML elements to be already inserted in the web-document before their initialization. The Post-Insert Setup is a placeholder function that is automatically called immediately after the HTML code is built and inserted in the web-document. By default, it does nothing, its intended purpose being to be overridden as required by the specifics of the application developed.

During the course of its use, the application can undergo several transformations in its data, operational state, preferences and so on. The cloud labelled “use” in Figure 3.8 therefore abstracts all server communication that may take place to manage and persist these changes.

As with C++, JavaScript requires particular attention to destructors regardless of the specific technology used for a given application. Developers must pay special attention to destroying all member objects, arrays and any kind of pointer, including pointers to `this` object itself. It must be noted that the deletion of variable names and pointers does not guarantee the destruction of objects in JavaScript. This is discussed in section 6.12.1.

Restoring from XML concerns not only restoring the data model, but also restoring everything affected by it, which includes the metamorphic addition and removal of sub-components, inheritance changes, function redefinitions and visual aspects. The constructor for any component creates a number of default sub-components and defines a certain number of interfaces (Plugs), but others can be added and/or removed at runtime and this metamorphism must be restored when a user logs in after having logged out of

the system. Section 5.10.2 will provide more details on data definition, persistence and restoration.

3.2 General Use-Cases Supported

The same system is used for end-user activity, application development, portal management and portal construction. Figure 3.9 shows how the entire architecture is built in order to support all use-cases through the web-document interface.

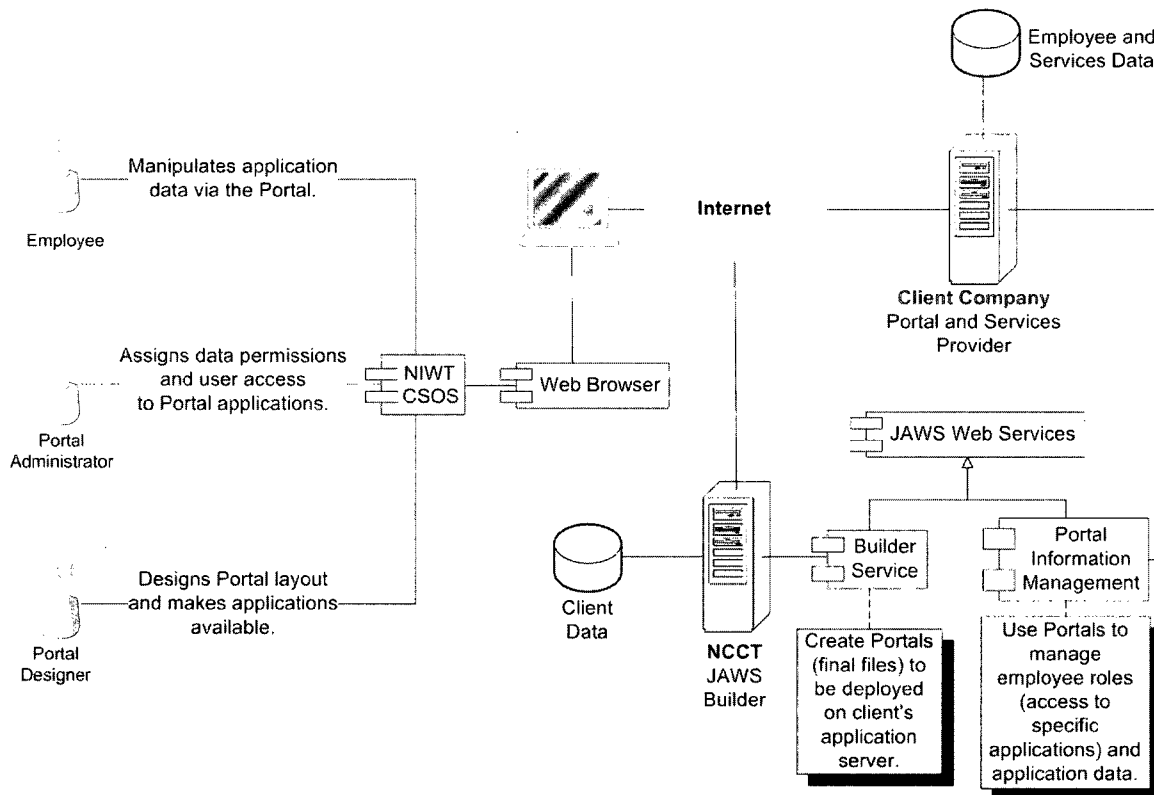


Figure 3.9: Use-case driven deployment.

The same GUI environment must allow not only the use, but also the design and modification of the portal and its contents. The client side therefore provides:

1. The end-user's working environment.

2. An IDE for the development of small components, full applications and even entire portals.
3. An administrative control centre.

Administrative use-cases belong to the scope of another thesis. What deserves mention here is that the client-side restoration process allows entire desktops etc. to be added or removed either at design time or at runtime. The component oriented architecture automatically adjusts the entire environment and the Observer Pattern ensures that all concerned components react accordingly. This is an important step in supporting multiple portal and multiple user login from the same workspace. That is, the same user environment is capable of aggregating desktops from as many portals as the user has access, and the component oriented architecture allows more users to login and append their own desktops to the existing environment. Also, container independence allows extra workspaces to be contained in a window in order to provide a remote-desktop like experience.

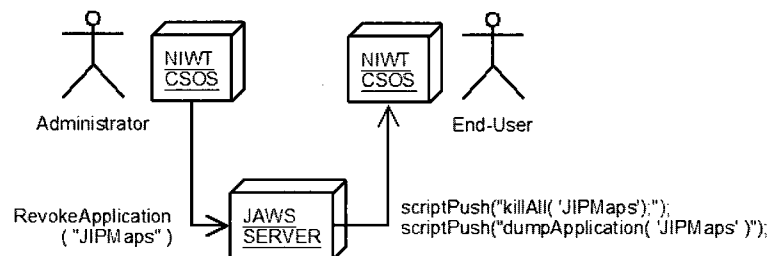


Figure 3.10: Administrator revoking ongoing permission for using an application.

Finally, the remote administrative control of a user's activity, as illustrated in Figure 3.10, has been demonstrated by:

1. forcefully shutting down a (remote) user's workspace,
2. forcefully shutting down specific applications,
3. forcefully starting applications,
4. removing application definitions (denying the user future use of them)
5. 'injecting' new applications into the user's workspace

These are possible because of client-side operational management routines as described in section 5.10.1.

3.2.1 End-User Cases

Recall that NIWT is a multi-desktop WIMP environment. Use-cases for the end-user will therefore mostly revolve around the manipulation of windows and other GUI components. Finally, the component oriented and metamorphic architecture of NIWT allows users to redefine applications at runtime which will impact other classes of use-cases, especially where saving data and collaboration are concerned.

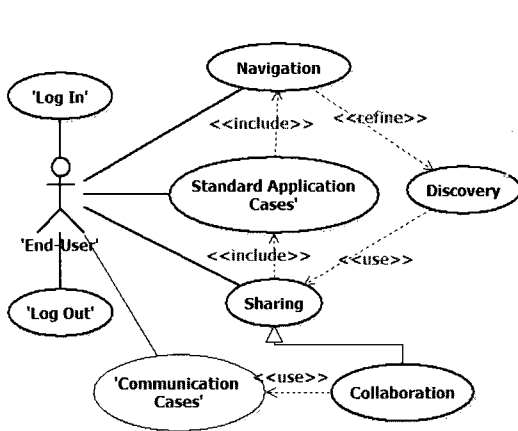


Figure 3.11 : Classes of End-User Use-Cases.

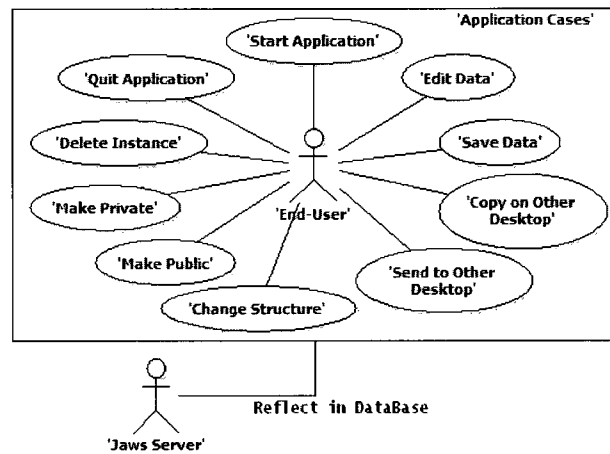


Figure 3.12 : Standard Application Use-Cases.

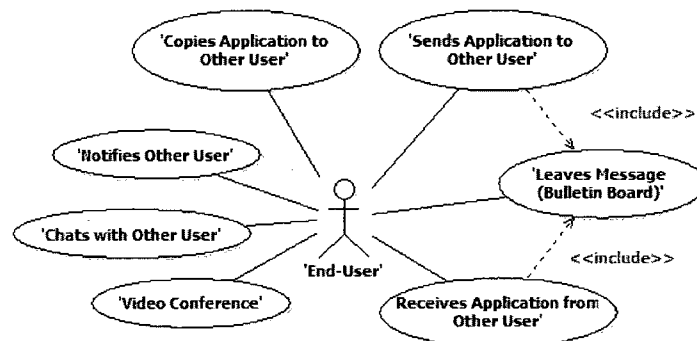


Figure 3.13: Communication Use-Cases

Figure 3.11 shows general cases of user activity and Figure 3.12 shows activities specific to the manipulation of applications. The 'Communication Cases' from Figure 3.11 are

detailed in Figure 3.13. Note that ‘Navigation’ in Figure 3.11 is a general term that describes a user's ability to visually discover resources. These might be applications in separate windows on a desktop or on different desktops; they might be favourite web-pages or shared resources from other users; they might be different portals with any combination of the above. Discovery is therefore an important aspect of navigation, but it also includes how a user can find components or running applications in order to aggregate them into a mashup.

The standard application use-cases also include the JAWS server as a participant because a very important requirement in nomadic computing is that the workspace be restored exactly as it was left; JAWS must therefore shadow in its database many of the actions performed by the user. Such actions include, but are not limited to:

1. Editing data.
2. Changing parent components (e. g. sending a window to another desktop or mashing-in an application).
3. Changing visual attributes and user preferences.

The remainder of the use-cases will be described in the results pertaining to applications implemented on this platform.

3.3 Specific Architecture of the Client Side

This section will summarize the most important components that build the client-side architecture. Recall that the client-side is built of two large-scale packages NIWT and CSOS. Because there is no package resolution in JavaScript¹⁴, the diagrams below will group them to best describe their functional relationship.

NIWT is the windowing toolkit that produces the multi-desktop user environment. Its

14 Package structure is used in the modelling for convenience and clarity: JS does not support them instead resolving all functions and objects to the global scale. The notion of packages clarifies modelling and relates to folder structure in the server's directories.

components include:

1. User environment components: the required containers and routines for GUI organization, multiple user login and awareness activity.
2. Application components: the required containers and routines to build applications and use them in RT collaboration, mashups, LCS, DnD and all related activities which also include the required routines for browser-to-browser ORB.

CSOS is the operational structure which manages the client-side and coordinates with the JAWS server. Its components include:

1. Browser and HTML management components.
2. Communication Components: the required routines for requesting user activity, updating application activity, browser-to-browser RPC and collaboration.
3. Recovery Manager: a single component that keeps the user state for network disconnected activity and communicates with the server for synchronization on network re-connection.

3.3.1 Components of the User Environment

NIWT allows a user environment to be built dynamically, inserting and removing desktops, applications, user profiles and avatars etc. at run time. Figure 3.14 shows the important classes that participate in this construction¹⁵.

The workspace is the single starting point which organizes all the users logged into the single client-side environment and all the portals to which each user has access at any given time. Each portal generates one or more desktops, granting the logged-in user access to its Role-Based Access Controlled resources, including applications, as well as avatars to other users registered in that portal.

¹⁵ The stereotype <<SDMD>> refers to classes' partial or total implementation of the Single Data Multiple Display pattern explained in section 5.4.2 and the stereotype <<Plug>> refers to the particular implementation of interfaces and removable multiple inheritance explained in section 4.1.3.

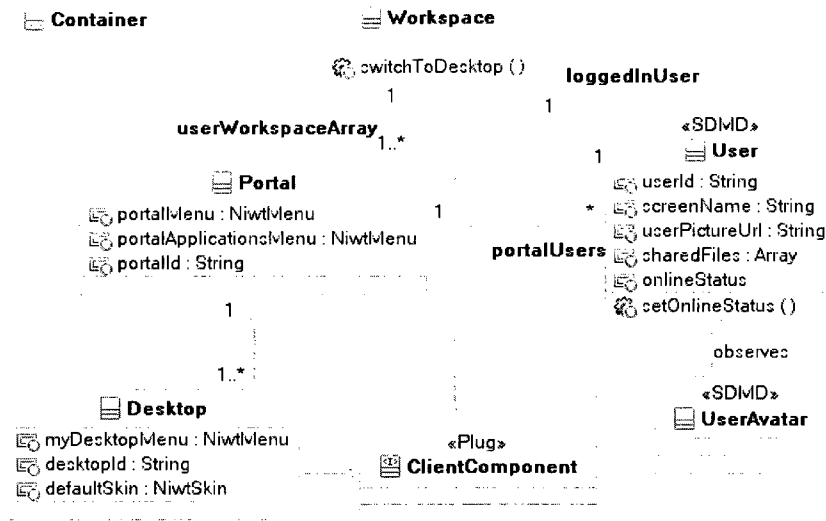


Figure 3.14: Principal components of the user environment.

3.3.2 Components for Environment Management and Communication

The class diagram in Figure 3.15 shows the main components of CSOS and NIWT for environment management.

The AjaxObject implements new functionality, unavailable in the current state of the art: support for OOP within the context of callback coding and deferred synchronicity (explained in sections 2.15.7 and 5.6 respectively) and passing an array of local arguments from the calling function (i.e. that do not come from the server's response) to the callback function.

The CSOS.RPC object handles the marshalling, sending, reconstruction (on the receiver's side) and execution of browser-to-browser RPC calls by automating serialization with the Contract object (sections 5.5 and 4.4 respectively) and the reconstruction and execution of function calls.

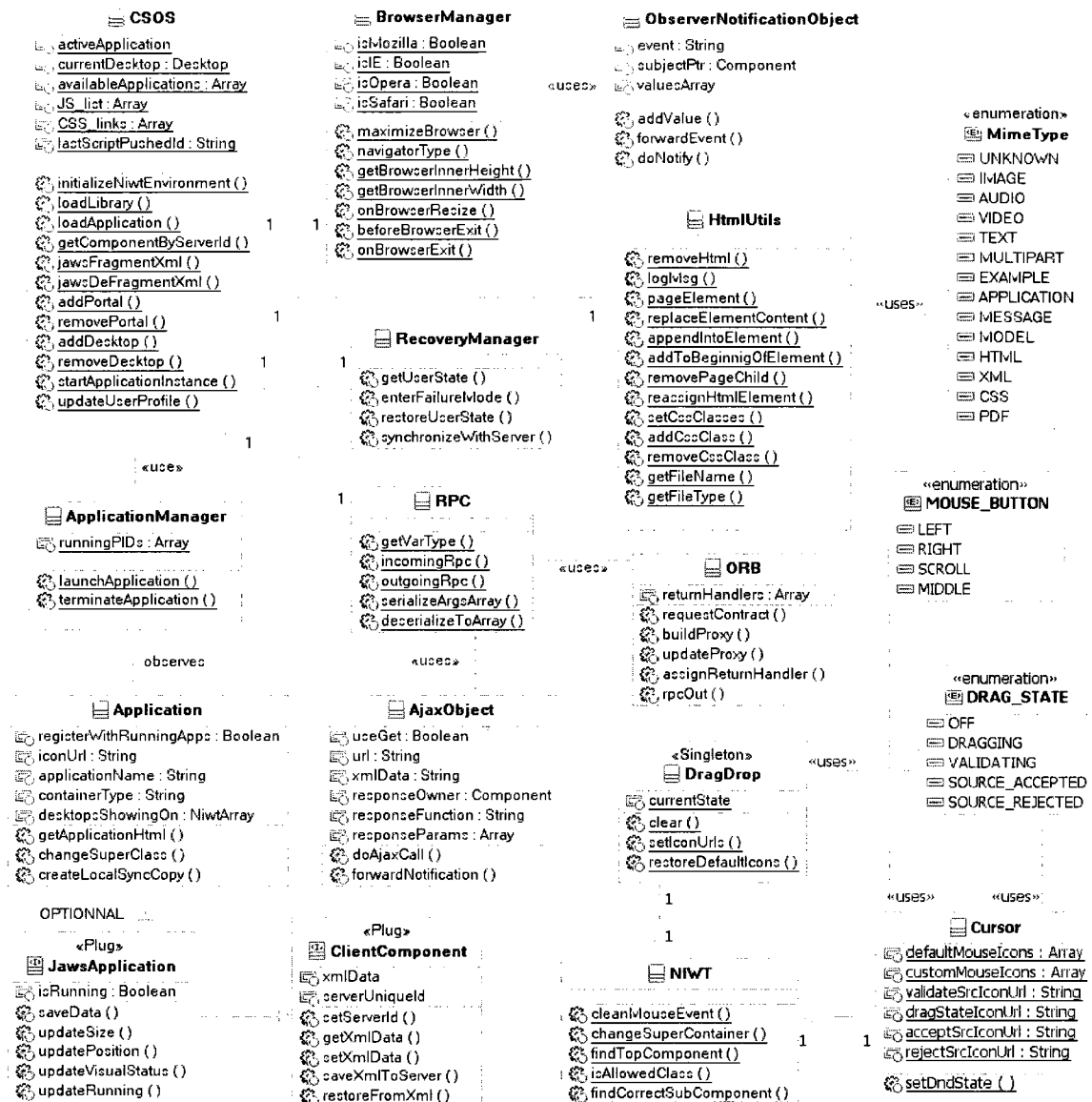


Figure 3.15: Components for environment and use-case management.

The web-browser itself has a number of properties and fires a number of events, any of which might have an impact on the NIWT-CSOS environment and some of the applications running therein. The CSOS.BrowserManager object provides access to these events thus allowing applications to respond in accordance with their needs. Also some applications may need to perform routines when a user logs out. For example, SIP needs to adjust if a user closes the browser (implicitly leaving the session): the

CSOS.BrowserManager allows components to register for notification so they can react before the browser closes.

HtmlUtils is a collection of utility functions for HTML manipulation. It is represented as an object for the convenience of modelling as each function is defined on the global scope for better reusability. They mostly consist of DOM manipulation, but also some functions perform string manipulation and CSS manipulation.

The CSOS.RecoveryManager is a singleton object consisting of a Java Applet which is called when connection to JAWS fails. As long as the AjaxObject is used for AJAX calls, any HTTP error concerning the connection to the JAWS server is treated as a failure. The CSOS.RecoveryManager ensures that data is saved on the local host in order to enable synchronization with JAWS at the next connection (see section 3.1.5).

The remaining classes in Figure 3.15 will be discussed throughout the thesis.

3.3.3 Components of the Windowing Toolkit Proper

Applications are entirely client-contained giving the client side full control over their use while providing network independence. Integration of third-party RIA may require resources only available via dynamic interaction with a web-service, but their containment in NIWT components keeps their activity under the control of the client. Figure 3.16 shows the component classes provided by NIWT, with which new applications are built, including integration of existing RIA. Using these ensures support for the features discussed in chapters 4 and 5.

Many of the classes in Figure 3.16 are self-explanatory, but a few features are worth mentioning here. All classes in NIWT derive from the Component class which gives them immediate support for the observer pattern. GUI components are made incrementally more complex through inheritance by adding attributes and functionality pertinent to their specific use (detailed in Appendix F).

Chapter 4

Adjusting the Paradigm of Web-Programming

The current paradigm for programming RIAs is mostly a static approach consisting of hard-coding HTML and hard-coding a script which calls a library that performs changes on the HTML. Server-side dynamic approaches do exist, all of which are approximated by Java Server Pages or Servlets, both of which offer some control over the HTML content with server-side objects. The combination of object-oriented control and client-side self-determination is a virtually absent paradigm.

This chapter presents a new paradigm where standard HTML programming and classic object-orientation are combined in a client-side equivalent to servlets. A comparison between coding styles is given in Appendix E, String-Code Programming. This improves the MVC while providing all the benefits of object-orientation to web programming, notably reusability through iterative augmentation, all-way component communication and responsiveness to external stimuli. It further facilitates the participation of random RIAs in real-time collaboration by automating key mechanisms which can then be made transparent to programmers.

4.1 Complete Object Orientation in JavaScript

This section recalls important aspects of OO theory and compares two methods of implementing it in JavaScript. The Factory Pattern as implemented in NIWT will be shown to in result in a more legible source-code that also more closely behaves like OOP. Dynamic and volatile inheritance are introduced as new features made possible by the Factory Pattern JavaScript and multiple and removable inheritance will be introduced with the new concept of Plugs.

4.1.1 Improving JavaScript Inheritance

Inheritance serves not only to simplify class definition through reuse, it is a very important tool in the identification of objects that can be accepted or must be rejected for given operations in given contexts.

The base object in JavaScript is endowed with a `.prototype` variable that allows any object to derive from any other [97]. This serves as the basis for OO inheritance in the vast majority of JavaScript resources in the current state of the art.

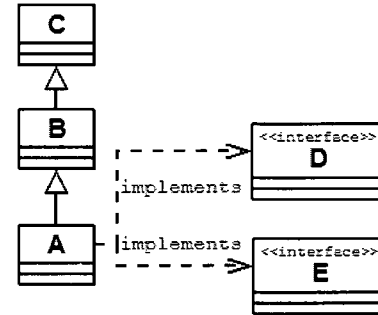


Figure 4.1:
Inheritance and implementation.

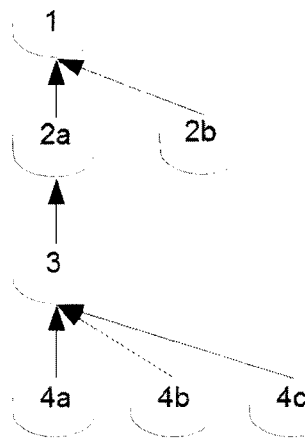
Java set the bar with its operator `instanceof`. JavaScript has a similar operator which works almost as well when using prototype inheritance. Consider Figure 4.1 where `instanceof` will return true when testing for either A, B or C. Prototype programming in JavaScript supports this, but to actually call a function definition from one of the superclasses requires that the exact depth of inheritance be known. For example if `aObj` is an instance of class A, then the definition of `someFunc` from B is obtained with `aObj.prototype.someFunc()` whereas the definition from C is obtained with `aObj.prototype.prototype.someFunc()`.

In Java, interfaces are also included in the class type validation. This means that in Figure 4.1, class A is also an instance of interfaces D and E. In JavaScript, neither prototype programming nor the strategy pattern (or mixin as per [96] in section 2.14.2) support this.

(1) *Factory Pattern, not Prototype*

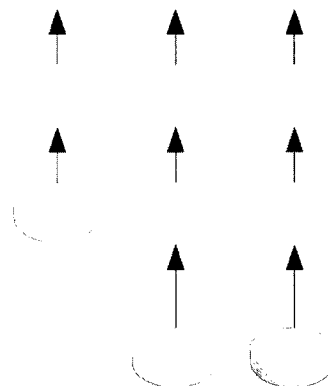
As with Java, prototype inheritance excludes multiple inheritance, but unlike Java, it allows for runtime changing of superclasses. That is, changing an object's prototype changes the template used to find inherited member values and function definitions. It must be emphasized that Prototype Inheritance works on objects, not classes; the consequences of which are discussed below.

Prototype ‘superclasses’ are links to instantiated objects resulting in a lookup table: if a member is not defined in the closure instance, prototypes will be recursively inspected until it is found. This means that any change made to the object serving as prototype will affect all closures at all depth of derivation as well as all instantiations of these classes done before or after the change is made. Consider Figure 4.2, any change made in node 1 will be reflected in all nodes. Changes made to node 2.b will affect none whereas changes to node 2.a will also affect node 3 and all the 4.x nodes.



Arrows: “inherits from”

Figure 4.2 : Prototypal inheritance.



Arrows: “inherits from”

Figure 4.3 : Factory Pattern Inheritance.

Classical inheritance does not work this way: superclasses at various levels provide templates that increase in size incrementally with each new level [98]. Figure 4.3 shows that classical inheritance is concerned with the actual object irrespective of its depth of derivation (the number of levels of inheritance): changes made to a given instantiated object affect that object and nothing else, which is in keeping with Java and C++ inheritance.

JavaScript is capable of mimicking classical inheritance rather well by using the Factory Pattern. At every level of inheritance, a new object of the superclass is created and new features are added to it. The exact coding style is important as it impacts on the behaviour of a script, notably the scope-resolution of objects (see Appendix D, Code Conventions).

In particular, the use of the keyword `this` does not make sense within the Factory (the constructor) but it does maintain its scope in the external definition of its member functions. More will be said on the keyword `this` in section 4.3.1, Scope Decomposition of Sub Components.

Using the Factory Pattern as described above also makes function pointers changeable at instantiation time and at runtime. Overriding functions is just a matter of reassigning the pointer to another definition.

(2) Class Type Identification and Inheritance Lineage

An important aspect afforded by introspection is the ability to identify its class type on all levels of inheritance. Among many other things, this can be used to filter which kinds of objects are admissible to given operations. Recall Figures 4.2 and 4.3, proper class identification should identify that any of the 4.x nodes:

1. As instances of class 4.a.
2. As instances of class 3.
3. As instances of class 2.a.
4. As instances of class 1.

However, class 2.b is neither an instance of class 3 (and so on) nor 2.a, but it is an instance of 2.b and, like 2.a, it is an instance of class 1. This is well supported by prototype programming with no amount of extra effort required on the programmer's part. Unfortunately, the incremental nature of the Factory Pattern does not immediately lend support to this. An alternative is therefore found and explained below.

Java's support of class identification also includes interfaces. Prototype programming does not support this as the very notion of interfaces is absent from the paradigm. This thesis has also extended class identification to Plugs (see section 4.1.3), thus supporting identification of both direct inheritance (class extension) and interface implementation. Because the Factory Pattern does not implicitly support the recursive lookup, the basic definition of the Component object in NIWT provides access to information about the

inheritance line¹⁶, as well as function definitions that have been overridden.

(3) Introspection and Calling Overridden Definitions

Introspection is the pattern of self-reference by which a class can provide information about its inheritance [120]. [44] explains two types of self-reference: delegation and concatenation. OOJS uses delegation when a closure points to a function defined outside the closure. For example, overriding functions is only a matter of assigning a different function pointer to a member variable. This can be done at runtime when deemed advantageous. Concatenation consists of one function calling another function, for example calling the previous definition of a function from within the overridden definition.

Java provides the ability to call definitions from a superclass in the case where one or more generations of subclasses have overridden it. A pointer on the immediate superclass definition can be acquired and used to invoke method definitions from that superclass. Definitions from superclasses at higher levels of inheritance can be obtained by typecasting the keyword `this` to the required class. Prototype programming in JavaScript affords calling superclass definitions at any level by concatenating the keyword `.prototype` as shown on page 72.

Calling superclass definitions is not supported at all with the Factory Pattern. NIWT components were therefore given the ability to save and call overridden superclass definitions by using the string-indexing property of JavaScript. JavaScript objects and closure instances can be used as hash-tables whereby any member can be accessed by treating the object like a string-indexed array: the name of the member becomes the key (or string-index). Strings can be concatenated which subsequently allows string-indices of members to be expressed as the summation of variables. Listing 4.1 shows how this can then be used to find the function definition before it was overridden:

1. When a superclass definition is saved, the definition is assigned to a new member name

16 The class diagram in Figure 3.16 details what the Component class offers in terms of introspection.

consisting of a prefix: the superclass name at the time of saving it.

2.This prefix is returned and (preferably saved as a member variable) can added to the function's member name, thus pointing to the original definition.

```
function DerivedNiwtComponent () {  
    var newComp = new NiwtClass ();  
    newComp.addClassName("DerivedNiwtComponent");  
    newComp.superClassName = newComp.saveSuperFunction("doSomething");  
    newComp.doSomething = doSomethingElseDef;  
    return newComp;  
}  
  
var derivedObject = new DerivedNiwtComponent ();  
  
//call the overridden version of a function  
derivedObject.doSomething();  
  
//call the superclass version of the same function  
derivedObject[ derivedObject.superClassName+"doSomething" ] ();
```

Listing 4.1: Finding a superclass definition of a function in NIWT components.

Using the variable `this.superClassName` is only good as long as it is not overridden. Furthermore, by default, it only points to the immediate superclass making it impossible to point to any exact level of inheritance. This problem is solved in by returning the prefix when calling either `saveSuperFunction()` or `addClassName()`, the class name of the immediate superclass at the time the function is called. This prefix can then be kept as a new variable for each function saved. These variables are inherited and can be reused, ensuring the correct prefix for the desired definition at the desired level of inheritance which allows several different definitions from several different levels of inheritance can be used.

(4) Special Case: Destructors

Destructors must be called from leaf to root on the inheritance tree, or else a resource can be destroyed in a superclass definition that is required by a sub-class definition. Whereas the string-addressed strategy grants access to all definitions, it does not ensure the order. Saving function pointers in arrays indexed by integer ensures the correct order, however

doing so breaks the scope resolution of the keyword `this` (see section 87).

There is a solution to this: the same integer-indexed array is used to keep the member names of functions, not the pointers to the definitions themselves. They are then retrieved in reverse order as string-indices to member functions which are thus executed in the correct order. This is made possible for a special-case function available to all NIWT components: `saveDestructor( "memberName" )`.

4.1.2 Static Members and Classes

In Java, a class that contains only static members can be called a static class. [2][96][97] call of JavaScript objects “classes” though they are really global variables: this is in fact closer to the notion of a static class. JavaScript does not support static members *per se*, although using the `.prototype` property can produce this effect. This is problematic when using the Factory Pattern as all classes will then be affected by changes without any possible filtering. A variable of type `Object` on the global scope can be used to produce fully static classes to group related member variables and functions where instantiation has no effect. Naming conventions are to be established to make the pairing of instantiable class and static object more evident (e. g. `SomeName` for the class and `SOME_NAME` for the related static object – see Appendix D). Otherwise, programmers must use numerous individual variables on the global scope and both encapsulation and scope are lost.

4.1.3 Multiple and Removable Inheritance

Important JavaScript properties open the door to three metamorphic aspects (see section 2.5):

1. Metamorphosis of interface.
2. Metamorphosis of structure (composition).
3. Metamorphosis of implementation.

The requirements for mashable applications and container independence imply that any

one, or all three of these aspects can manifest at run-time. Metamorphosis of interface and implementation are of particular interest as they are rarely encountered.

Interfaces in Java provide contracts requiring that any class implementing them define the functions listed therein [96][11]. Multiple inheritance in C++ provides both the interface and the function definitions [121]. Neither strategy affords adding inheritance at runtime and neither strategy supports removal. The programming paradigm used to build NIWT and CSOS provide multiple inheritance with definitions and removable inheritance at run-time with the Plug object.

(1) The Plug Object:

Consider the following properties of both JavaScript objects and JavaScript closures:

1. JavaScript supports looping through objects or closures as though they were arrays which makes it possible to find all members (variables and functions).
2. The declarative nature of JavaScript grants support for adding ('injecting') new members (both variables and functions) at runtime.
3. The support for function pointers makes it possible to add replace function definitions at runtime.

The Plug is a new type of object that uses these properties to provide runtime metamorphosis of interface and implementation. Multiple inheritance is therefore a matter of augmenting classical inheritance with one or more Plugs.

Figure 4.4 shows a Plug and a JavaScript class. The above properties afford plugs with the following actions:

1. **Hard plugging:** forcibly defining member variables and function pointers (replacing them, where they already exist) in a class.
2. **Soft plugging:** defining member variables and function pointers only if they do not yet exist in that class (Figure 4.5).
3. **Unplugging:** forcibly deleting member variables and function pointers (Figure 4.6).

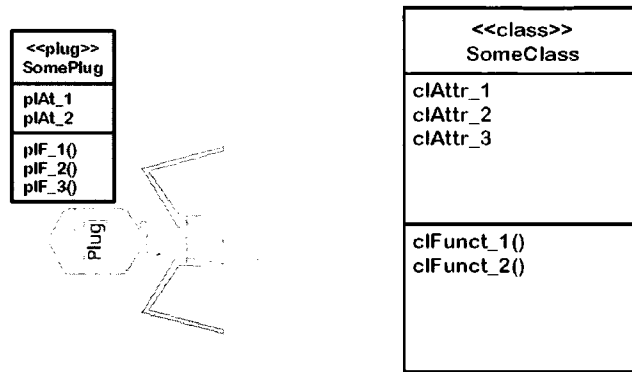


Figure 4.4: Plug Object and JS class (closure).

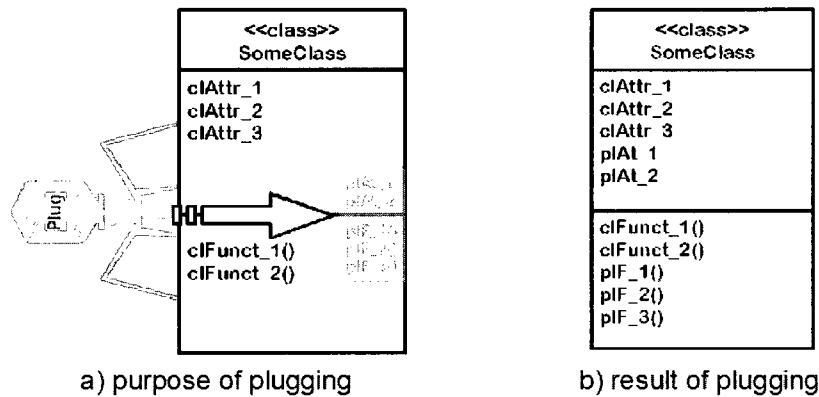


Figure 4.5: Plugging augments a class with its own members.

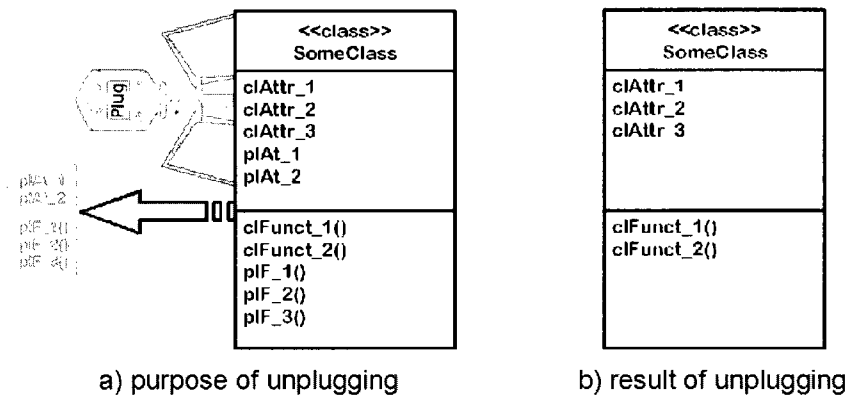


Figure 4.6: Unplugging removes the Plug's members from a class.

The result of plugging is a class that exhibits augmented identity and behaviour. Because there is no distinction between class definition and the constructor in JavaScript closures, Plugs can be applied anywhere in the object's life cycle. Because all members are public,

any object can plug something into (or unplug from) another object. As long as a pointer to the target object is provided, the plug will inject its members, values and definitions into the target (Figure 4.7).

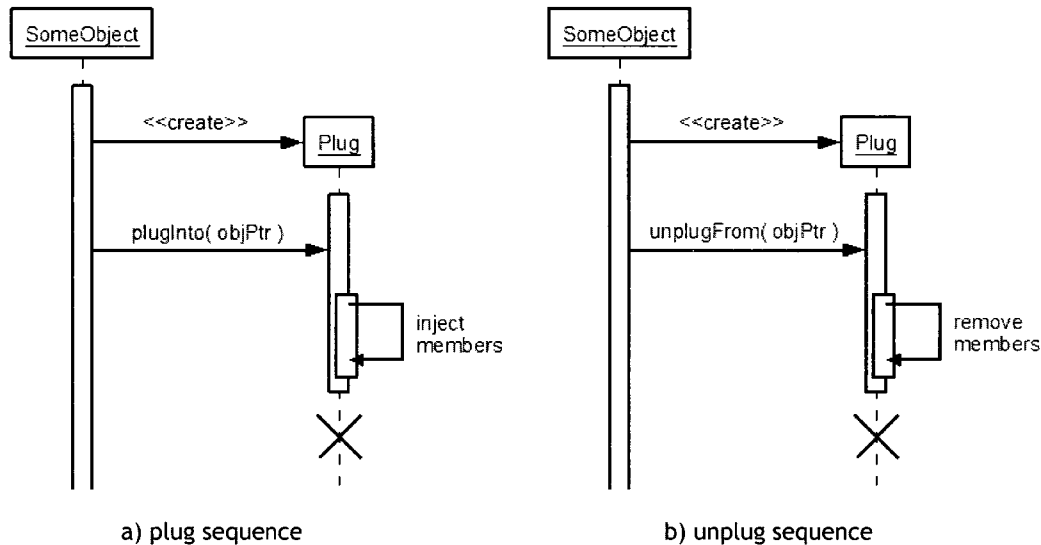


Figure 4.7: An instantiated object using a Plug on another instantiated object.

(2) Active Plugs

There exist situations that require specific actions must be executed in association with the plug at either plug time, unplug time or both. Two very important examples of such actions are:

1. Saving superclass functions when overriding them.
2. Performing server-operations related to the specific class.

The Active Plug (Figure 4.8) therefore differentiates members to be injected or removed from those required for execution. Figure 4.9 shows the sequence of events involved at both plug time and unplug time.

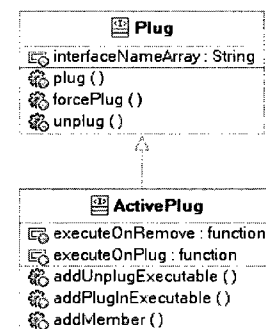


Figure 4.8: Active Plug

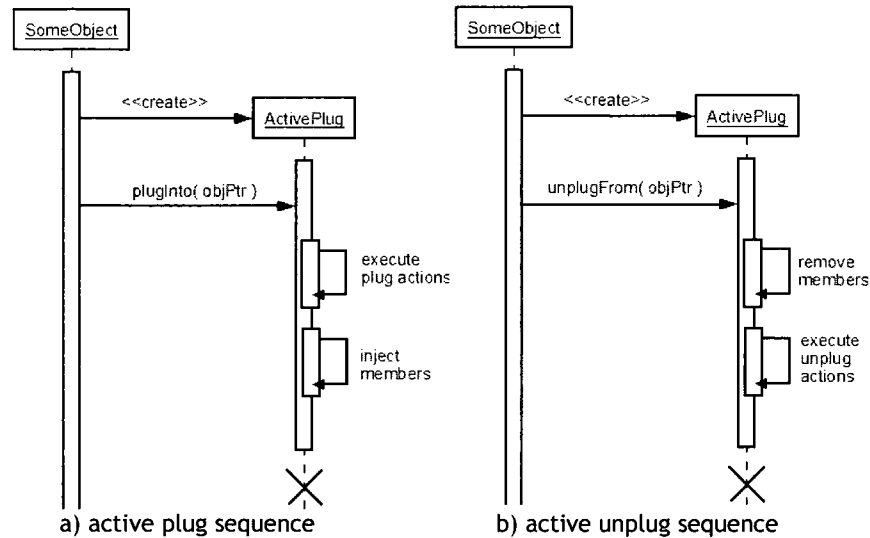


Figure 4.9: Active Plugs allow execution of specified actions.

(3) *Dynamic and Volatile Inheritance*

Java GUI developers are used to defining applications as sub-classes of an existing GUI component. This thesis proposes a similar method whereby the Application class requires a specified GUI component to be its superclass. The difference with Java is that the superclass is unknown before construction time and changeable at runtime. These two faculties respectively define dynamic and volatile inheritance.

Applications need not be tied to any single visual display container. For example, a word processor can be included as a docker in a slide-show builder or an accordion panel of a communications tool. End-users can thus start an application in a default container, and move it to be part of another application which places it in the required container type and location.

JavaScript supports building functions from strings which allows new constructors to be defined at run time. This is used to dynamically construct the application's superclass from a string passed on as a parameter. Figure 4.10 shows the sequence involved with changing a superclass: a new application is instantiated from the desired superclass which is restored with the data from the current object which is in turn destroyed.

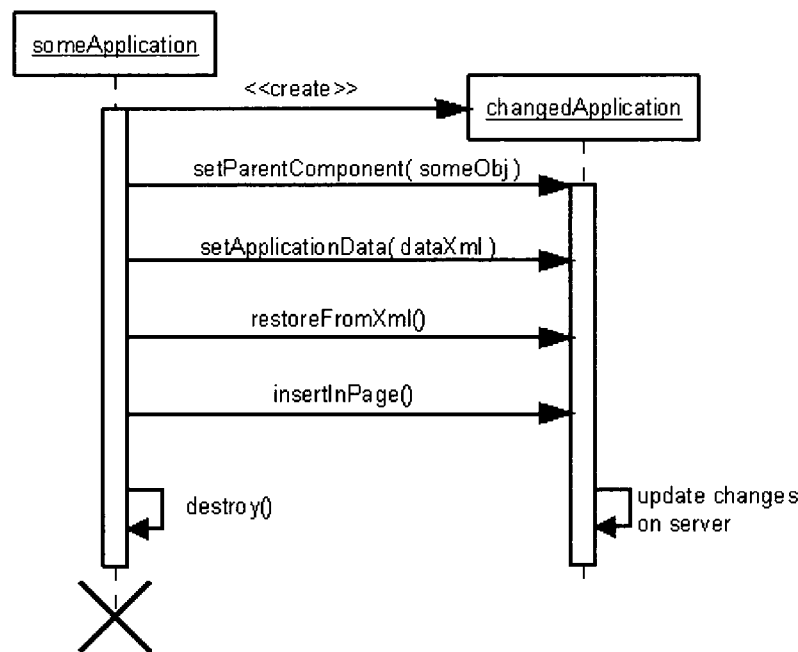


Figure 4.10 : Sequence for changing a superclass.

Strictly speaking, this is not changing the superclass as a new object is created and the original object is destroyed. But in terms of the server, it remains the same object as the application's server ID and data are maintained: only the container type needs to be updated to the server's existing persistence mechanisms. Also, the mechanism is built to ensure full transparency to programmers, again giving the impression of an actual change in superclass.

4.2 Standardized MVC

In response to sections 2.14.1 and 2.15.14(3) this thesis proposes a programming paradigm where MVC is the main goal, giving both the programmer and the user-environment means to call member functions on any component without requiring the user to perform an action. The user environment and/or the server-side is capable of enquiring on the state of certain parameters while being made capable of changing the GUI through programmed routines. Such routines are crucial to real-time collaboration (see section 5.7).

Figure 4.11 shows that only the view¹⁷ need be in HTML and that it can be generated by the controller. The data model consists exclusively of JavaScript variables. Where such data need be HTML, the JavaScript variables can keep it as values (e. g. a URL for an image or a string for code). Having the controller completely separate, responsible for server communication and also responsible for the generation of HTML code considerably facilitates other components querying information and calling functions on it. How components find each other is discussed in section 4.3.1.

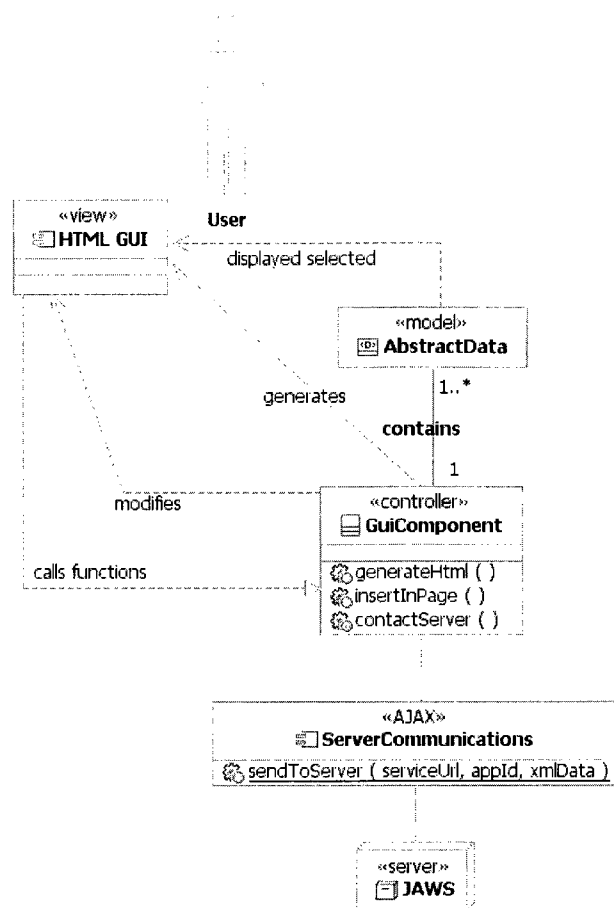


Figure 4.11: Clean MVC separation in facilitates communication with and control from the server.

4.2.1 Communicate

Data, not HTML Content

Typically, AJAX requires servers to send HTML content generated on the server-side [2] [51] and again HTML content generated by these applications also contains other AJAX requests. Therefore, the view contains the controller which is a violation of the MVC pattern.

In browser-to-browser collaboration, it remains tempting to extract and send HTML

¹⁷ The word 'view' is used by data-base science to describe sub-sets of data whereas the design pattern sometimes uses it for the GUI. This thesis will use it for the former and 'display' for the latter except where reference to the 'V' in 'MVC' is required.

content when it concerns purely visual elements. Consider Figure 4.12 which shows a drawing application that can be used in collaboration. Elements from the drawing do nothing more than appear, get deleted and moved. The drawing elements are sent from one participant to another as they are created, and so can be simply inserted directly from the HTML definition, but when they are deleted or moved, they must be identified in order to replicate the changes to other users. If the drawing element is purely HTML, then identification of exact elements on remote hosts becomes problematic.

Reconstructing JavaScript components from its data remains the best way to ensure object manipulation. Sending a signal with only the data required to replicate the action (e. g. create a draw-shape or move it) best ensures the extensibility of use-cases.

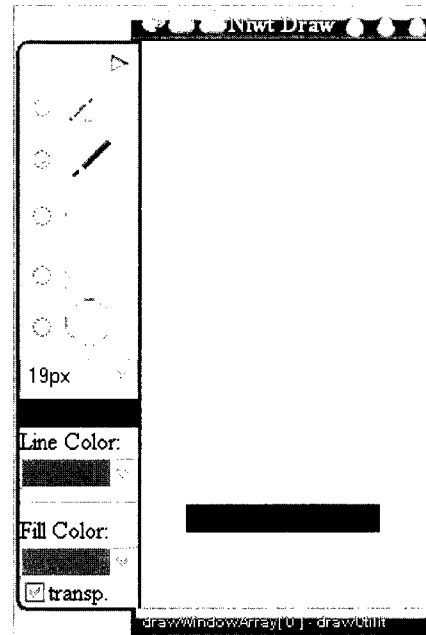


Figure 4.12 : Content that would likely be sent as HTML, not XML.

4.2.2 *Thick Client: Roles*

The various guides to RIA development, notably [2], have the controller neither on the client nor on the server alone, but in a combination of the two through AJAX. This has been shown to create a lot of network traffic. This thesis defined components such that the model and the controller are fully contained by the client (browser) reserving communications to the server for important exchange of state information, signals between users, and data updates. To achieve this, a clean distinction between Model, View and Controller is obtained by ascribing each component to HTML/CSS, JavaScript and XML.

(1) *Roles of HTML/CSS and XML*

HTML/CSS should provide nothing more than the display, which includes access to the controlling functions. They should provide neither the model nor the controller itself.

HTML reflects the view, doing nothing more than displaying it and giving access to the manipulation functions that relate to it.

XML could be considered the data model, but that would be a misrepresentation of its purpose which is to carry self-describing data between applications [44]. Data-centric XML documents are created to in response to database queries or to communicate a process information. Therefore, any one XML document is really only view of the data model, not the model itself. The transformation from data model to an interactive user-display, is therefore a multi-step process whereby the data view is extracted from the model, carried and communicated to its destination by XML, and then translated into a display in accordance with the program logic. These steps may be either server-based, client-based or both.

(2) Role of JavaScript

JavaScript serves as the client-side controller. Notwithstanding Adobe Flash or Java which are not directly a part of the web-document, JavaScript is the only language that can be used by a RIA programmer¹⁸ and therefore will serve several purposes. Because the support for AJAX in browsers affords JavaScript with the capability to parse, create and manage XML documents as well as perform logical routines based on the content, JavaScript will play a central role in handling the data model. Because OOJS keeps members which represent the sum total of an application's data and functionality, JavaScript plays a central role in the data model itself. Finally, because a browser's ECMA engine affords JavaScript with a direct access to the web-page's DOM and CSS properties, JavaScript plays a central role in both creating and managing the views and displays.

(3) Role of the Server Application

The server application plays a very important role in MVC. AJAX was defined to vehicle

¹⁸ There are indeed other functional entities that can be used by a programmer such as XUL and XPCOM. But the purpose here is to discuss the mechanisms which typically develop browser-contained RIA.

XML between client and server, both ways. Robust architecture requires a thorough analysis of roles and output of the various participating technologies. By locality of reference, the server has the responsibility of making decisions based on business logic whereas the client has the responsibility of performing client-centric tasks; in terms of a desktop environment, this includes starting applications and generating the corresponding GUIs. Following this architecture results in network traffic only when it is required, resulting in a flow of data transformation from the model to content to the view to the display.

The server application's acceptance of a client-side request therefore stimulates business logic to respond with content that can be either data or commands to be executed by the client. The client, on the other hand, has the responsibility to interpret the response and perform accordingly.

4.3 Component Orientated Programming

Standardizing MVC is only the first step in ensuring fully reusable software components. The Component object is the root of all inheritance in NIWT and is endowed with:

1. Required routines for inheritance and introspection (section 4.1.1(3)).
2. Functionalities and identities to act as Observers and Subjects (section 5.4).
3. Required members to build a composition tree.
4. Inherent variables and routines for collaboration.

Point #3, the composition tree, is important to programming new applications as they, or parts of them, can consist of several other components, each of which is capable of performing its task independently. Each component can also collaborate with a component from the same application or another one, locally or remotely. How to find these components is therefore a critical issue, compounded by structural metamorphosis which is central to the dynamic definition of mashable applications by the user.

4.3.1 Scope Decomposition of Sub Components

Collaboration and interoperability assumes that any component is capable of finding pointers on (or references to) the other components with which it must communicate. Many of these components are ultimately rendered in HTML for user interaction thus requiring the HTML elements to find the corresponding JavaScript components that perform the required actions. This is taken for granted by developers using Java, dot.NET, Delphi and the like, but DHTML poses certain problems which are discussed below.

(1) General Indexing Schema

In terms of structural metamorphosis, the composition of component oriented programming as presented here implies that a tree is constructed from the root component (e. g. the application proper) to all leaf sub-components (e. g. the actual fields and controls manipulated by the user). This tree has uneven levels of sub-composition down to the leaf as exemplified in Figure 4.13 which gives a highly simplified composition tree for a typical user application built with dot.NET technologies; it omits recursive items (e. g. sub-menus) etc. yet the complexity of the tree structure becomes readily apparent.

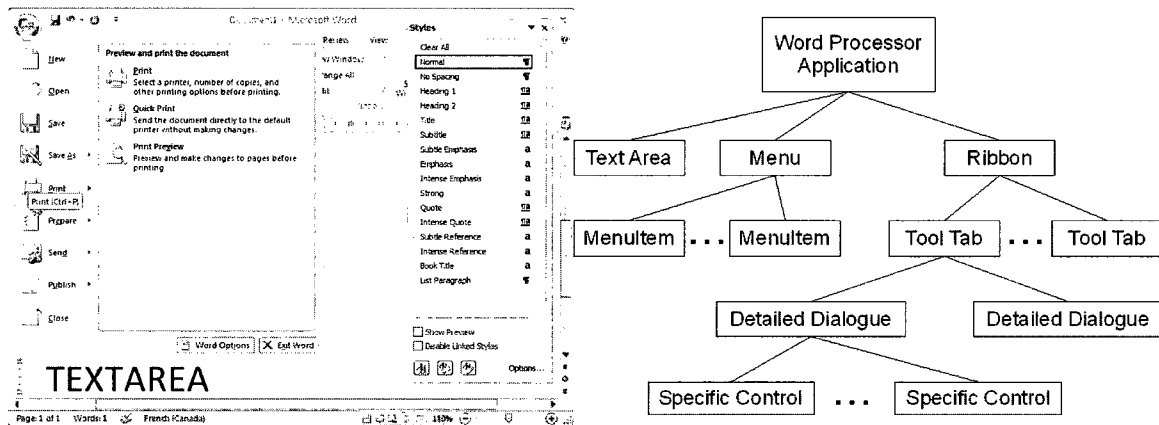


Figure 4.13 : Composition tree of a typical dot.NET office application.

All NIWT components are derivatives of the Component object which is endowed with:

1. an array of sub-components,

2. a pointer to its parent component and
3. a variable that keeps its index in the sub-component array of its parent component.

These three points form a doubly-linked list and implements the composition tree.

NIWT often uses introspective properties to apply a change or perform an action on a given object: it must therefore be possible to reference a given component using any level of class derivation. Figure 4.14 shows how each class definition provides a global-scope array and a member with the index of each instantiated object in that array such that a specific object can be found at any level of derivation.

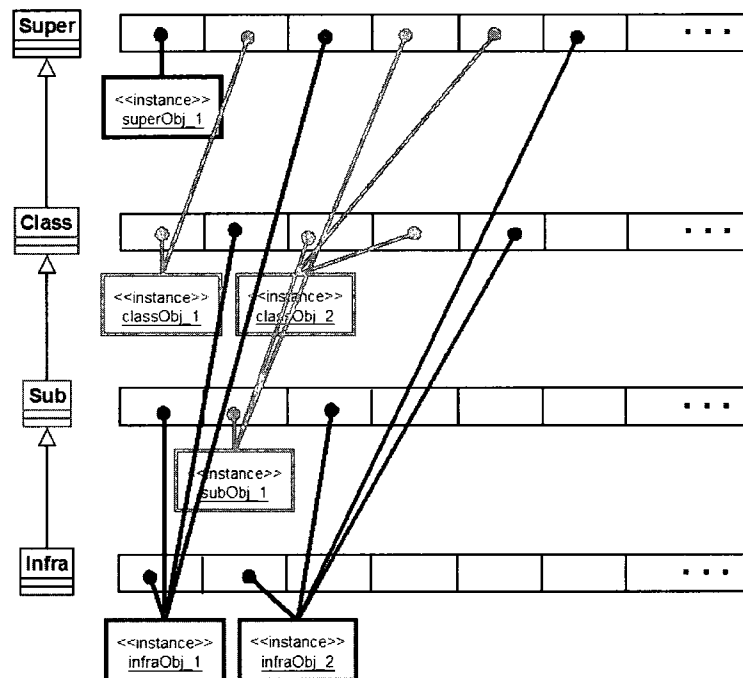


Figure 4.14: Inherited global-scope arrays of pointers to objects.

(2) Indexing for Interoperability and Collaboration

Even if a JavaScript controller object generates and inserts the HTML code, HTML is executed entirely at the global scope and retains no connection to the JavaScript controller whatsoever. Therefore, means are required to obtain a global pointer from a member object which compensates for the depth of composition (i.e. a member of a member of a member and so on) such that it can be called from the HTML.

Listing 4.2 illustrates such a composite global pointer with a commonplace example: consider a wizard with a button to validate entries. This wizard consists of a dialogue window that contains several panels, each consisting of several tabbed panes, each containing several forms, each in turn containing several fields. Any one of these items can be found by index in the sub-component array of its parent; with the window at the global scope, then a particular form item can be found by recursively addressing the corresponding member arrays. Listing 4.2 a) shows how this is done in JavaScript whereas Listing 4.2 b) gives the equivalent in HTML code. Note that the indices are represented by variables which must translate into concrete integers in the HTML code; how this will be discussed in Appendix E, String-Code Programming.

a) in JS:

```
windowArray[ winInd ].memberPanels[ pnlInd ].memberForms[
    frmInd ].memberItem[ itmInd ].validateEntries()
```

b) inside the HTML:

```
<button onclick='windowArray[ winInd ].memberPanels[
pnlInd ].memberForms[
    frmInd ].memberItem[ itmInd ].validateEntries()' ">
>Validate</button>
```

Listing 4.2 : Scope decomposition for an item in a form in a panel in a window.

(3) Scope Resolution of this Keyword in JavaScript and HTML

Developers of Java, C++ and so on are accustomed to using the keyword `this`, but it must be noted that DHTML yields very different resolutions for the `this` keyword not only between HTML and JavaScript code, but also in different styles of JavaScript coding.

The scope resolution of `this` in JavaScript always refers to the exact function or object in which it is located, not the object instance. Listing 4.3 shows the scope resolution causing problems when defining functions inline. The keyword `this` used in the `alert` has a scope that resolves to the new function, not to the closure (i.e., `this` equates to the member `showVar`, not the object that owns it). This is an important difference between classes and closures: the new function `this.showVar()` is also a closure. However, using the C++ header style definition of the same class in Listing 4.4 preserves the scope

of `this` and the alert will show “Hello World!”.

```
function someClass() {  
    this.someVar = "Hello World!";  
    this.showVar = new function() {  
        alert( this.someVar );  
    }  
}  
var someObj = new someClass();  
someObj.showVar();
```

Listing 4.3: 'this.someVar' will show 'undefined'.

```
function someClass() {  
    this.someVar = "Hello World!";  
    this.showVar = showVarDef;  
}  
  
function showVarDef() {  
    alert( this.someVar );  
}
```

Listing 4.4: C++ Header-Style Closure Definition.

Another significant problem: in the HTML definition, `this` refers to the DOM element, not the instance of the JavaScript instance that created the HTML. Very careful attention must therefore be paid when coding and the general indexing schema defined in this thesis must be used.

4.3.2 Hierarchical HTML

HTML is a character string parsed by the browser when it is inserted in the web-document. It can therefore be built hierarchically from class to sub-class and so on. This requires the HTML builder to recursively call the superclass definitions. This is made possible by the ability given to NIWT components to invoke superclass definitions as detailed in section 4.1.1(3). Another positive result of this is that such things as DOM element ids, mouse and keyboard event handlers are inherited and automatically inserted in the HTML.

The current development practice for RIAs is to register event handlers directly with the HTML element. This opposes the MVC pattern insofar as functionality is the domain of the controller, not the display. With the paradigm proposed by this thesis, the HTML code is predefined with calls to the principal event handling functions (mouse down, up, over, out, click, double click, scroll and keyboard actions). By default, these functions do nothing and application developers override them or augment them at will. Again, saving superclass definitions and overriding them allows a concatenation of inherited functionality; that is, the instructions performed of a given member function are

iteratively augmented.

4.4 Contract Object

OMG UML describes interfaces as being contracts of services rendered in accordance with parameters given [11]. OO and SOA use a variety of service definitions including Interfaces, Document Type Definitions (DTD) and Schema to describe available functionality [123]. Contract objects are a new service descriptor for JavaScript components which follow this concept and provide a definition used to automate a variety of ‘response scenarios’ to any situation supported by the contract. In particular, these Contract objects are used to:

1. Publish services in the form of events fired and/or available functions.
2. Automate RPC marshalling, especially in real-time collaboration.
3. Automate proxy building in browser-to-browser ORB (see section 5.8).
4. Replicate an action in the synchronization of local copies of the same application instance, notably in OLE (section 5.7.2(1)) and Single Data Multiple Display (SDMD, section 5.4.2).

Each NIWT component has a member Contract object that describes the functions required to replicate specified actions as well as the required arguments for each function described. Deriving a superclass inherits its Contract and enables the addition of events to it, completing existing events by assigning new functions that respond to them or reassigning function pointers for the purpose of overriding.

The contract is therefore defined as follows: let A be a component, E be the set of events fired by A , F be the set of functions required by every E_i , and P be the set of parameters or arguments required by each F_j with $\{i, j, k\} \in \mathbb{N}$, then:

$$contract = \bigcup_{i \in A} \left[E_i \bigcup_{j \in E} \left(F_j \bigcup_{k \in F} P_k \right) \right] \quad \text{Equation 1: Event Contract}$$

Equation 1 is coded as a JavaScript object (JSON) in the manner of Listing 4.5.

```

this.contract.eventName_1 = {
    function_1_1: [ param_1_1, ... , param_1_n1 ],
    ....
    function_1_m1: [ param_m1_1, ... , param_m1_o ]
};
...
this.contract.eventName_n = {
    function_n_1: [ param_2_1, ... , param_2_n2 ],
    ....
    function_n_n: [ param_m_1, ... , param_m_o ]
};

```

Listing 4.5 : Building the Event Contract

4.5 DnD: Data Transfer & Event Firing

Recall from section 2.15.14(4) that the purpose of DnD is to establish an association between objects to transfer data between them or start another specified process. Three participating classes can be surmised from the pattern:

1. **The drag-source:** visually, the object being dragged. More importantly, the first object for potential association.
2. **The drop-target:** visually, the object on which a drag-source is being dropped. More importantly, it is the object that will decide if the drag-source is acceptable for association and action.
3. **Drag-drop controller:** typically a singleton associated with the mouse pointer; in NIWT, it is the DndController object. It keeps a reference to the drag-source and manages the DnD finite state machine.

Dropping a drag source onto a target establishes a link between two objects which can then proceed to perform the actions specified by their relationship. Visual re-association of the drag-source is frequent (e. g. a file icon sent from one folder to another), but not required. The basic states for DnD (shown in Figure 4.15) are:

1. Off
2. Dragging
3. OverTarget
4. Validating
5. DropHere

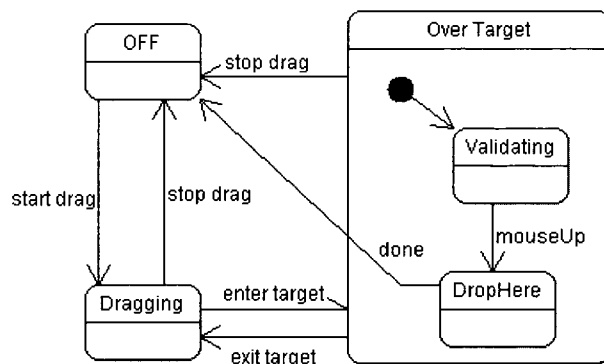


Figure 4.15: FSM for DnD.

The DropHere state can consist of any routine specified and is not limited to the two objects (drag-source and drop-target). An important concept, in terms of building GUIs is delegation to another drop-target which is used by NIWT to send applications to other desktops and cross-desktop mashup, which are results of the application of the pattern's algorithms; they will be described in section 6.2.5.

The Validating state requires a list of acceptable sources which is built by calling the function `addAcceptedSource( ClassNameStr )` for each acceptable class. Remark that using inheritance here saves trouble as all subclasses of an acceptable class will also be found acceptable through the test `objPtr.isInstanceOf( ClassNameStr )` defined in the NIWT Component class from which all others derive; similarly, instead of a class name, an accepted source can be an interface (Plug) name. All that remains is for application developers to define the DropHere state function `dropHere()`; the default definition only cancels the finite state machine for DnD.

Chapter 5

Collaboration and Interoperability:

Architecture and Algorithms

This chapter will discuss the components and algorithms that support real-time collaboration and interoperability. The notions of sharing and collaboration are often confused in product advertisement; this thesis makes the distinction by defining collaboration as a completely dynamic and active and RT interaction process whereas sharing is static and passive, consisting only of making resources accessible. Collaboration is also asynchronous inasmuch as the actions of one participant will usually require no feedback from any other participant. In the context of this thesis work, interoperability will include two aspects: different applications working together and different technologies communicating for common tasks.

5.1 States of Application Visibility

(Collaborating, Shared, Public)

The following definitions clarify the architectural implications of different types of visibility assigned to applications.

1. **Sharing:** Making a resource available to a select group of other users. Any other user who access it obtains a new and independent copy; no interaction is defined between users on that resource.
2. **Public:** A single copy of a resource that is available to all users. By definition, there is no implicit interaction on the resource. However, if edition permissions are granted, then changes must be persisted and reflected to other users.
3. **Collaborating:** Multiple users concurrently operating on the same copy of a resource. This implies that any and all changes are reflected in RT to all other users (as permissions allow).

The combined notions of sharing applications and collaborating on them require some modelling to determine the underlying mechanisms that need to be activated at any given time. The Venn diagram in Figure 5.1 sets the stage for a finite state machine on application updating activity (Figure 5.2). The Dormant set simply signifies that an application is not currently running. This does not, by definition, exclude it from being public or shared, but in such cases there would be no RT impact on it from other users' modifications until it is taken out of the Dormant state. The Single Copy set is the more habitual state of an application running on a desktop environment such as Windows. The set Multiple Synch. Copies describe multiple copies of the same application running on the same client at the same time where changes made to one copy are immediately replicated on the others (see section 5.7.2(1)).

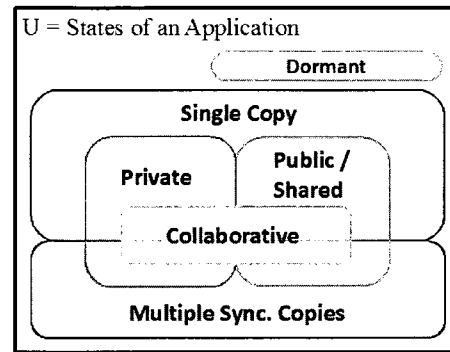


Figure 5.1: Application States.

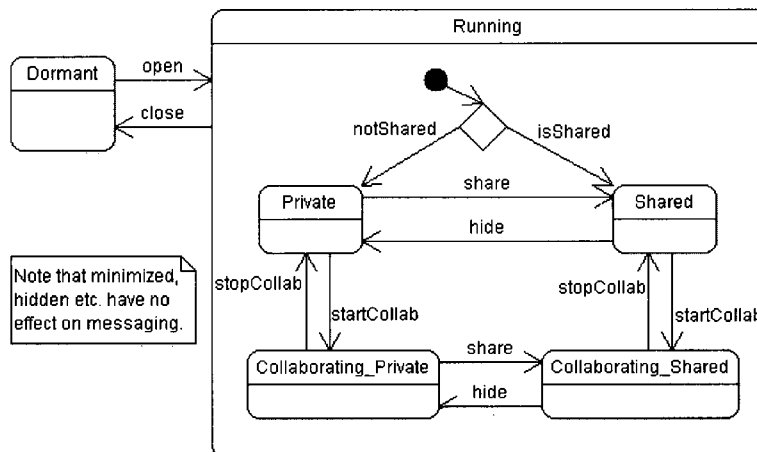


Figure 5.2: FSM for Application Updating

The Private set includes applications that are not accessible to other users while the sets Public/Shared and Collaborating conform to the definitions above. It is assumed that to be shared, an application is automatically public as well as the converse.

5.2 Access and Indexing

Recall the indexing schema discussed in section 4.3.1. The application states enumerated above imply obtaining a pointer on an application in the environment of remote users. This poses a problem: suppose a word processor application can be found by the pointer `wordProcArray[this.wordProcArrayIndex]`. Because word processors are capable of multiple instantiation, it is highly likely that the index will be the different in each environment of the collaborating users. To overcome this, a number of controlled identifiers have been devised between the client and the server:

1. The `serverId` member variable uniquely identifies all applications instances such that the server application can save and restore their data every time they are reused and also perform control actions on them.
2. The `shareId` member variable that identifies instances of the same application on the workspaces of different users when they are in collaboration. These remain distinct instances as users can keep a copy for independent work after the session terminates.
3. The `publicInstanceId` is a member variable that identifies applications that are considered the same, never distinct, no matter how many users have a copy of it on their workspaces. Therefore, any changes saved by any user will affect the unique data set for it on the server.

These indices are all defined server-side, which is responsible for managing access and communication. When components of the `Application` class are instantiated on the client-side, they are registered with the server for database persistence. The server application responds with a command to set the `serverId` of that component. This then allows:

1. Client updates to the server: data changes, state changes and destruction.
2. Server-side control of the client-side component: the specific component can now be found by the `serverId` and a pointer on it acquired to call its member functions.

5.3 Browser-to-Browser Client-Centric Collaboration

The Use-Case Analysis from the original project underlying this research described scenarios where data is barely an issue at all; instead, actions must be mirrored and shared controls provide feedback to other users when an action is performed. Therefore the purpose of collaboration as seen by the thesis research is not to merely synchronize data from one client to another but rather to synchronize the state of application located on the workspaces of all participants in a collaboration session¹⁹; this is done by replicating actions performed by an originating user to the environments of all other collaborating users. This distinction provides support for collaboration on data-centric as well as non-data-centric applications; to design support for the general case opens the door to reusability and transparency to application developers for collaboration on applications of unknown description (i.e. the application can do anything) and of unknown distribution (i.e. collaborating applications or parts thereof can be local, remote or a combination thereof).

5.3.1 Relay Architecture

Collaboration on multiple remote clients implies two issues that must be addressed:

1. Higher-layer transmission protocol between web-documents in remote browsers.
2. Capture of application state changes and mirroring them among participants.

For browsers to be able to send messages to each other, a connection must exist between them. Several solutions can be devised including, but not limited to P2P Applets, a shared memory architecture (server-side) or a message-box architecture. This thesis assumes a relay, the JAWS server, which implements a broadcasting service.

Capturing changes in application state is done on the client-side only with an architecture similar to that of [46] (Figure 2.2 on page 13). Figure 5.3 shows a very high level architecture of the browser-to-browser collaboration mechanism. The client-side provides

¹⁹ 8 refers to ongoing work in the field while Chapter 7 proposes future work in this area.

a communications layer consisting of both a browser-to-browser RPC Manager (details in section 5.5) and a Collaboration Manager.

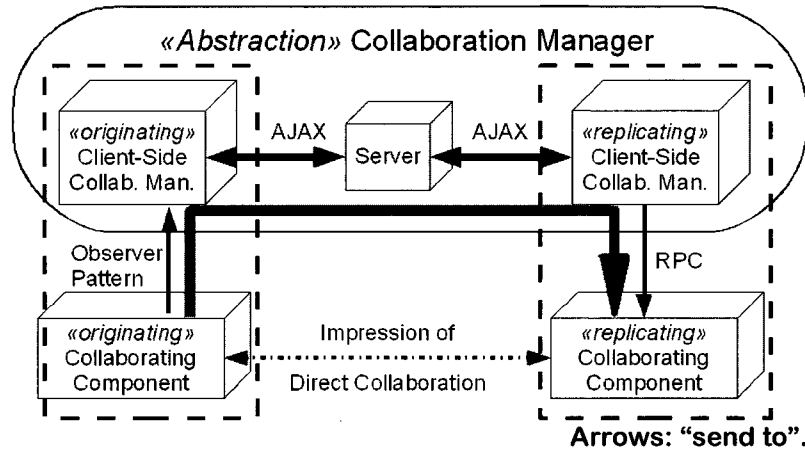


Figure 5.3 : Modules involved in BTB collaboration.

The browser-to-browser RPC Manager is a general case module used by the Collaboration Manager, which is specific to real-time collaboration sessions managed by the server for permissions control and communications filtering (i.e. what actions are allowed for broadcast and which ones aren't). The remainder of this thesis will abstract the top area of Figure 5.3, stating only the 'CollaborationManager' and a more detailed architecture of the client-side classes for real-time collaboration is given in Figure 5.4.

5.3.2 Many-to-Many-on-Many Relationship Collaboration

The NIWT/CSOS client must ensure application compatibility, communication resources and API support for the M3 relationship which is created and managed by the JAWS server. The client-side ensures compliance with the API by abstracting the main modules of SIP as defined by the server. Figure 5.4

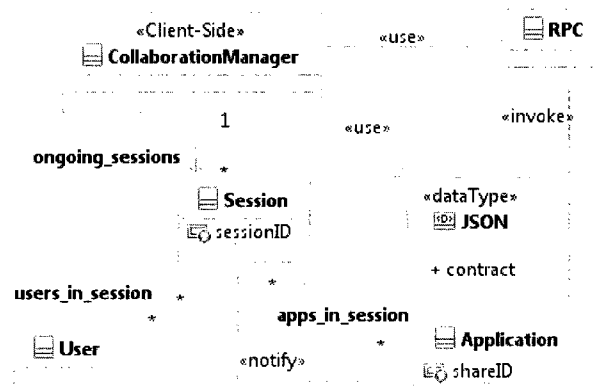


Figure 5.4: Client-side architecture to support the M3 relationship.

Shows the client-side components involved with real-time collaboration. The objects characteristic of M3 are the User and the Application, either or both of which can be located on other domains.

Client-side support of users located on other domains is a trivial matter as they are made accessible by the server which instantiates User and Avatar objects on the workspace. Applications located on other domains require resources and definitions that may not be available locally. In such cases, the dynamic resource loading mechanisms allow the new applications to be launched on the workspaces that normally don't have access to them (discussed in section 5.10.1).

5.4 Observer Pattern: Cornerstone in Collaboration

Both the Observer Pattern and the Proxy Pattern [3]²⁰ have proven to be cornerstones in client-managed collaboration in the project underlying this thesis and so they deserve a review here. Also, some modifications to the classic patterns were designed to reduce network traffic and automate critical routines.

Firstly, observers register for specific events or for 'all' events. Notifications send updates only the observers registered to the given event, reducing the number of communications performed. Secondly, an array of relevant values indexed by name is handed to the observers as an argument in the update function call. Observers thus receive relevant data, thus again reducing the need for communication.

5.4.1 Notification Object

Even without these modifications, for an observer to access data from the subject, a pointer to the latter must be provided. This is not automatic in JavaScript and some algorithms require data that is not necessarily relevant to the event, and therefore not

20 The Proxy Pattern is also known as the Remote Observer Pattern which will be favoured by this thesis to better compare it with the local Observer Pattern and avoid any confusion with other uses of the word 'proxy' (e. g. in the browser-to-browser ORB protocol described in section 5.8).

provided in the values array.

The NotificationObject was built to ensure the automatic collection of important information. This object adds key-value pairs to build the values array itself. It also supports the forwarding of events for algorithms that will be described in section 5.7.

UC-IC does not require the use of this object, but there are many advantages in doing so. In particular, it ensures proper execution of the remote observer pattern routines and for the automation of Action Replication (section 5.7). Figure 5.5 summarizes the sequence resulting from these changes.

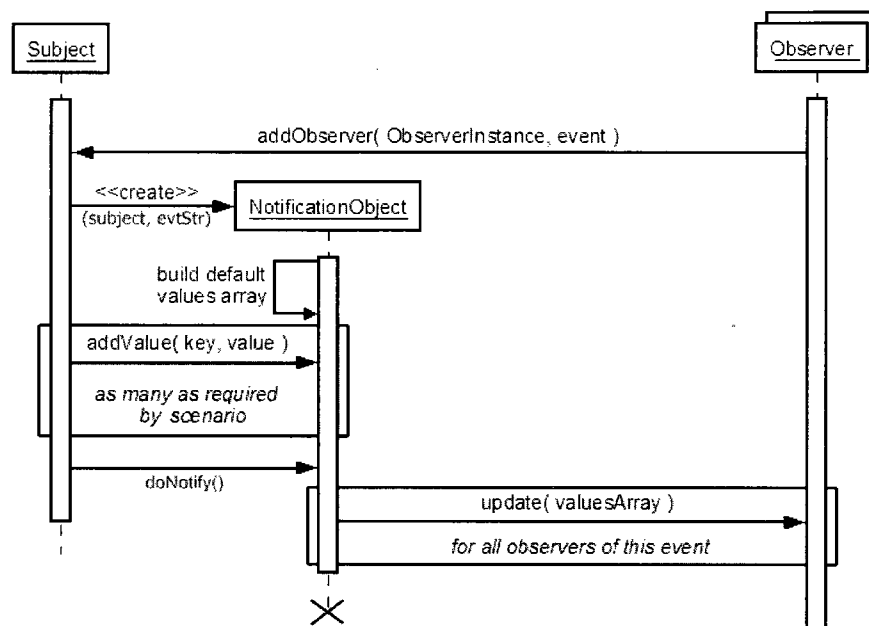


Figure 5.5 : Observer Pattern using the NotificationObject.

5.4.2 Single Data, Multiple Displays (SDMD)

SDMD is a special use-case for the observer pattern with the purpose of reflecting changes to the data model on all displays of its views. As introduced in section 2.14.1, the importance of MVC in decoupling view from data and control are fully shown when several views display and manipulate the same set of data. In such cases, multiple independent GUIs must be easily created, each reacting immediately to any change in the

data they present [93].

A common example shows how the SDMD pattern is taken for granted by PC users: having several file managers open at the same time. There is only one file system, but multiple copies of an application that both manipulates the file system and immediately reacts to changes in its state. Though they depend on the same single object, they each maintain independence of their own state (displayed location in the tree and so on).

SDMD is unheard of in web-programming inasmuch as the desktop metaphor is hardly the full equivalent of the PC desktop with multiple displays of the same model hardly ever provided. It is combination of two software patterns:

- 1.Strict application of MVC: the data object and the displays are distinct.
- 2.Observer Pattern: each display responds to notifications from the data object.

Figure 5.6 shows the various players in SDMD. Each one of several display instantiations (GUI components) observes and reacts to a single instance of the data model. The display is stereotyped as both controller and MVC because it has its own set of controls and the display need not be constrained to that one data model alone. By the same token, different displays can observe the same model which also has its own set of controllers for access and modification.

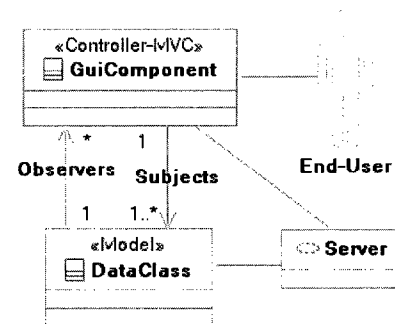


Figure 5.6 : SDMD Class Diagram

Any component can serve as both data and display to any other component and vice-versa a mirrored relationship. All that matters is a strict separation between Model (data variables) and View/Controller (HTML/JavaScript) in order to allow the differentiation and independence of instances. There is also a many-to-many relationship between the **GuiComponent** and the **DataClass** in Figure 5.6 which attests to the fact that a display can be observing several different data models concurrently. The result confirms the importance of de-coupling the aspects of MVC and of differentiating ‘display’ from ‘model’ as discussed in section 4.2.

The display and/or the model can interact with any other component on the local client-side and/or with the server-side and/or in collaboration with a remote user. None of this matters as the Observer Pattern will ensure that any fired event will be captured and handled by all interested parties.

5.4.3 Remote Observer Pattern Between Web Clients

The remote observer pattern is well known [3] but not implemented between distributed documents in web-browsers. The architecture used here consists of the RemoteSubject and RemoteObserver classes which are proxy counterparts to the Observer and the Subject respectively. The proxy objects use a broadcast module to manage the location of the actual objects they mirror and transmit messages giving the impression that the remote objects are actually local. A component can thus register as an observer of a remote subject and a subject can notify remote observers with no more coding or concern than it takes for local components. The broadcast module is the communications layer on the client side which interacts with the broadcasting mechanism on the server-side. How these proxy objects are created is left to developers; the architecture for browser-to-browser ORB (section 5.8) creates them automatically as part of the algorithm.

5.5 RPC Between Browser-Contained Clients

The client side is fully responsible for marshalling and producing the server-calls in accordance with its API²¹ as well as reconstructing and executing the incoming function call. To perform RPC from browser to browser, a relay between them is required in order to ensure the connection: section 5.3.1 discussed the server relay assumed by this thesis to broadcast marshalled messages and section 5.7, explains the protocol and architecture whereby both the Observer Pattern (local) and browser-to-browser RPC are used for real-time collaboration. In order to ensure reusability for all applications (existing and future) as well as transparency to application developers, here follows an explanation on how the

21 This server-side API was developed concurrently with the client-side as part of another thesis.

marshalling of RPC can be automated in JavaScript as part of the communications layer.

Marshalling requires the serializing of each function call including its arguments into a byte array for transmission and de-serializing them for reconstruction and execution. Because JavaScript supports the creation and execution of new function definitions from character strings, which in turn concatenates data primitives, marshalling is well supported. Any function call is essentially a combination of three items:

1. the function's identifier (or function name);
2. the owner of the function (instantiated object or void for global functions);
3. type-constrained arguments required by the function.

In terms of browser-to-browser RPC, the owner of the function is whatever series of pointers is required to find a global scope pointer on the function in the context of the specific instance of the component (see section 4.3.1).

Therefore, what is required for browser-to-browser RPC is a marshalling mechanism that will:

1. On the sending side:
 - a)identify the function name;
 - b)serialize all pointers required to identify the owner;
 - c)serialize the arguments (including type identification).
2. On the receiving side:
 - a)concatenate the pointers to rebuild the owner and function name;
 - b)de-serialize the arguments (including type restoration);
 - c)rebuild the function signature;
 - d)execute the function call.

Points 1.c) and 2.b) have a significant corollary: a given argument can be an abstract data type (e. g. a collection) recursively consisting of other abstract data types which must also be recursively marshalled. Assumptions and the marshalling process are detailed in

Appendix B (Browser-to-Browser Marshalling).

A RPC call can originate from either another client (in the case of browser-to-browser RPC) or by the server (in the case of administrative commands). In both cases, the issue of obtaining a pointer on a specific class instance is not trivial. In the architecture proposed, components are capable of multiple instantiation; this goes for complete applications as well as individual sub-components at any level of composition. Therefore, if, for example, three word-processors are running concurrently on the browser environment, but only one is involved in collaboration, how will the RPC know which one? Section 4.3.1 describes the referencing mechanisms to this end.

5.6 Deferred Synchronicity and Sequential Continuity

Invoking a function on a remote object using AJAX has a corollary on the local execution flow due to the ‘asynchronous’ part of AJAX by which any sequence of instructions making an AJAX call will *not* be suspended until the call returns. Ideally, protocols like RPC, RMI etc. are based on the intent to provide programmers with transparency of object location. That is, the programmer can assume that invoking a method on a remote object will behave in exactly the same way as it would were the object local. This assumption does not hold true in AJAX programming.

Consider the following program statements:

```
var x = someObject.getNumber();  
x = x+1;
```

They include the following assumptions:

1. The first line actually consists of two operations: a method invocation and an assignment.
2. The second line must not be executed before the first one has completed; that is, all of the actions implied by calling `someObject.getNumber()` must complete before assigning the result to `x` or else the incrementation will happen with a possibly erroneous value.

3. The return from the function call is followed by the next operation in the routine that made the call which then continues its sequence.

Let these assumptions be called “Sequential Continuity”.

In the above example, if `someObject` is local, then most compilers will build an executable that will support the assumption. But in the case of remote objects, sequential continuity must somehow be guaranteed by the network protocol as well as the compiler and/or interpreter as well as the runtime environment. In this case, both the environment and the interpreter are the browser's JavaScript engine and the protocol is AJAX; though AJAX does support synchronized calls, its event-based nature still eludes sequential continuity.

At best, AJAX supports “Deferred Synchronicity” [117] insofar as the response from the server application must be given to a response handler function. The response from the remote object cannot be used in sequence within the routine that called the action. This is no small concern and deeply impacts how software is designed. Whenever there is a chance that a component could be used remotely, then proper software design must take be wary of it.

5.7 Event Replication or Echoing Actions

Event replication (or echoing actions) is the principle mechanism by which real-time collaboration is supported in the thesis work²². It is also used to synchronize the state of equivalent components locally (see section 5.7.2(1)) and works by implementing the following definition:

Let A be an action on a component performed either by a user or a program sequence, E an event fired, F be a call to a function, C be the contract for the component on which A is performed (as per Equation 1, p. 91) and $i, j, k, l, m, n, o \in \mathbb{N}$ with $[m, o] \neq [0, n]$

²² The remote observer pattern has also been used but has been found less scalable and more difficult to maintain.

$$\text{then } \exists \left(\left[A_i \Rightarrow \bigcup_{k=0}^n F_k \right] \neq \left[A_j \Rightarrow \bigcup_{l=m}^o F_l \right] \right) \Rightarrow \left(\left[\bigcup_{k=0}^n F_k \Rightarrow E_i \right] \wedge [C_i \Rightarrow E_i] \right) \quad \begin{array}{l} \text{Equation 2} \\ \text{Event Replication} \end{array}$$

In Other words, for real-time collaboration to be supported on a RIA application, an application developer must:

1. Determine characteristic actions represented by distinct sets of function calls.
2. Ensure events are strategically fired in those actions (and provide the requisite values array).
3. Ensure an exact correspondence of distinct action and distinct contract definition for the action.

This is required to fulfil the architecture shown in Figure 3.4 following the recommendations of [46] (section 14).

5.7.1 Automation of Action Replication

Automation of critical processes simplifies the creation of collaborative applications²³ as per [46]. This thesis solved the problem by using the Contract object. This section discusses both the automation process and how event replication can be delegated to other components.

(1) The Contract: Key to Automation

Contracts are used by the CollaborationManager to build RPCs on notification from the Observer Pattern. In order to automate the process and promote both scalability and reusability, the CollaborationManager does *not*:

1. Limit which events can be replicated remotely.
2. Know how many functions are associated with each event, let alone what the function names are.
3. Know the number of arguments required, nor their type, let alone their arrangement.

²³ The Application class defines local services in a manner similar to the CollaborationManager. The descriptions in this section therefore apply to both, but mention only the remote use-cases for clarity.

Rather, the CollaborationManager depends on the values array from the update to be a hash with all required values indexed by keyword (hence the preference for using the NotificationObject described in section 5.4.1). Equation 1 and Listing 4.5 (p. 91) make it clear that, assuming all the values are in the notification's values array, a nested loop algorithms can extract the arguments from the array to rebuild the necessary function call signatures and perform a RPC for each. The actual reconstruction and execution of RPCs is discussed (listings and algorithm) in 5.5, RPC Between Browser-Contained Clients.

(2) Contract Delegates

As multi-level aggregates of components, collaborative applications pose yet another problem: RPC calls perform an action directly on the concerned component whereas real-time collaboration is concerned with the root component only. Delegates allow any component to define any other component for the RPC call; these may be sub-components or even objects unrelated to the application.

There are three types of delegates defined in accordance to possible use-cases:

1. the entire contract,
2. a specific event or
3. any single notification occurrence.

A combination of these causes a general delegate to be overridden by the more specific one. In all cases, delegates can be concatenated such that they provide a deeper level of sub-composition as per the indexing schema (see Listing 4.2 on p.88) and support the case of cascading notification discussed in section 5.7.2(3). The Collaboration Manager uses delegates to correctly rebuild the browser-to-browser RPC call to the correct component and the Application class does likewise for locally synchronization (see section 5.7.2(1)).

5.7.2 Mashups, Aggregation and Real-Time Collaboration

Recall the third model of mashups as explained by [18] in section 2.2. In response to the

deficiency noted in section 2.15.11, mashups are defined in this thesis to include RT data from user interaction with any given application. As a result, the COP architecture presented in section 4.3 allows the runtime definition of mashups in terms of a volatile aggregation of user applications as sub-components: mashups are therefore a specific case of component aggregation. This section will discuss the implications of aggregation on the event replication algorithms.

(1) *Local Synchronization and OLE*

Locally synchronized copies consist of replicated components that update each other in RT through event replication to ensure data integrity as well as integrity of state between them. There are two principal uses for locally synchronized copies:

1. Linux and Solaris users have long been accustomed to multiple desktops and being able to ‘pin’ an application on all desktops. Figure 5.7 shows a sticky note application shown on all desktops of a user environment. Changes made to one copy are reflected immediately on all other desktops.

2. Locally synchronized copies enable OLE in NIWT applications. Figure 5.8 Shows a drawing embedded in a word processor which is modified by a separate application.

When a component is made locally synchronized, its `updateValues` function is overridden to concatenate the previous definition with synchronization mechanisms which forward the event to the other copies which in turn replicate the event as detailed in section 5.7.1. Because each copy can be contained in different containers (e. g. one in a window, another in an accordion pane, another in a plain box – see section 4.1.3(3)) and because there is no requirement on what component can contain them, OLE is thus achieved.

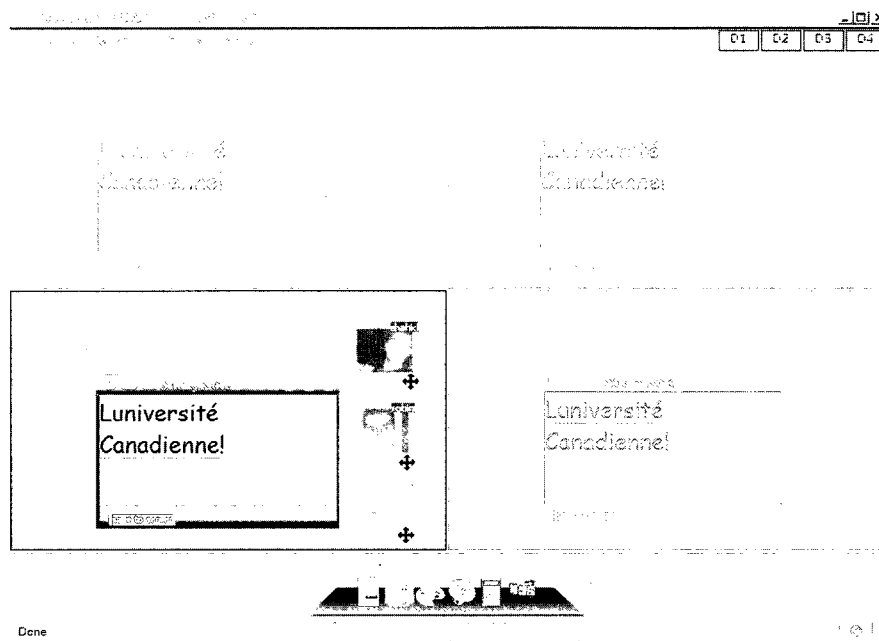


Figure 5.7 : Locally synchronized copies of one application shown on all desktops.

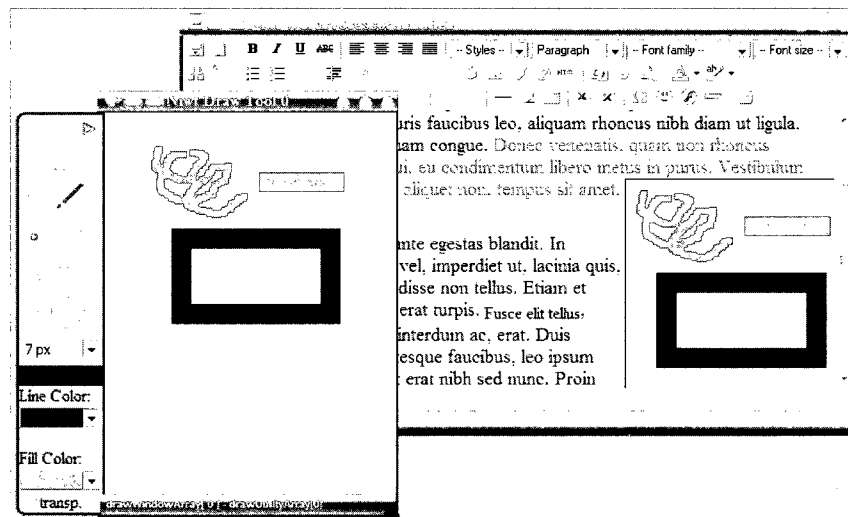


Figure 5.8: OLE support through LSC and Container Independence.

(2) *Dynamic Aggregation and Redundancy Evasion*

Any application can be represented as an aggregation or composition tree (e. g. Figure 4.13 on p.87). When a given application is involved in real-time collaboration, the run-time addition and/or removal of sub-components implies the following:

1. When one user adds a sub-component, a corresponding and equivalent component must be added in the environments of every other user participating in the collaboration session (*idem* for removal).

2. The actions performed by any component at any level of sub-composition must be reflected on the corresponding component of every other user participating in the collaboration session²⁴.

Point 1 is handled by the application life cycle discussed in section 3.1.6 and the mechanisms that support dynamic composition structure discussed in section 5.10.2. What remains is to design the support for echoing actions on corresponding components at unknown levels of the composition tree while ensuring this support will self-adjust to their dynamic creation and destruction.

Figure 5.9 shows an application A on one user's environment which is in collaboration with an equivalent application B on another user's environment. The 'Collaboration Manager' module shown is an abstraction of client-side and server-side architecture. A.1 is the root application with sub-components A.2 to A.5. and similarly for B. The only components that are actually of any concern are A.1 and B.1 as the others are composite sub-parts that can be added or removed at runtime without terminating the actual collaborating application. Some sub-components observe each other (e. g. Figure 5.9 A.2 and A.4) while others don't (e. g. A.3 and A.4). In all cases any event fired on any component A.x must be replicated on the corresponding B.x component.

24 This could result in collisions where two or more users perform conflicting actions. As mentioned in sections 2.15.2 and 7.3, this will be left to future research.

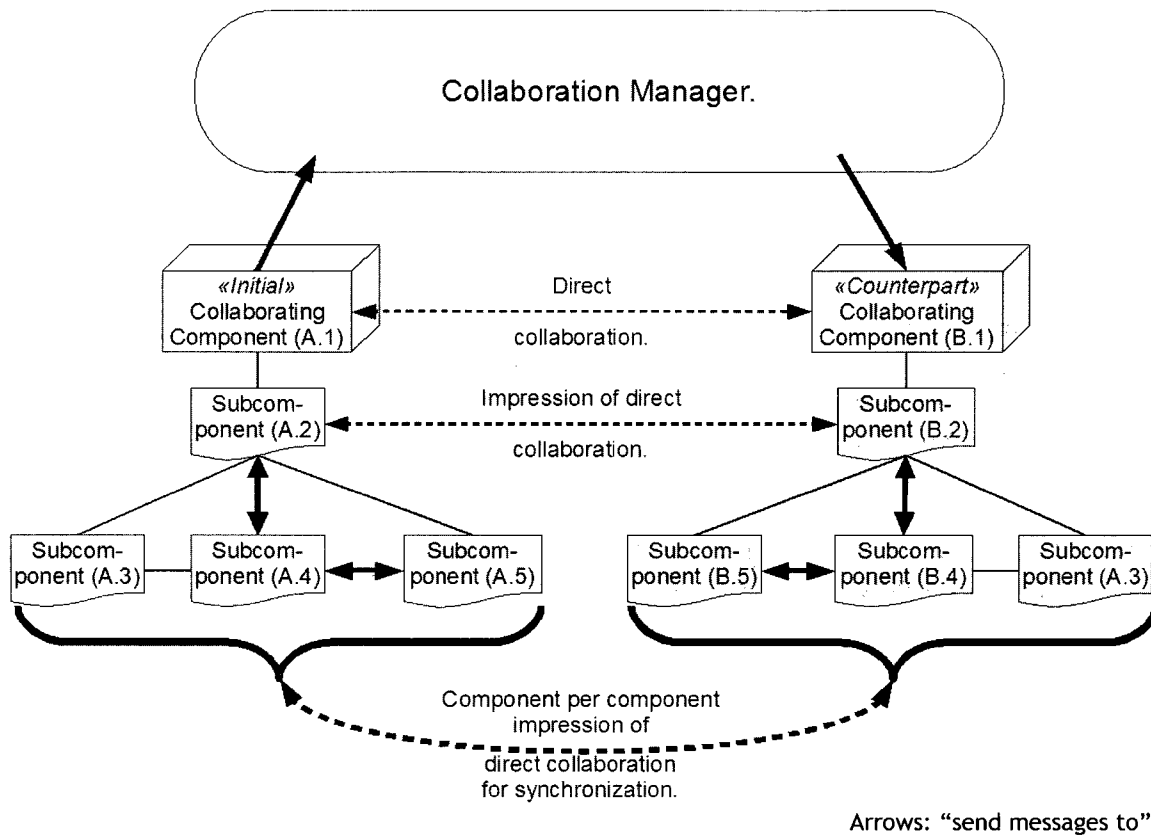


Figure 5.9: Example use-case map for BTB collaboration on a composition tree.

(3) Mapping Sub-Component Level Actions

If all levels of sub-composition notify their events upwards to the collaboration root (the actual component registered as collaborator), then all events reach the root and can be echoed to all collaborating users (Figure 5.10). Let this model be called “cascading notification” which may be an unreasonable assumption. More than likely, some events will not be observed up at least one level and therefore will not reach the root as shown in Figure 5.11. Let this notification model be called “broken notification”, the result being that events will not get echoed.

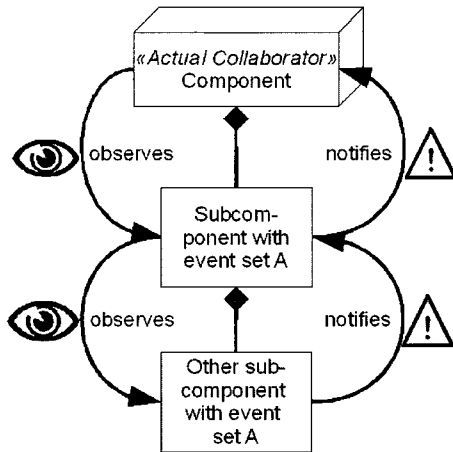


Figure 5.10 : Cascading Notification

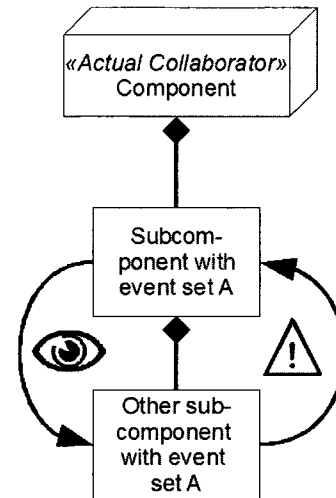


Figure 5.11 : Broken Notification

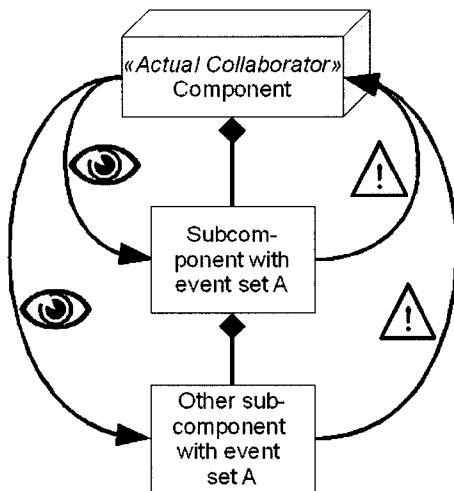


Figure 5.12 : Full Notification

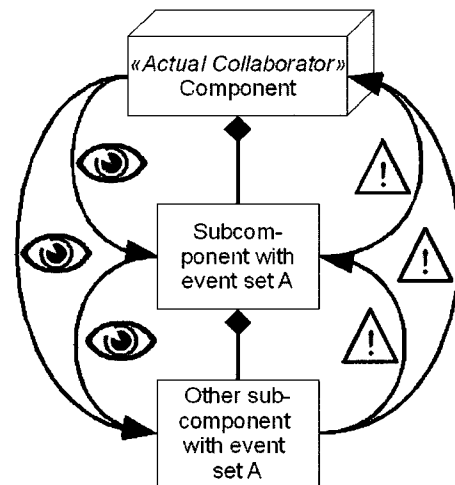


Figure 5.13 : Hybrid Notification causes redundancy.

Broken notification can be rectified if the root receives notifications of events fired by all sub-components at every level²⁵. In this case, any event can be echoed (replicated) to all the users participating in the collaboration (Figure 5.12). Let this be called “full notification”.

However, Figure 5.13 shows that if full notification coincides with cascading notification, let this be called “hybrid notification”, then redundant events will be sent to the root in a

²⁵ A simple algorithm that will ensure this can be executed when the root is made collaborative.

manner that can be modelled as follows:

Let C be the set of all components, E be the set of all events fired from any component, C_i be a specific component, E_i the set of events fired by C_i and $\{i, j\} \in \mathbb{N}$.

$$\aleph(Events_{observed}) = \sum_{i \in C} \left[\aleph(E_i) + \sum_{j=0}^i (\aleph(E_j)) \right] \geq \aleph\left(\bigcup_{i \in C} [E_i \in C_i]\right) \quad \text{Equation 3: Event Redundancy}$$

whereas what is needed is:

$$\aleph(Events_{observed}) = \aleph\left(\bigcup_{i=0}^n E_i\right) \equiv \aleph\left(\bigcup_{i=0}^n \left[E_i \setminus \left(\bigcap_{j=0}^i E_j\right)\right]\right) \quad \text{Equation 4: Redundancy Evasion}$$

On the perspective of any given event fired, the root will receive as many notifications of the event as there are components firing it. With cascading and hybrid notification, this means that:

$$\aleph(Notifications_{given\ event}) = \aleph\left(\bigcup_{i \in C} [E_i \in C_i]\right) \geq 1 \quad , \quad \text{Equation 5: Multiple notifications received per unique event.}$$

when it should be exactly 1.

Furthermore, because JavaScript is an interpreted script, concatenated function calls are executed as a tree with depth-first search traversal and LIFO order; consequently, the last notification will be the first one echoed remotely which thoroughly misrepresents the order of local events.

The redundant notifications in Equation 5 are evaded in Equation 4, the implementation of which is discussed below and includes a fix to the depth-first search and LIFO problem.

(4) Aggregates and Real-Time Collaboration

The NotificationObject implements an algorithm that follows Equation 4, thus ensuring that any given component will only be notified once of a given event (detailed in section 5.7.2(5)). Event replication can come from forwarding or by firing new events in

response to the first. In either case, echoing the action on the workspaces of the other collaborating users implies replicating the event on the component that started the process. Mahups, or any aggregate application involved in real-time collaboration, must therefore heed only the component that originated the process. Figure 5.14 shows the classes involved in ensuring this requirement and their functions are explained below.

When an aggregate is made collaborative (and when an application is added to a mashup), a tree traversal of its sub-components produces full notification. The NotificationObject uses EventVisitation objects which contains a time stamp

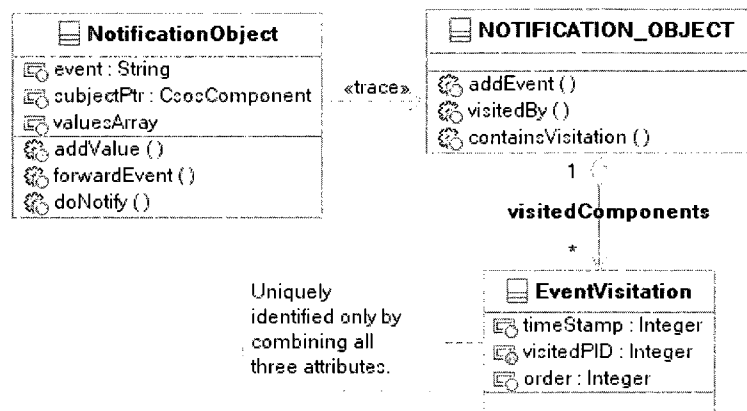


Figure 5.14: Event Notification Classes

such that forwarded events are not considered distinct. The EventVisitation object is paired with a static record (the NOTIFICATION_OBJECT) that tallies the components visited by events; these visited components are associated with a counter which identifies the order in which they were visited. Thus, the root component (the collaborating application or the mashup) receives all notifications, including redundant ones, but reacts only to the component marked as the originator (the first to have fired it).

In the case of new events fired in response to previous ones, the time stamp is ineffectual as the new event will have a different stamp; the new event must still not be sent to the other collaborating users as it will be automatically generated in response to the first as it was done initially. This is solved with a coding standard: in functions described by the contract, the programmer must provide a place holder for an extra argument that is not normally called; it consists of an EventVisitation object which is appended automatically by both RPC and locally synchronized copy mechanisms. Within the function, this argument is tested for existence: if it exists, it is forwarded to the Notification object

which keeps the time stamp from the extra argument and will identify itself with the original event and thus conform to the requirements.

(5) *Redundant Notification: Endless Loops*

When a locally synchronized copy of an application is created, a 2-way Observer Pattern is set up so each copy can replicate the actions of the other upon notification. If the synchronization mechanisms forward the event to the originator, an endless loop is created. Similarly, events fired with browser-to-browser RPC must not be returned to the collaboration manager, which would result in the Collaboration Manager returning the RPC call, thus creating a distributed infinite loop. Figure 3.13 Shows the redundancy risk when both LCS and real-time collaboration combined.

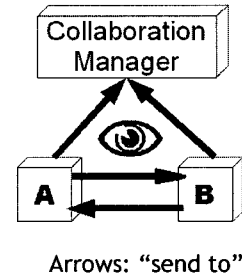


Figure 5.15 : Infinite notification loop.

The solution to this is ensured by using the NotificationObject discussed in section 5.4.1 which ensures that components that have already been visited by a given event (either firing it or handling it) will not be notified anew.

Figure 5.16 is a communication graph where two users (above and under the cloud shape) are in a collaboration session – the double arrows show the channels of communication between applications – and with the application also having locally synchronized copies (three copies total above, two copies total below). An action is performed by subcomponent 1ai which is cascaded down to its root, component 1.

The local synchronization notifies the initial component, the one that participates in collaboration. The synchronization algorithm uses the mechanisms for RPC using the contract object to reproduce the action on the corresponding sub-component 2ai which, as a result of reproducing the action, also cascades its notifications to its own root, component 2. The synchronization algorithm repeats this process by notifying component 3 (which repeats the RPC on its corresponding sub-component 3ai and notification cascading) and the collaboration mechanisms which replicate the entire process with

components 5, 6 and their respective sub-components. Throughout the entire process, each component visited by the process adds its identifier so that it is neither notified nor the corresponding functions called on it again, thus evading redundant replication.

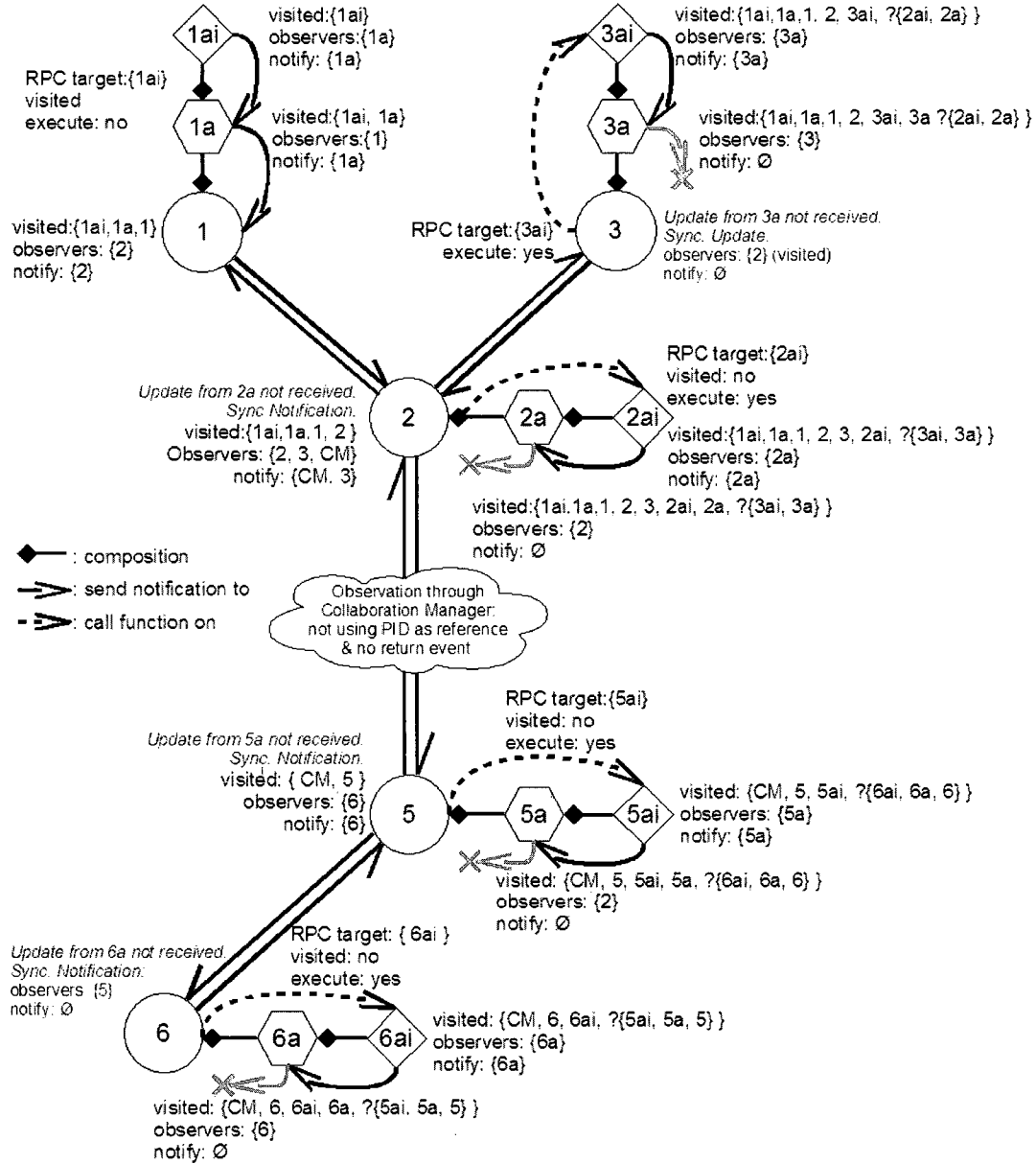


Figure 5.16: UC-IC's Algorithm for synchronizing both locally and remotely avoiding endless loops.

5.8 Browser-to-Browser ORB

As discussed in section 2.15.13(3), the state of the art normally implements ORB with a client that uses services located on a remote server but any description of ORB between client applications on any platform has yet to be found. The following use-cases were found where ORB can benefit distributed applications:

1. The case where computation on the local side depends directly on the state of user-activity on the remote side (complex Proxy Patterns).
2. Using browser-to-browser ORB is used to bypass role based access control restrictions.

Having designed and implemented browser-to-browser RPC as well as action replication for real-time collaboration, the remote delegation of behaviour in ORB is thus provided.

The second point is significant. The deferred synchronicity in AJAX applications (see section 5.6) means that a routine cannot call member functions from the remote object where the return from that function has an impact on the routine. This thesis will assume that such dependency is absent, leaving its solution to future work. Instead, the goal of this thesis is to demonstrate the feasibility of the automated creation of these proxy objects. Conversely, where member variables are kept up-to-date with the remote observer pattern, then they can be used within a routine as long as it is tolerant of the lag incurred by remote observer pattern.

What remains is the automated creation of proxy objects²⁶ which exhibit two properties:

1. The programmer is capable of calling functions on them as though they were local.
2. The actual execution of the function is performed on the remote.

There are two ways to do this:

26 There also remains the issue of discovery (e. g. UDDI) which belongs to the scope of the thesis on the server. As with real-time collaboration, this thesis will use application spawning for implicit discovery.

1. JavaScript supports looping through a closure and describe the functions and variables it contains. Because this looping is done on an instantiated object, each description will be specific and correctly reflect the metamorphic state of its definition, structure and implementation.
2. The contract object can be used to build a new object with the properties it describes. The routine can simply use the existing marshalling from browser-to-browser RPC to delegate the function calls to the (real) remote object.

The looping mechanism (point 1) gives no information whatsoever on arguments required by any given function; consequently browser-to-browser marshalling is unsupported. It also describes everything in the object without restriction. This means that access control is more difficult to implement. The work underlying this thesis therefore chose the contract method. More details are given in Appendix K and results are discussed in section 6.9.

5.9 Sets of Inter-Dependent Applications

The first prototype of interactive RIA which led to this thesis raised the notion of inter-dependent applications: sets of applications where changes in the state of one has implications on the state of at least one other (an example given in section 6.4). To address the deficiencies noted in section 2.15.12, the following features are required:

1. Independent containment of each application in the set while maintaining association with the set.
2. Support for multiple instances of any given application associated with the set.
3. Support for multiple independent instances of the entire set.
4. The set must ensure the presence of applications where actions on another application depends on them.
5. Means to save and restore the state of the set which must include:
 - (a) Applications launched (or not) in each set.
 - (b) Associations between instantiated applications in the set.

- (c) The full state of each application in the set.

Concurrent activity on multiple instances of a given application set poses particular problems. For example, the use-case “Send to User” indirectly implies support for multiple sets as the receiving user might already be working on the same application set: the two sets must be able to co-exist without interfering with each other. This can be modelled as follows.

Let S be a set of inter-dependent applications, A be an application and E be an event. Independence of application sets is defined as:

$$[E_i \in A_p \in S_u \Rightarrow E_j \in A_q \in S_u] \Rightarrow \neg[E_j \in A_q \in S_v]$$

Equation 6: Independence in Concurrent Application Sets.

with $\{i, j, p, q, u, v\} \in \mathbb{N}, i \neq j, p \neq q \text{ and } u \neq v$.

This is accomplished by building an invisible component called the SetManager which ensures communication between the correct instances of the concerned applications (Figure 5.17 a)) and registers required applications for actions performed by any other application in the set. Because some required applications may not have been instantiated at the time of a given action (e. g. a fired event or a function call), it ensures smooth behaviour of the set by filtering actions that require other applications: when absent, required applications are automatically started thus allowing the action to successfully complete its course (Figure 5.17 b)).

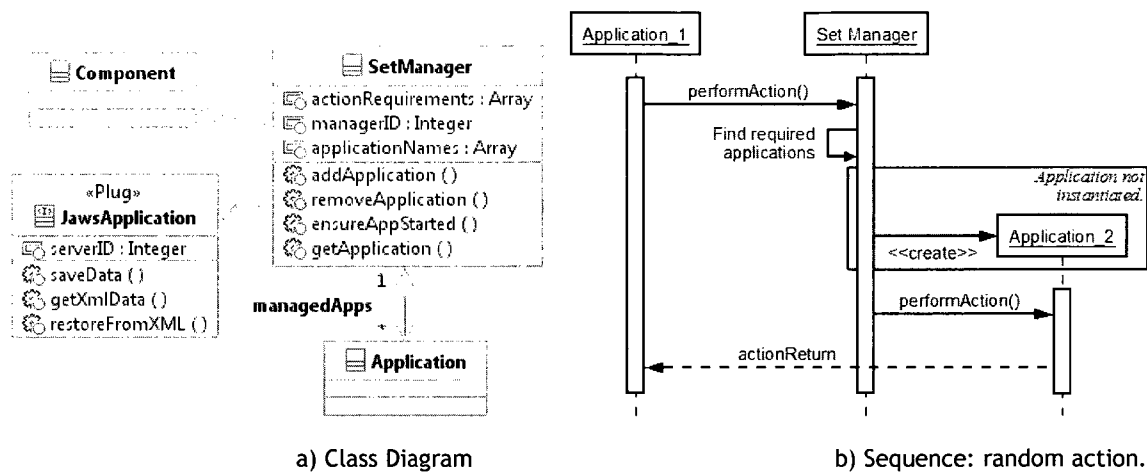


Figure 5.17: Sets of Inter-Dependent Applications.

The SetManager plugs the JawsApplication Plug, which allows it to be registered with the server for server management including persistence and restoration of state and replication to other users' environments (the "Send to User" use-case). Being ultimately a derivative of the Component class, it supports persistence and restoration with components that are added or removed dynamically (see section 5.10.2).

The `getApplication` function (SetManager class in Figure 5.17 a)) enables applications to obtain a pointer on a specific application; when the required application is absent, the SetManager instantiates it.

5.10 Dynamic Properties

The NIWT and CSOS environment is capable of acquiring resources at runtime to improve the user experience. This and the metamorphic quality of components describe the most important dynamic properties of the client-side.

5.10.1 Dynamic Resource Loading and Application Definition

Because the user environment is entirely contained in a web browser, all of its functionality must be loaded and rendered anew at every login. Where the state of the environment includes many running applications, the time and sequence of DHTML

rendering can exhibit some unexpected and unpleasant behaviour. A means to speed-up the resource-loading process therefore offers clear advantages.

In RIA, an application can be described as a combination set of JavaScript and CSS files. JavaScript objects build the necessary HTML and, in so doing, obtain all required embedded objects such as images, movies, embedded players and so on. An `ApplicationDefinition` is therefore defined along these lines to manage required JavaScript and CSS files. To do so, it uses the CSOS²⁷ which has means to know which JavaScript and CSS libraries have been loaded. Together, they allow the server application to determine which libraries are needed and limit the client resource loading to only those required. Afterwards, the client-side is responsible for ensuring the availability (loading) of required resources whenever an application is launched. Figure 5.18 provides a finite state machine for the process.

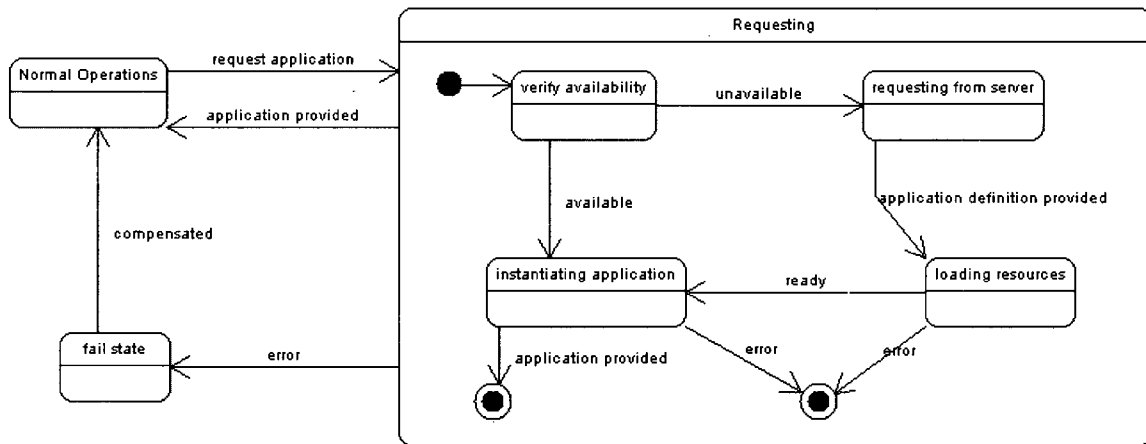


Figure 5.18: Algorithm to load application definitions on demand only.

5.10.2 Persistence & Metamorphism: Dynamic Data

Recall that the component architecture proposed by this thesis supports runtime changes in structure (number and type of sub-components) and interface (the superclass container can change and plugs can be added or removed). In order to restore such a component to

²⁷ Client-Side Operational Structure described in section 3.3.

its previous state after logging out and logging in anew, the data structure must reflect this volatility by defining itself dynamically.

AJAX features simplify the creation of a dynamic data structure. It supports creating XML documents from character strings and vice-versa. These strings can be the result of concatenations resulting from as many function calls as required. Figure 5.19 gives a block-diagram of the data of an Application contained in a Window and comprising utilities and other sub-components. Segments are reserved for:

1. standard data relevant to all components;
2. data recursively required by superclasses;
3. extra segments for each complexity of GUI component;
4. application XML;
5. the presence of atypical Plugs and the absence of standard Plugs.

Application XML refers to the Application object which, among other things, manages local synchronized copies which are not necessarily contained in the same type of GUI component. During data definition, the base-class (the derivation used to instantiate the object) recursively acquires and concatenates the relevant data model from each level of inheritance. The component also iteratively concatenates similar data from all of its sub-components, including the same recursion in each of them.

The restoration algorithm given in Figure 5.20 possibly makes the process and the reasons thereof easier to grasp. The data is parsed, handed-up the inheritance tree and delegated to the sub-components as needed in the reverse order of the creation process.

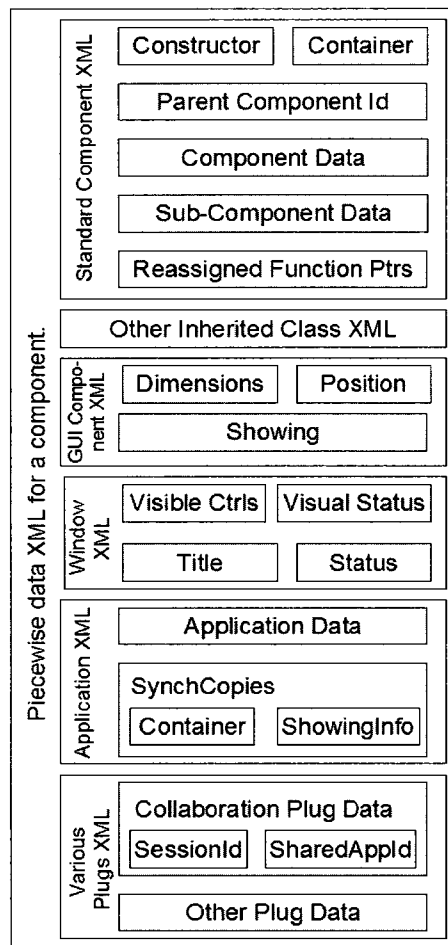


Figure 5.19: Dynamic data defined by inheritance and composition.

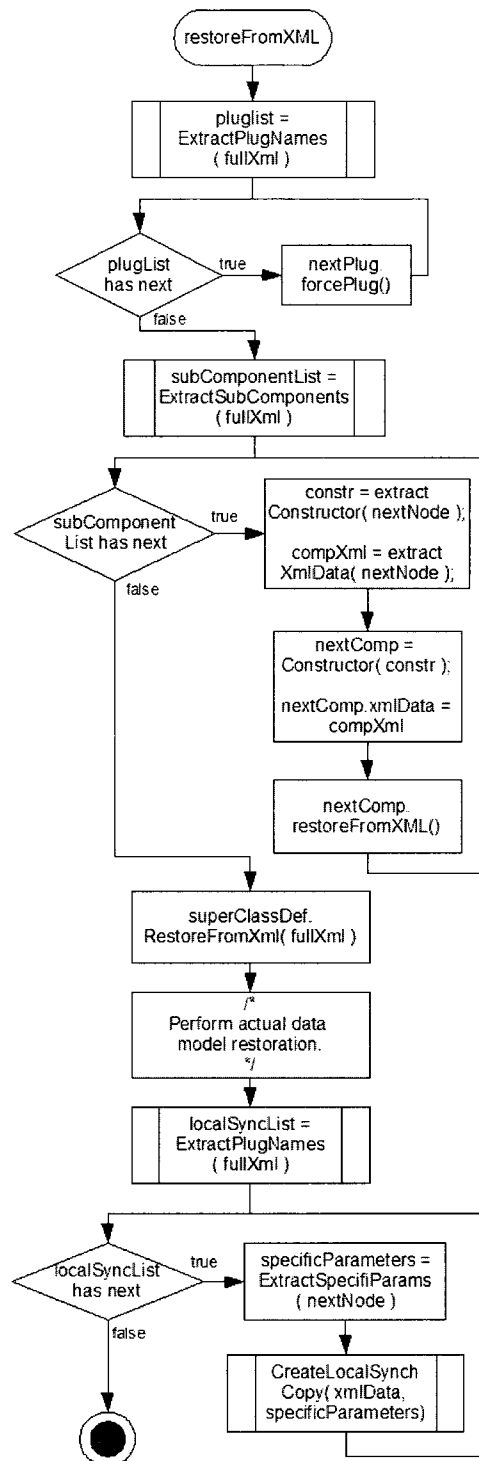


Figure 5.20: Dynamic Reconstruction of Metamorphic Components

Locally synchronized copies are not directly maintained on the server: they are built from the initial component. Therefore, they are also be obtained and concatenated in the data structure in a similar manner to sub-components. A list of copies with specific parameters for each (e. g. what was its parent component, what was its container type, size and position values etc.) is preserved for correct restoration. Data *per se* is not included in the locally synchronized copies algorithm as the same data is used for all copies.

5.11 Scalability to Multiple-Technologies

As was mentioned in a few places before, this thesis considers ‘scalability’ as the ability of increasing the number of applications and increasing the number of compatible technologies. Here are the most important effects on the architecture for the desktop environment. The section on inter-technology DnD is especially important as it includes basis notions (API) for any other kind of technology interoperability. Where other resources are included via IFRAME objects, the potential exists for other technologies than DHTML to be used. The injection algorithm is therefore a first gateway to communicate which requires the API described in DnD to complete the pipe.

5.11.1 Interoperability Between Technologies and DnD

DHTML is capable of calling functions in several technologies that are embedded in the web-document and vice-versa²⁸, most notably Adobe Flash and Flex (ActionScript) and Java Applets, hereafter called “embedded object”. The MVC design in section 4.2 uses JavaScript objects for the control structure and data models thus building a ‘wrapper’ component for the embedded object and generate HTML to contain and establish the active relationship with it; this effectively creates a relay between the embedded object and the user-environment. Of particular interest, this wrapper allows DnD to operate between applications built with different technologies or between different applications

28 Both Java Applets and Adobe ActionScript based applications provide an API for two-way communication with JS. Other technologies such as Windows Media Player provide an API to their functions.

of the same technology where visibility is not normally supported²⁹.

Mouse events over embedded objects are not automatically propagated to the DHTML environment, so DnD inclusive of embedded components implies that each component will handle fragments of the finite state machine instead of the user environment (Figure 4.15). This results in a model that is always incomplete from the perspective of the environment as well as from the perspective of any given embedded object. Recall from section 4.5 and Figure 3.15 that the user-environment is responsible for managing the DnD finite state machine. By providing means for the different technologies to communicate between each other through the user-environment, changes in the DnD state can be signalled between the environment and an embedded object. Figure 5.21 illustrates a web-page (large surrounding area) that contains an embedded object (central area): in order to ensure the API for mouse-event communication, the embedded object is surrounded with a DHMTL wrapper (intermediate area) that captures and relays mouse events in the web page and relay them via the JavaScript controller.

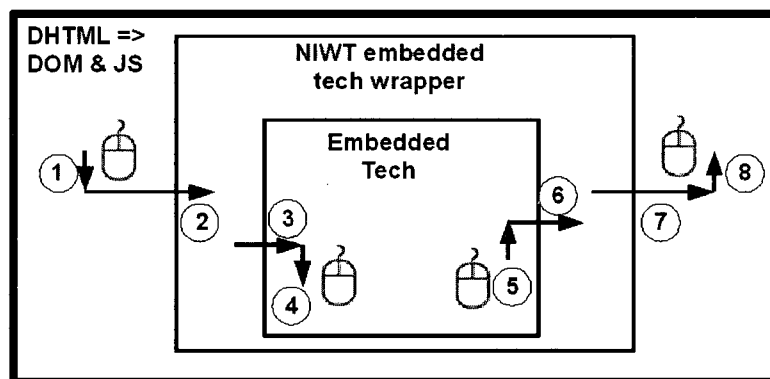


Figure 5.21 : Event model for DnD with embedded technologies.

Point (1) shows a drag being started in the main page. In (2) the Enter Target event is captured in order to inform the embedded object that DnD is in progress. In (3) a link is established between embedded object and acceptability validation is performed by the latter. In (4), the Drop Here event is captured and handled by the Flash movie which

29 For example: numerous Java Applets can co-exist in a web-document, but they have no awareness of each other; they are unable to communicate with each other without special support.

relays the state change (now Off) to the user-environment through the JavaScript object that controls the DHTML wrapper.

Points (5) to (8) show the same process in reverse. In this case, (5) must inform the main page that a DnD scenario is either started or was already under way. (6) will signal the Exit Target event to the HTML space and set the mouse-cursor according to the DnD state. In the case where a DnD has been started on the embedded object, (6) ensures that the DnD state “dragging” is set and the drag-source object has a pointer or delegate. Then, (7) and (8) can complete the finite state machine.

Some embedded technologies (e. g. PDF viewers) do not communicate with JavaScript. In such cases, mouse events resulting in changes in the DnD state are not handled and therefore not conveyed to the DnD controller. Assumptions can be made on the state of the mouse as it fires events on the DHTML wrapper and decisions taken from those assumptions. Figure 5.22 shows such a use case where a cover cannot be used. In point (1), the mouse enters the embedded wrapper with the mouse pressed and the DnD is “Dragging”. When entering the wrapper, the mouse enter event will instruct the JavaScript application that manages the wrapper to advise the embedded application of the DnD state. No events were fired when the mouse entered the embedded wrapper from the embedded application, but an event is fired on exit³⁰. If no signal has yet been sent by the embedded application, the wrapper is still capable of determining from the mouse-button state if a DnD action is still under way or not and react accordingly.

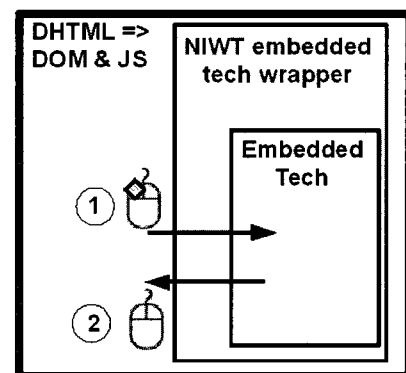


Figure 5.22 : The state of mouse buttons allows assumptions on DnD.

30 More precisely, when the mouse re-enters the wrapper element.

5.11.2 IFRAME injection

Many RIA are made available for reuse in the form of links to web pages, servlets and other such web-document generators. One common way to integrate these RIA is to enclose them in an IFRAME element so they can be manipulated as independent entities within the user environment [108][109]. A serious corollary is that any resource loaded in an IFRAME is actually an embedded page treated by the browser as a separate web-page: there is no implicit communication between the main page and the embedded page (Figure 5.23 a) and therefore interoperability is greatly hampered. This means the environment containing the RIA is unable to manage the imported application in any way.

It would be preferable if the embedded page were able to call functions from the main-page environment to notify observers of events (JavaScript or HTML; 'onload', 'onunload', etc.). Simple experiments have found a way to inject code into the embedded page and have it call functions from the main page and vice-versa (Figure 5.23 b).

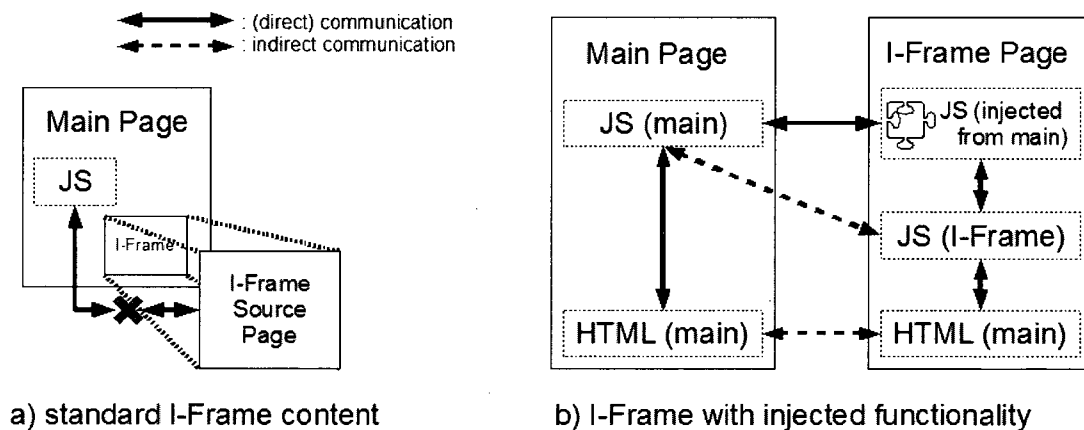


Figure 5.23: IFrames are external pages merely embedded in the main page.

Definitions:

1. Let **parent** be the page that contains the IFRAME or EMBED³¹ element.
2. Let **embedded** be the page contained within the IFRAME or EMBED element.

Though direct JavaScript communication between the parent and embedded page is not supported *a priori*, certain properties have allowed the research prototype project to establish a 2-way link between them and inject the necessary code into the embedded page thus ensuring communication (Figure 5.24). The injected code consists of the necessary pointers and communication mechanisms for event handling and for calling functions both ways. It is also possible to modify the HTML of the embedded page (e. g. to insert new sub-components).

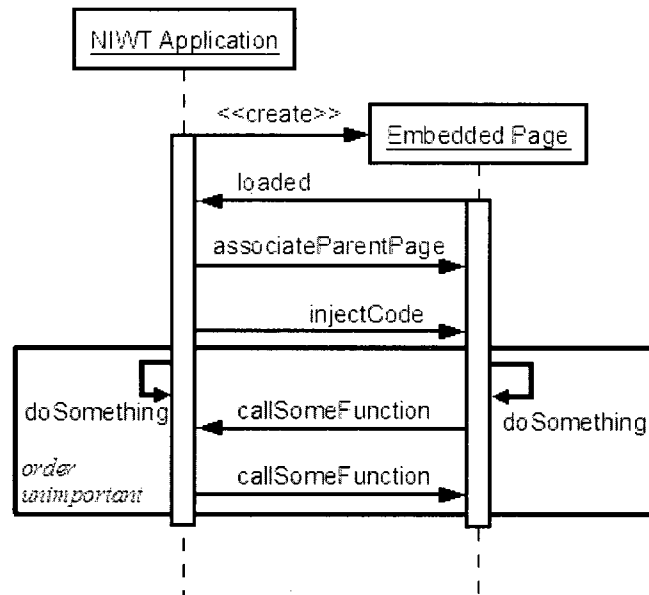


Figure 5.24: Injecting code for interoperability with IFRAME RIA.

Of course, integrating third-party resources remains a non-trivial matter. A common API between the parent page (the user environment) and the embedded page (the imported application) is still required for interoperability and collaboration to take place.

31 The HTML element used for embedding other technologies, usually plugins.

Chapter 6

Results: Validation of the Design

The results of the research work and experimentation presented in this thesis consist of the user-environment itself. Its architecture (Chapter 3) is the application of the programming paradigm (Chapter 4) which supports the application of design patterns and mechanisms (Chapter 5) to build the user-interface that will support the use-cases. This chapter therefore presents selected details about NIWT and CSOS which validate the design and theory presented in the as supporting the protocols and use-cases discussed in the preceding chapters.

This chapter will begin with a review of the deficiencies described in section 2.15, Summary of Deficiencies in Previous Approaches, with a short description on how this thesis work has solved them. It will then follow with a validation of the concepts, algorithms and architecture developed in this thesis with concrete examples from the various iterations of the client side and other experiments.

6.1 Answer to the Deficiencies in the Literature Review

Here follows a summary of how this thesis work answers almost all the deficiencies highlighted in the literature review.

Deficiency 2.15.1 Appearance and Disappearance of Related Projects

The availability (or lack thereof) of libraries and platforms that addresses a business or commercial issue that cannot be resolved by this study on technology. It can be considered, however, that in addressing the deficiencies found by the literature review, this thesis work stands a better chance at adoption and longevity.

Deficiency 2.15.2 Sharing and Real-Time Collaboration Issues

Current research deals mostly, if not exclusively, with protocols and algorithms to resolve

concurrent edition conflicts on data-centric collaboration, leaving the general case of collaboration on random applications unaddressed. The protocol and architecture for event replication (or echoing actions) presented in this thesis does the exact opposite. With event replication, any application can now participate in real-time collaboration regardless of what it actually does or how it structures its data. Be it an editor or control device, any application can now benefit from real-time collaboration. Furthermore, whereas previous collaboration was devised for specific applications which do not change, now change in the composition and functionality of an application is built-in to the collaboration process itself.

Deficiency 2.15.2.1 Current Sharing is User Activity Intensive

Whereas the current state-of-the-art makes sharing a passive and static activity by only making resources available and sending notifications of the availability, this thesis work has made sharing fully dynamic and interactive. By directly sending applications to another user (e. g. by dragging an application onto a user avatar), users need no more perform several steps to access a resource and sharing becomes a living interaction between people. With dynamic resource loading in the case where the recipient does not possess the required application definition, sharing increases productivity like never before. Benefiting from miniatures and lists on the user avatars, discovery of available files, applications and the online status of other users, awareness also makes sharing a more interactive process between people.

Deficiency 2.15.2.2 Real-Time Through Polling is Inefficient

The drawbacks of polling include time-out problems which can cause a browser to completely destroy the web-document serving as the user-environment. By using AJAX-Push, the client-side is now capable of receiving signals from the server asynchronously.

Deficiency 2.15.3 One-to-Many = Incomplete Collaboration

This thesis proposed an architecture based on composition and abstraction where communication happens bilaterally between any number and arrangement of

components; rather than providing a single callback function for a given event, any number of components add themselves as observers of any number of other components. This allows a many-to-many-on-many (M3) distribution, where components can respond to the same event originating from any number of sources and also notify any number of sources of their own events.

Deficiency 2.15.4 RIA: a Backwards User Interface Paradigm

The desktop metaphor has been restored as a versatile user-environment. By providing a component oriented design which supports and favours new development and the integration of third-party resources, RIA now benefit from all the advantages of a Linux-like multiple desktop metaphor which has also been adopted by MacOSX. The ability to simultaneously contain multiple applications from any number of domains furthers the support for M3 distribution.

Deficiency 2.15.5 Reuse is Absent from the Design

Reuse is now an integral part of the windowing toolkit and application development itself. Visual containers and interactive components are now the basis of inheritance for augmentation or modification by the specific needs of the new application. Replacement of sub-components is encouraged rather than a difficulty. The full object-orientation of components with support for introspection facilitates filtered reactions to events from components on the basis of their interfaces or inheritance properties.

Deficiency 2.15.6 No Iterative or Hierarchical GUI Building

NIWT components benefit from reuse through inheritance and the iterative building of its HTML code by concatenating calls to superclass process that build it. Thus the visual features of a given class can be reused and augmented with only what is needed rather than requiring re-coding with every new application.

Deficiency 2.15.7 Unidirectional Method Invocability

The principle problem with callback coding is that only one object (or the global scope)

can react to a an event fired by a given object. As explained for Deficiency 2.15.3, One-to-Many = Incomplete Collaboration, the observer pattern is one means by which any number of components can react to stimulus from another. But above this, the correct object-orientation definition of NIWT components with MVC re-designed to suit the pattern, components now have methods (member functions) that can be called randomly by any other component or routine whether it had provided a call-back function, registered as an observer or not. Finally, the global indexing scheme makes possible the discovery of any instance of any type of component for action on a custom set of components.

Deficiency 2.15.8 No Technology Independent API

NIWT provides wrapper components which support DnD, the use of automated mechanisms for real-time collaboration as well as application management from both the client-side (CSOS) and server-side concerns. Through the wrappers, these features are afforded to applications programmed in DHTML, Flash, Applets or any other technology supported by web-browsers and capable of communicating with the web document.

Deficiency 2.15.9 Absent Combination of Supported Technologies

The CSOS/NIWT environment already combines AJAX (DHTML), Applets and Flash in its own basic components. It also makes use of JSPs in iFrames for file uploading and third-party libraries for the menu, calendar, map applications to name only a few. Add to this the features by answering deficiency Deficiency 2.15.8, No Technology Independent API for a highly inclusive platform.

Deficiency 2.15.10 Not True RT Collaboration, Not on Any Application

As explained in Deficiency 2.15.2.2, Real-Time Through Polling is Inefficient, the use of the observer pattern and RPC allows the replication of any action to occur asynchronously on any application of any purpose. True real-time activity depends on real-time constraints which are left to network service vendors. But the event-based nature of the communication better approximates real-time activity then periodic polling,

and with less unnecessary network traffic.

Deficiency 2.15.11 Mashups don't Collaborate

As per the third model of mashups described in [18] dynamic aggregations of applications benefit from both the transparent mechanisms and algorithms for collaboration. The responsive aggregation algorithm from section 5.7.2 ensures that new applications added to a mashup are also added to the remote counterparts in the collaboration session.

Deficiency 2.15.12 Inter-Dependent Applications are Full-Screen Singletons

Containment of applications in a NIWT window (or similar) allows for the management of multiple instances of applications as independent entities. Groups of applications are now managed as independent instances in a manner similar to mashups through the architecture for sets of inter-dependent applications as if they were single applications, thus allowing multiple and independent instances of these groups.

Deficiency 2.15.13 Programming Language and Software Design Issues

The new paradigm for web programming introduced by this thesis addresses the following deficiencies.

Deficiency 2.15.13.1 Misconception Leads to Performance Degradation

This deficiency described issues that are not cause for great concern. Nonetheless, current AJAX libraries approximate Threads through infinite loops of function calling. This has proven to slow down execution of any given call as the number gets very large. In NIWT and CSOS, activity is based on the reaction to events thus causing only the requisite number of calls to be made. Also, NIWT and CSOS components need not make the false claims of private members and other such claims described in this deficiency.

Deficiency 2.15.13.2 Multiple Inheritance

Most scholars will agree that multiple inheritance is actually a step backwards compared

to Java. However, in web-development and for the purposes of metamorphic objects, NIWT Plugs offer the benefits of run-time adding and removing of functionality directly on the object, rather than requiring delegation to unknown member variables that may or may not be present. Furthermore, Plugs can be added and removed at runtime which increases the versatility of which instances of a given component is capable of doing what as well as which are admissible to given routines.

Deficiency 2.15.13.3 RPC and ORB do not Connect Clients

The server-relay architecture presented in this thesis allows RPC and ORB to reach other clients. This has greatly benefited the creation of a layer for collaboration which in turn facilitates automation and transparency for developers.

Deficiency 2.15.14 Conceptual Problems with Popular AJAX Libraries

The NIWT and CSOS libraries were designed in direct response to these deficiencies as explained below.

Deficiency 2.15.14.1 Server Generated Client-Side and Locality of Reference

With the new architectural implementation of the MVC pattern, the server is only responsible for providing the client with resources: the client-side is then fully autonomous in generating its display (view) HTML and managing its own concerns – thus, locality of reference is ensured. Where server determination is still required for security and/or management reasons, the old paradigm is still supported through AJAX.

Deficiency 2.15.14.2 Network Dependence

The issue concerned continued user activity during network failure. By virtue of locality of reference, NIWT and CSOS are fully client contained with all required application definitions given loaded by the browser. Where network failure occurs or if the JAWS server is inaccessible, the user can continue to use applications until the connection is restored. This does not address use-cases of broken connections to specific servers are required by design. In order to ensure no user activity is lost to server concerns, a

protocol of recovery from failure keeps a log of all client-side activities and the state of application data for the purpose of both restoring the client side (done) and synchronizing the server-side (yet to be designed).

Deficiency 2.15.14.3 Inadequate Object Orientation and Improper MVC

As mentioned above, NIWT and CSOS benefit from the declarative properties of JavaScript and inheritance through the Factory Pattern to provide a full introspective properties. Multiple, independent instances of the same application can now co-exist and communicate with any other instance of any other component while any entity, local or remote, client or server, can acquire information or call functions on them.

In terms of MVC, even JSP technology fails in this respect as the client-side HTML is generated by the server side whose purpose should be management and communication. With the architecture proposed in this thesis, only data is communicated between server and client to be interpreted – and sanitized – by the receiver.

Deficiency 2.15.14.4 CSS DnD and Server Communications

Contrary to the current state-of-the-art, class identification for DnD, server communications and any other purpose is no longer a province of the HTML or CSS (view) but consists of member variables and mechanisms of the JavaScript controller object as provided by inheritance from CSOS and NIWT components.

Deficiency 2.15.14.5 Memory Leaks

This issue is not yet resolved as it depends on garbage collection properties of the browser's ECMA engine.

Deficiency 2.15.15 Multi-Touch Support ≠ Multiple Mice

The NIWT pointer object can be controlled by the local Operating System's, real-time collaboration process or any programmed routine. These are independent components that replicate the actions of a mouse on NIWT components. An API is provided for

normal mouse functions which can be overridden for specific purposes. By instantiating multiple copies, a user environment is afforded true multiple pointing devices which can be controlled locally or remotely.

6.2 *Features of NIWT and CSOS*

Many components and features have already been described in this thesis to illustrate the concepts proposed. This section will elaborate on the more noteworthy features of NIWT and CSOS thus validating the thesis.

6.2.1 *Three Kinds of NIWT*

Having built the server and client sides concurrently, a reusable API was established with implicit recommendations on database design such that the NIWT/CSOS client can be reused with other existing server applications. The project requirements for the client-side included that strategic parts of NIWT be usable in a random HTML page; in other words, NIWT components must be capable of fulfilling their tasks independently of the larger-scale environment. This resulted in three supported deployments of NIWT:

1. The full-featured JAWS managed version offers all the benefits of nomadic computing with dynamic resource loading, real-time collaboration and restoration.
2. A ‘no-server’ version of the entire environment can be deployed with links to required resources pre-defined and static.
3. Many select sets of components can be extracted for specific use.

Though version 2 does not support nomadic computing, it can still serve as a presentation layer where contents are specially designed for the purpose. This version was the one most often used for the development of new client-side components and full applications as well as the testing of new algorithms and routines. Otherwise, this version can also be used for other server technologies that implement the required API.

The usefulness of version 3, is apparent in the plethora of so-called ‘Widgets’ (more

apply termed web-based utility applications), only a small portion of which was discussed in Chapter 2 .

6.2.2 Containers, Layouts and Mashup

The NIWT Container component provides content layout management in a manner similar to Java.awt and Javax.swing. Beyond the capabilities offered by Linux and Solaris, the multi-desktop metaphor environment is capable of several layouts. Users can see all applications on all desktops simultaneously in a tiled layout, or navigate them in a cascading motion (Figure 6.1).

The slide-show and accordion layouts was tested along with container independence (or volatile inheritance, section 4.1.3(3)) with the Accordion Mashup application which consists of a blank window serving as a Drop Target for other applications. Typical tests consist of dropping a Calculator, a Sticky Note, an Image Previewer and a Word Processor whereby they are re-associated as sub-components of the Accordion and change their container, exchanging their Window for an Accordion Pane.

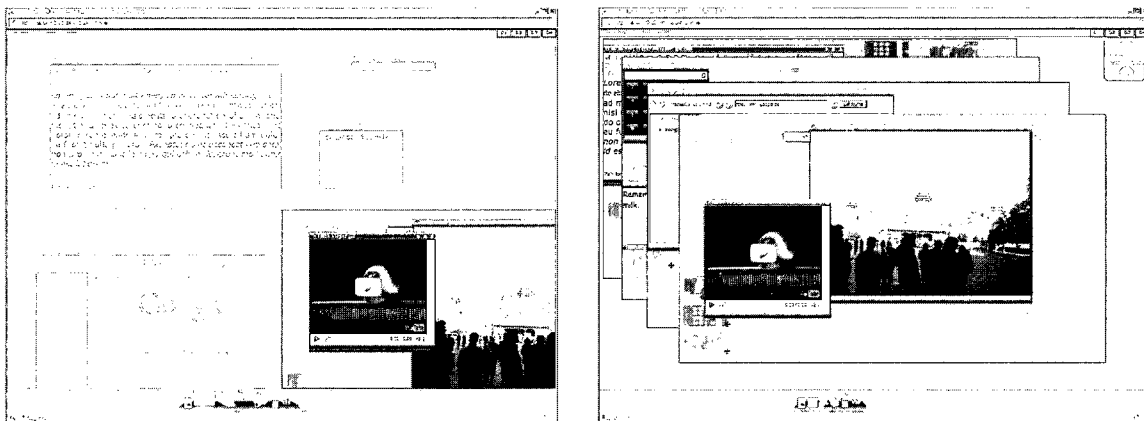


Figure 6.1: Workspace layouts: tiled and cascading desktops.

Though layouts similar to Figure 6.1 can be obtained by tabs in browsers or by the most recent versions of browsers, the dynamic adding and revocation of desktops, the interoperability of applications between them and the application of these layouts to any

component whatsoever is not provided by the browser.

6.2.3 Utilities

Utilities are small application components that are not intended to be used alone, but rather inserted into another application. Corollaries include:

1. Visually, they must be inserted in strategic parts of the main application.
2. Their data must integrate the data scheme of the main application.
3. They must do the above with minimal effort by application developers and, preferably, support runtime addition/removal by users.
4. No programmatic effort beyond adding them is required to ensure support for collaborative along with the main application.

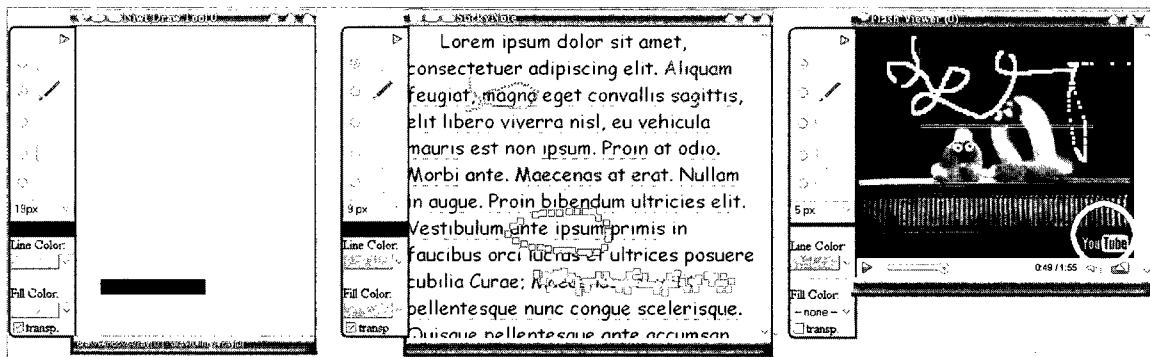


Figure 6.2: Three uses of the Draw Utility

The above have been successfully designed and tested with the NIWT Draw Utility (Figure 6.2) which was inserted in an empty window container to build a stand-alone draw-tool, the Sticky Note application for graphical annotations and a simple YouTube Player application for graphical annotations.

All corollaries above are supported. The programmer need only add 2 lines of code in the application's constructor and one line of code in the post-insertion setup. Automatic collaboration is supported by automatic registration of the main application as observer when the utility is constructed. This does, however, require the main application to implement application-specific event forwarding and delegation.

6.2.4 SDMD & DnD

Among other applications, the SDMD pattern has been successfully applied to:

1. The JAWS File Browser contains a directory tree that was the first experiment in SDMD. It implements the PC file manager example described in section 5.4.2.
2. User Avatars: only one user profile and communication object, but many copies of the Avatar's GUI (one on each desktop). They each react to changes in the user's connection status and changes in shared file content.
3. Running applications menu on the desktop switcher.

6.2.5 Advanced DnD Use-Cases

The detailed requirements analysis for the UC-IC project stated maximizing the use of DnD to help users intuitively learn the system's potential. The client-side has implemented the following using DnD:

1. Sending an application to another desktop or user.
2. Adding to and removing from mashups.
3. Starting real-time collaboration sessions.

Sending applications directly to other desktops is supported by DnD in three ways:

1. When desktops are tiled (Figure 5.7, p. 109), simply drag an application directly onto another desktop.
2. Drag an application onto a desktop icon in the desktop switcher (Figure 6.3).
3. Cross-Desktop Mashup: Figure 6.3 shows these switchers displaying the running applications of another desktop. Dropping an application onto one of these application listings initiates DnD delegation (p. 93).

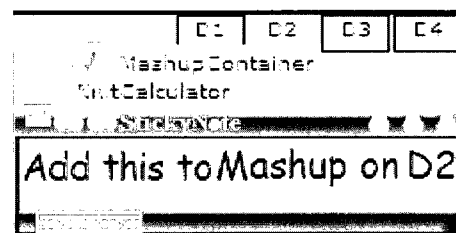


Figure 6.3 : Using DnD delegation for cross-desktop mashup.

Applications are sent directly to other users by dragging the application onto a User

Avatar, thus avoiding the multiple steps detailed in section 2.15.2. The opposite motion (dragging a User Avatar on an application) will initiate, a collaboration with default settings with the corresponding user.

6.2.6 Performance of Dynamic Resource Loading

A full implementation of the architecture and finite state machine described in section 5.10.1 requires corresponding work server-side which was not completed at this time. Consequently, a performance test could not be devised. Nonetheless, a test-case for the functionality was devised and successfully performed. The scenario is described as follows:

1. A launcher for an application is included in the user environment (e. g. a menu item) for some application, but the required resources are not loaded.
2. The user launches that unsupported application whereby:
 - (a) The Application Manager requests the definition for the application and inserts the request in the launch queue.
 - (b) The definition is obtained and the JavaScript and CSS resources are loaded.
 - (c) The request queue is examined for this application and, for each request in the queue, an instance is created.

Currently, all this is performed as a unit test-case programmed in a single JavaScript function. The contents of this JavaScript function execute each step of the algorithm exactly as would a client-server interaction (minus the AJAX call and assuming its successful response). The feasibility of the algorithm is thus proven.

6.2.7 Recovery From Failure

As with Dynamic Resource Loading, a full implementation of Recovery From Failure (see section 3.1.5) requires corresponding work on the server-side, which is not available at this time; the synchronization protocol is missing. The client-side does

however perform the persistence of the state of the user environment and changes during disconnection and also restores these upon reconnection. The test-case is described as follows:

1. Any AJAX call to the server results in a failure code which sends the client-side into failure mode.
2. From that point on, the entire user state is saved and changes are logged in client-side file saved on the local disk.
3. On reconnection, the user environment finds this file and restores itself to that state.

This was successfully tested by:

1. Running the no-server version of NIWT (see section 6.2.1).
2. Performing user activity that incurs communication with the JAWS server:
 - (a) New applications were started and some were terminated.
 - (b) Visual aspects were changed:
 - i. Some applications moved, others resized.
 - ii. Some windows rolled-up, others minimized.
 - (c) Data was edited.
3. The browser was closed.

Without the recovery protocol, starting the no-server version anew results in the default setup only as no dynamic information on the state of the user-environment is stored in cookies or by any other means. Re-starting it with Collaboration Manager successfully found the local change log file and restored the environment with all the running applications appearing as they did on closure including data and visual preferences.

The corollaries of supporting continued user activity during network failure are not trivial; they include, but are not limited to:

1. Saving data changes, as usual, will only occur if a user or some other explicitly calls the save function. The recovery mechanism is not a fail-safe against loss of user work.

2. Consider a number of real-time collaboration sessions are in progress during the failure and the connection is re-established while at least one of these sessions is still in progress. Both the SIP server and synchronization protocols must model a solution on how the different application states will be restored between the users, some of which might have experienced disconnection, some might not have.

6.3 Applications Developed

A number of end-user applications were successfully developed³² with the purpose of proving the following points:

1. Application development – as understood for end-user desktop applications – is feasible on a web-based platform.
2. A WebOS platform is capable of supporting user-applications from the most common (e. g. office productivity) to the most specialized (e. g. RT data for specific use case scenarios, control interfaces, etc.).
3. Easy development of new applications with unknown purposes, supported by existing visual and algorithmic components, can be supported by a WebOS.
4. Regardless of the platform (WebOS or PC desktop), a user environment can be built with support for collaboration built-in such that developing real-time collaboration applications requires little more effort than for a single-user application.

6.3.1 Applications in Different Technologies

The applications developed include, but are not limited to:

1. End-User Applications in different web-document compatible technologies:
 - (a) **DHTML:** Sticky Note, full-featured Word Processor, Calculator, To-Do List, Web-Browser, GIS map with RT data, JAWS File Browser, an instant messenger (where the application need not be launched for reception of messages) and a collection of debugging and system management tools.

³² Some applications were developed by other people. See acknowledgements on page 2.

- (b) **Java Applet:** Moon Phase Viewer.
 - (c) **Flash/Flex:** Multi-user video conference utility, You-Tube Player with DHTML time-stamped comments.
2. User-Environment Components in different web-document compatible technologies:
- (a) **DHTML:** User Avatars, Pointer, Fish-Eye Applications Toolbar, alert bubbles and a desktop switcher with an active menu of running applications.
 - (b) **Java Applet:** Recovery Manager, Gestural Controller of NIWT Pointers.

6.3.2 Applications and Component Oriented Programming

The support for dynamic addition and removal of sub-components is not limited to applications but includes the Workspace itself (with Desktop sub-components), Running Applications Viewers (with Application Launcher sub-components), the Animated Launcher Bar (with Application Launcher sub-components) and the desktop itself (with Window, Miniatures and User Avatar sub-components).

Applications implementing dynamic sub-composition include the Collaboration Manager (each Session is a sub-component) the Slide-Show, the Tree View and the Directory Box in the File Browser, the Application Manager GUI, and most notably the Multi-Book which consists of Accordion Book sub-components consisting in turn of Sticky Note sub-components³³ (Figure 6.4).

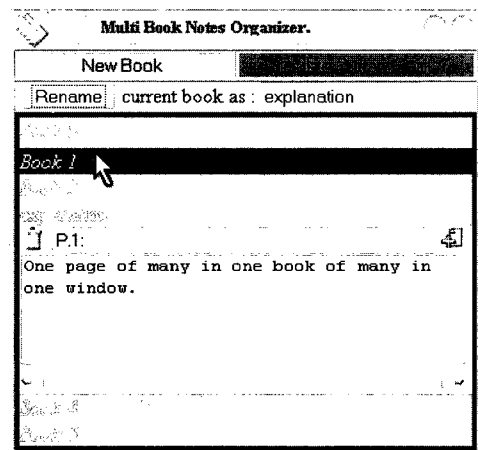


Figure 6.4: Multi-Book:
3-level sub-composition.

6.4 Sets of Inter-Dependent Applications

The Asynchronous Joint Intelligence Picture (AJIP) was the project that preceded UC-IC

³³ This is also an example of volatile inheritance as the Sticky Note applications in this case do not inherit from the Window class, but rather the basic GuiComponent class.

consisting of five static applications: a GIS Map, a Search tool, a Layers Selector, a Details viewer and a contextual Free-Search tool. AJIP is capable of displaying markers on the map whose position is updated in response to RT data acquired asynchronously ('pushed' to the client by the server). Being hard-coded, AJIP did not support multiple instantiation of any of its applications. Neither sharing nor real-time collaboration had yet been envisaged but AJIP did prove the concept of interoperability between AJAX applications and a thick client to manage RIA.

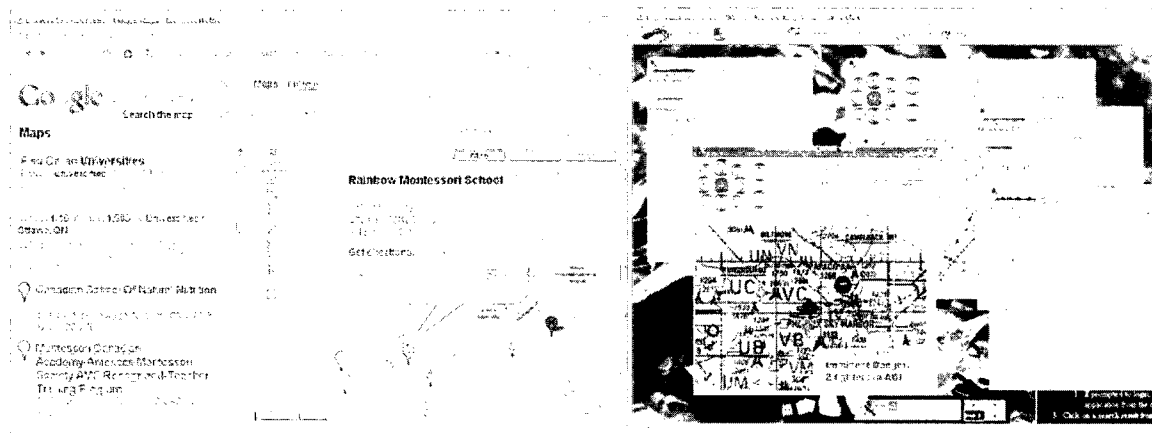


Figure 6.5: Google maps compared with two sets of inter-dependent applications (AJIP).

Generation 1 of UC-IC adapted the AJIP applications, defining them as a set of individual applications in the NIWT/CSOS user-environment and thus affording them with multiple instantiation and "Send-to User" type sharing. Association of applications with a set and the support for independent concurrent sets was successfully obtained by implementing the SetManager as described in section 5.9.

Figure 6.5 shows two concurrent and independent sets of AJIP (distinguished by the title bar). The two leftmost applications belong to the local set and the three rightmost applications are the result of receiving one application from another user: only the Search application was sent, the Map and the Layers selector were instantiated by the SetManager as a result of clicking on a search result.

6.5 Comparison with the Current State-of-the-Art

This thesis combines use-cases and technologies that other solutions propose separately. Comparison can therefore only be made on the basis of features provided by choice libraries and performance of WebOSes. Conclusions will show that the NIWT/CSOS prototype is at least as good as the state-of-the art while supporting much more.

6.5.1 Feature Comparison

The two following tables compare features of NIWT with the most important and popular libraries for the client-side development of RIAs. The features compared will address reusability, interoperability with other technologies and ease of use in terms of programming practice. The tables clearly show that NIWT, by implementing the theory detailed in this thesis, exceeds the overall potential of all other mainstream libraries.

	NIWT	GWT	ExtJS	Bindows
extendability (new components)	yes	no	no	yes
dynamic multiple, independent instances	yes	indirect	yes	yes
true MVC	yes	yes	yes	yes
standard JS, DOM and CSS	augment	abstract	abstract	abstract
multiple concurrent CSS	yes	no	no	no
state change notification	yes	no	no	no
component search and discovery	yes	no	no	no
all-way communication	yes	no	no	no
changeable containers	yes	no	no	no
dynamic contents	yes	yes	partial	yes
inheritable for augmentation	yes	no	yes	yes
Java-code generation	no	yes	no	no
abstract server layer	yes	yes	yes	yes
unsolicited server commands	yes	no	no	no
RPC between web clients	yes	no	no	no
ORB between web clients	yes	no	no	no
dynamic ressource augmentation and revocation	yes	no	no	no
Pre-defined animations	partial	yes	full	yes
data acquisition	yes	no	no	no
cross-browser compatibility	partial	yes	yes	yes
Java-styled Object Orientation	full	no	yes	no
learning curve for Java/C++ coders	direct	direct	direct	direct
existing widgets	few: windowing	moderate: forms, menus, layouts,	many: forms, data viewing, windowing	many: data, controls, monitors, windowing
current integration of other technologies	flash, Applets, PDF, You-Tube, Windows Media Player	no	no	SVG

Table 1: Feature comparison between NIWT and other libraries - part 1.

	NIWT	OpenRico	Dojo	YUI
extendability (new components) dynamic multiple, independent instances true MVC	yes	no	no	no
standard JS, DOM and CSS	augment	use	replace	replace
multiple concurrent CSS	yes	no	no	no
state change notification	yes	no	set callback	set callback
component search and discovery	yes	no	no	no
all-way communication	yes	no	no	no
changeable containers	yes	no	no	no
dynamic contents	yes	yes	difficult	difficult
inheritable for augmentation	yes	no	difficult	difficult
Java-code generation	no	no	no	yes
abstract server layer	yes	no	yes	yes
unsolicited server commands	yes	no	yes	no

Table 2: Feature Comparison between NIWT and other libraries - part 2.

6.5.2 Performance Comparison

The following metrics presented in Table 3 were gathered by running UC-IC on the following setup:

- Server:
 - 2 AMD Opteron processors at 2.2GHz each
 - 2 GB RAM
 - OS: Ubuntu 7.04 server edition
- Client Host:
 - Intel Core 2 Duo T5450 1.6 GHz
 - 2 GB RAM
 - OS: Windows Vista Home Ultimate
 - Browser: Firefox 3.0.7
- Network: Internet (Rogers ISP) advertising 7 Mbs

Apart from the server, all other WebOSes were also tested with the same client host and network (ISP).

Table 3 compares the user-experience in terms of waiting time for typical scenarios. Currently, though they deal with real-time collaboration, industrial vendors such as Citrix, IBM and Cisco do not provide purely web-based user environments but rather replicate a running PC desktop remotely to one or more users: comparison can therefore not be made with such industrial solutions. UC-IC will be compared with existing Web-Desktop distributions, though the projects generating some of the better ones (e. g. YouOS [74]) have terminated and have thus become inaccessible for inclusion in this comparison.

Platform	<i>UC-IC gen1 (AJAX)</i>	<i>UC-IC gen3 (AJAX)</i>	<i>Cloudo (AJAX)</i>	<i>G.ho.st (Flash & AJAX)</i>	<i>Desktop Two (Flash)</i>	<i>Glide (Flash)</i>
Time to load the environment (no apps running)	Fresh Login (clear cache) 11.5s Subsequent Login 4.0s	Fresh Login (clear cache) 35.0s Subsequent Login 7.0 s	20.0s	32s	23.5s	5.0s
Time to load environment (12 apps running)	Fresh Login 16.5s Subsequent Login 4.5s	Fresh Login 40.5s Subsequent Login 10.5s	Logoff not working: cannot save apps. for restoration.	36s	N/A: Closes applications at previous logoff.	N/A : Applications are loaded in new browser windows.
Time to exit environment (no apps running)	0.0s	0.0s	Logoff not working.	1.5s	0.5s	2.5s
Time to exit environment (12 apps running)	0.5s	0.0s	Logoff not working.	1.5s	6.0s	N/A because of apps in new windows.
Time to start a new application (first app)	1.0s (simple word processor)	0.5s (Slide-Show with 5 pictures)	4.0s (calendar) 2.0s (plain text pad)	Calculator 2.0s	0.5s, 20s, 120s ³⁴	8.0s (opens a new browser window)
Time to start a new application (12 apps running)	2.5 s (simple word processor)	2.0s (Slide-Show with 5 pictures)	3.5s ³⁵	Calculator 2.0s	<i>same as first</i>	N/A because of apps in new windows.

Table 3: Web Desktop Performance Comparison.

Two ‘generations’ of UC-IC (project iterations with significant architectural changes) have been compared. The difference in performance (improvements and declines) can be explained by the vastly different complexity (content and architecture) as well as by the protocols used. This is discussed in 7.3.1(1)(a)(i)1.a.Appendix L but it can be stated here that generation 1 only loads one desktop at a time³⁶ whereas generation 3 loads the entire

34 No average and no trend could be established: fully dependent on type of application. Some are preloaded Flash, others required loading, others open a new window.

35 Maximum eight applications available. All applications are with tabs available for new files. For example, there is only one file browser: when a user wishes to access the trash can, the file browser navigates to that folder.

36 In generation 1, each desktop is a different web-document. Navigating to a new desktop loads it with its

workspace (with four desktops, in this test) immediately. Each desktop is loaded with its content of running applications including the restoration of its data state.

It can be concluded from Table 3 that, whereas the performance of UC-IC is stable and predictable, current Web-Desktops are often not dependable and the performance unpredictable; a remarkable exception being G.ho.st which is built in Flash. Nonetheless, no conclusion can be drawn between UC-IC and G.ho.st as their stated purposes are very different: G.ho.st aims to provide a virtual computer to individual users with emphasis on social networking whereas UC-IC is built for professional and large-scale use with emphasis on automating real-time collaboration and scaling to sophisticated distributed computing applications. Furthermore, the development of new applications development in G.ho.st's is barely featured whereas NIWT & CSOS were built around that goal; G.ho.st users are completely unable to define new applications at run-time through composition or through any other means.

6.6 Third-Party Integration

As will be explained in section 6.12.3, few existing AJAX libraries were reused. The most useful ones found by the research are less glamorous, but their usability, robustness and compatibility have proven their worth. SoftComplex [125] offers services and provides a number of widgets in their Tigra collection. Though their menu is easy to use and build, it does not support dynamic redefinition (insertions, deletions and other changes). NIWT used their calendar (date selector) and the colour chooser. Dynarch [71] makes an attractive and versatile freeware menu (among other utilities). It does not support dynamic change *a priori* nor does it support sub-composition substitution, but a work-around to the former was found and the latter was deemed inconsequential.

OpenLayers.js [119] was used for web-based GIS applications. A particular application was built with the notification mechanisms and components from NIWT and CSOS. This

contents of running applications but also dumps the former desktop and contents. This resulted in lengthy navigation between desktops.

application consisting of data displayed as markers on a map which are repositioned asynchronously in response to RT data. Section 6.10 will present a version of it ported to the iPhone.

6.7 Interoperability and Mashup

Applications programmed in Flash and Java Applets have successfully been integrated into the NIWT environment. These applications, by necessity of their purpose, communicate back-and-forth with the OOJS components for interoperability with the CSOS system and/or real-time collaboration. Most notably:

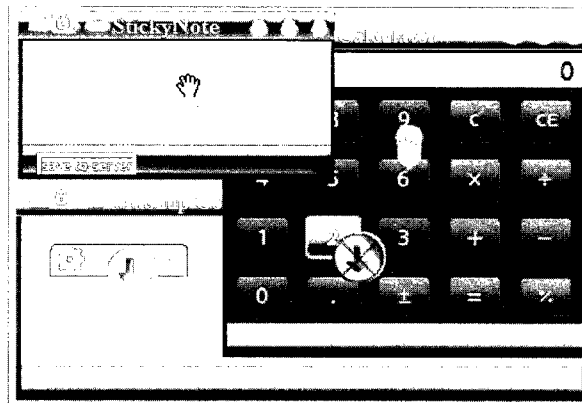


Figure 6.6: NIWT Pointers showing DnD states (counter-clockwise from top-right: off, dragging, source accepted, source rejected).

1. The Recovery Manager is a Java Applet that responds to AJAX calls and both collects and restores system state data.
2. Multiple pointing Devices made available for gestural manipulation of the desktop (see acknowledgements on p. 2) consisted of an invisible Java applet that communicates with a special depth-camera for gesture-based manipulation. The applet is fully invisible: NIWT Pointer components (Figure 6.6) with an inter-technology API built by this thesis work are controlled by the applet to move windowed applications on the desktop, send them to other users and perform mashups. The end result is the availability of multiple pointing-devices through the use of one or more depth-cameras (in response to section 2.15.15).

3. A full video-conferencing application (see acknowledgements on p. 2) was developed in Flex (ActionScript 3) whereby several people could dynamically join or quit an audio-video conference. Camera, microphone and media transmission were handled by a Flash Media server, but the application control and user information came from User Avatar components built as part of the work.

Figures 6.7 to 6.10 below illustrate the experiment with gestural control of the environment along with reactive mashup and interoperability. In Figure 6.7, the user is dragging an image book (right edge of left monitor) with his left hand (closed means 'grab') and is dragging it towards the map mashup application in the right monitor. In Figure 6.8, the user dropped the image book onto the map mashup which accepted it; however, it does not "know" what to do with simple images, so it stacks them on the bottom edge.

In Figure 6.9, the dragged took the XML Inspector application (right edge of right monitor in Figure 6.8) containing geo-location data relevant to the pictures and dropped it onto the map application: the map now has the information required and reacts by placing miniatures of the pictures in their correct positions on the map. In Figure 6.10, the user selects a picture over Brazil which causes another application, an Information Displayer (left edge of left monitor), to show the corresponding image in full size.

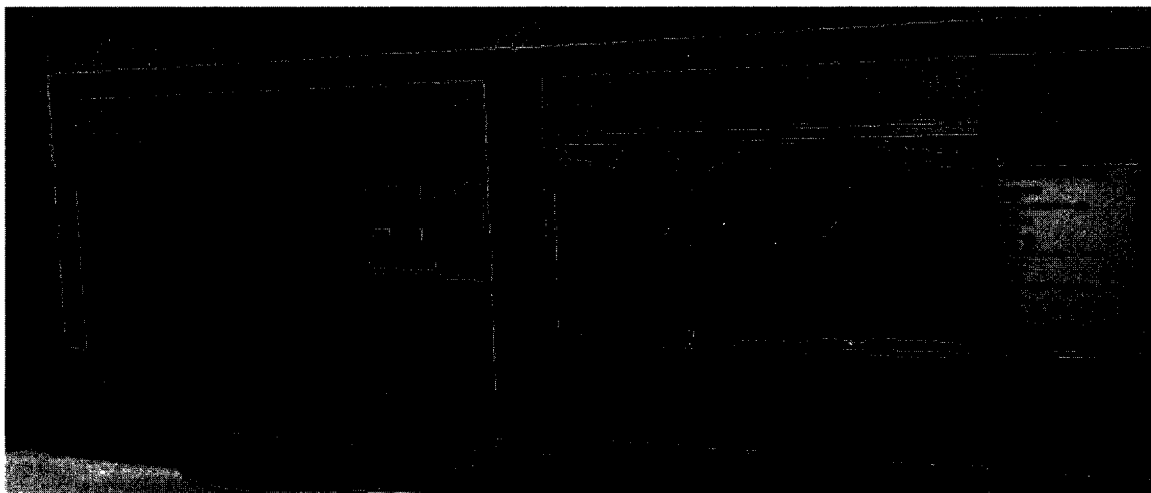


Figure 6.7: Gestural Manipulation with NIWT Pointers.



Figure 6.8: Gestural Mashup 1: pictures integrated to map.

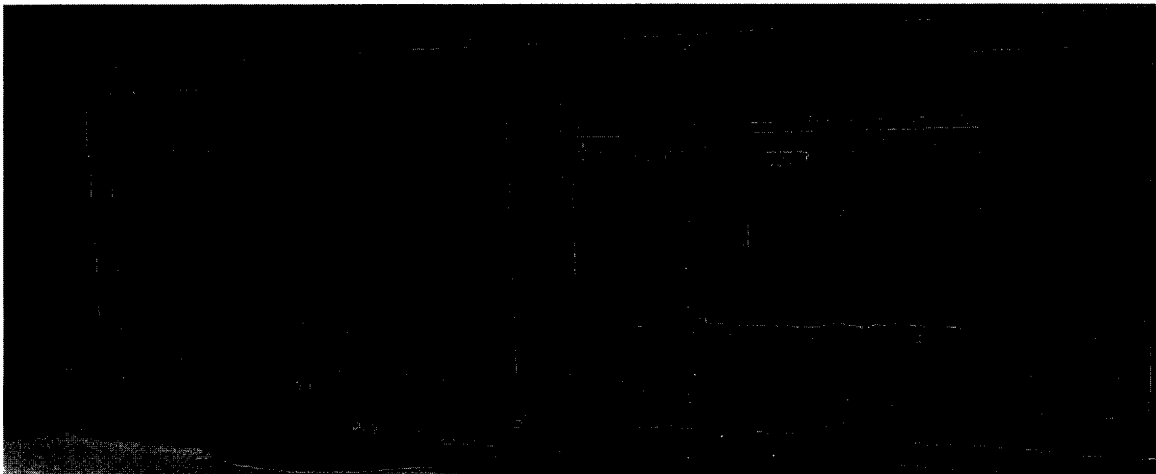


Figure 6.9: Gesture Mashup 2: integrating XML data.

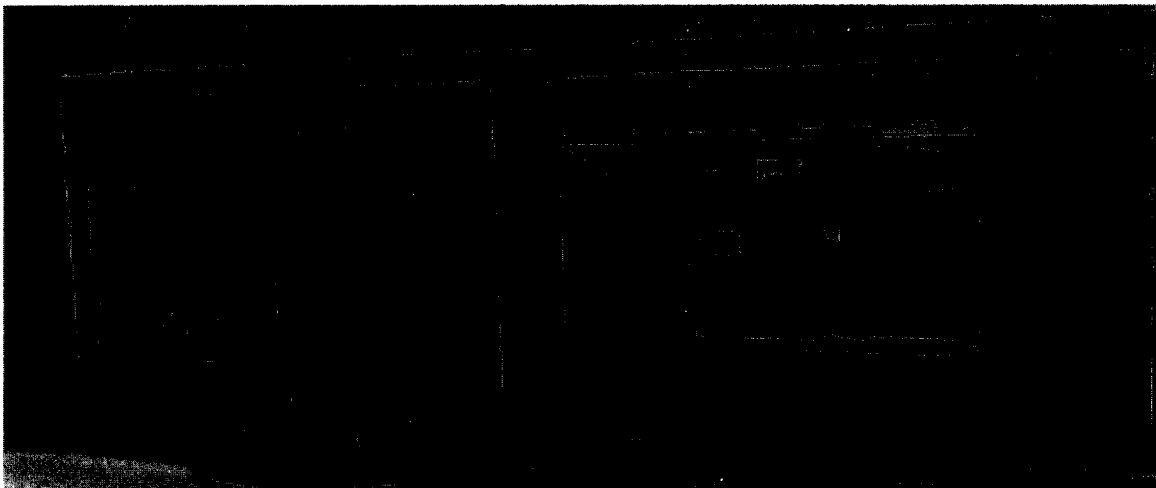


Figure 6.10: Gestural actions and interoperability.

A successful proof-of-concept prototype for DnD between a simple DHTML application (using NIWT components) and a similar one in Flash ActionScript 3 was successfully implemented using NIWT components. Though outside of the UC-IC project, it proved the usability of extracted components and inter-technology DnD.

6.8 Sharing and Collaborating

In response to the deficiency mentioned in section 2.15.2, UC-IC promotes simpler end-user access in the following ways:

1. Keeping files in a shared folder on the UC-IC file system makes them immediately visible to others through user avatars on their desktops. Users can browse them and import them into their own workspaces simply with DnD.
2. Defining application instances as ‘public’ also gives other users access to them through the user’s avatar plus affords data integrity on everyone else’s copy.
3. A user working on an application can drag it onto a user’s avatar which automatically opens the application on the corresponding user’s workspace with the data fully restored. If the other user is offline, the application will be received at next login. If the user does not have the application installed in her/his own workspace, it will be automatically and transparently installed with the dynamic resource loading mechanisms described in section 5.10.1.
4. Full real-time collaboration on applications is initiated with default permissions and settings by dragging a user’s avatar and dropping it on the desired applications. A CollaborationManager application allows users to add or remove applications to a given session, add or remove users to the session as well as change user permissions.

Figure 6.11 shows the JAWS file browser open to the files shared by the current user (Anny), the User Avatar’s mouse-over menu of the public applications of another user (Robin) with the selected file (picture taken in Brazil) having been opened in a FilePreviewer application equipped with a graphics annotations utility. Notice the blue

window decoration – contrasting with the default black – which shows the public status of the application.

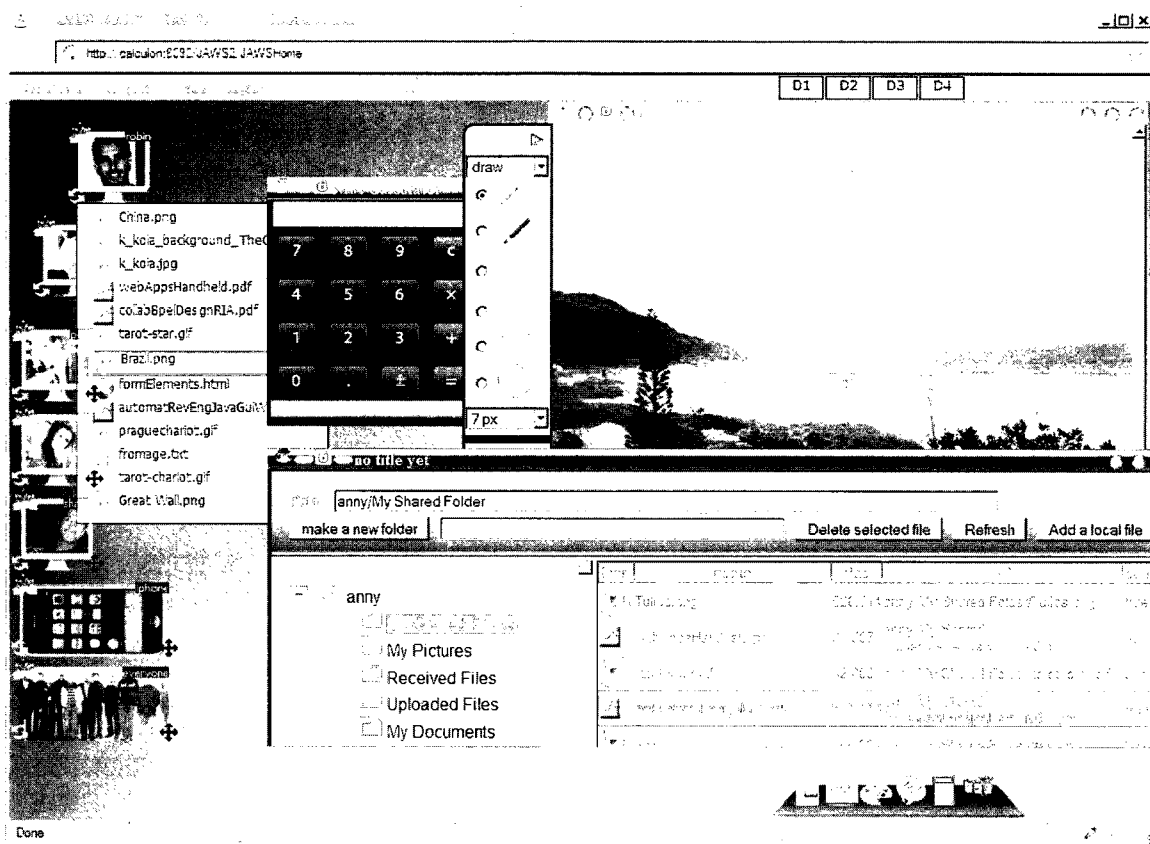


Figure 6.11: Sharing and public applications.

Sending an application to another user can barely be shown in still images, but the initiation of real-time collaboration is shown in Figure 6.12 with the result in Figure 6.13. Figure 6.12 shows two networked computers logged-in to UC-IC and working in the NIWT/CSOS environment. The top-right quadrant three user avatars, the middle one showing on-line status of the other computer has been dragged and dropped on a word processor to begin real-time collaboration. This is an excellent example of DnD being used not to move objects, but to launch a routine: the Avatar stays in place, but the mouse cursor changes to show the DnD state (Source Accepted). Figure 6.13 Shows the application with replicated data and preferences on the other computer along with data changes produced by action replication.

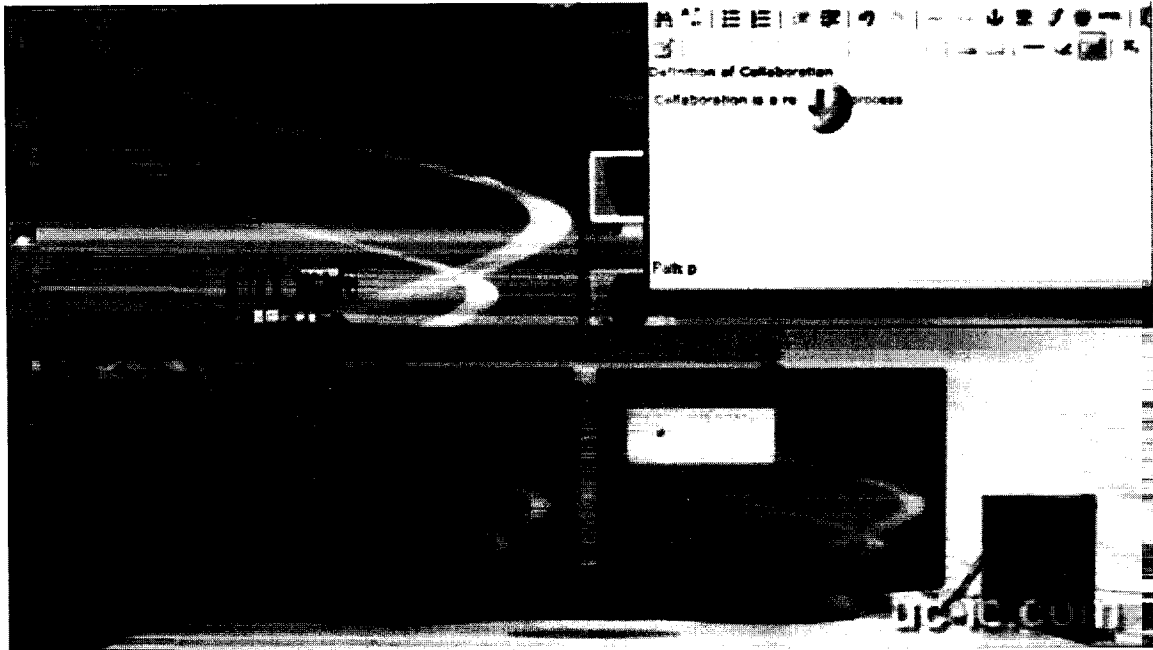


Figure 6.12: Starting collaboration on a word processor.

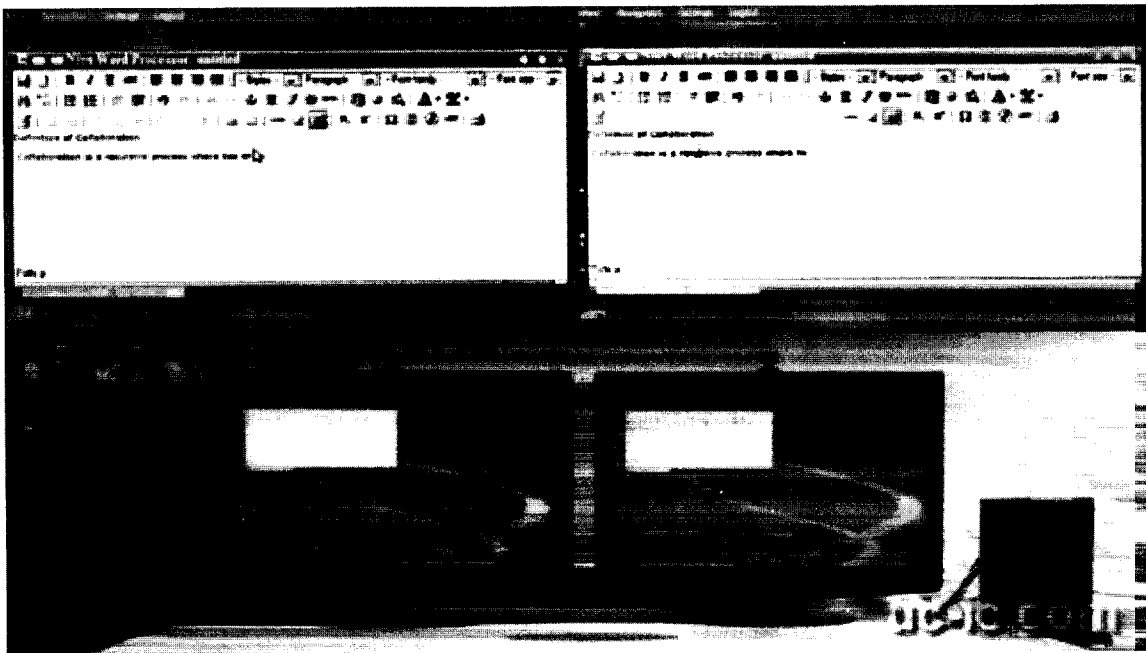


Figure 6.13: Real-time collaboration between two users on a word processor.

6.8.1 Applied Real-Time Collaboration, Locally Synchronized Copies and OLE

Recall that the mechanisms for browser-to-browser RPC automated by the contract object is the basis for real-time collaboration, locally synchronized copies and OLE. Applications that currently support real-time collaboration for data editing include: Sticky Note, Word Processor, and the Draw Tool. The Calculator also replicates button lighting on all copies.

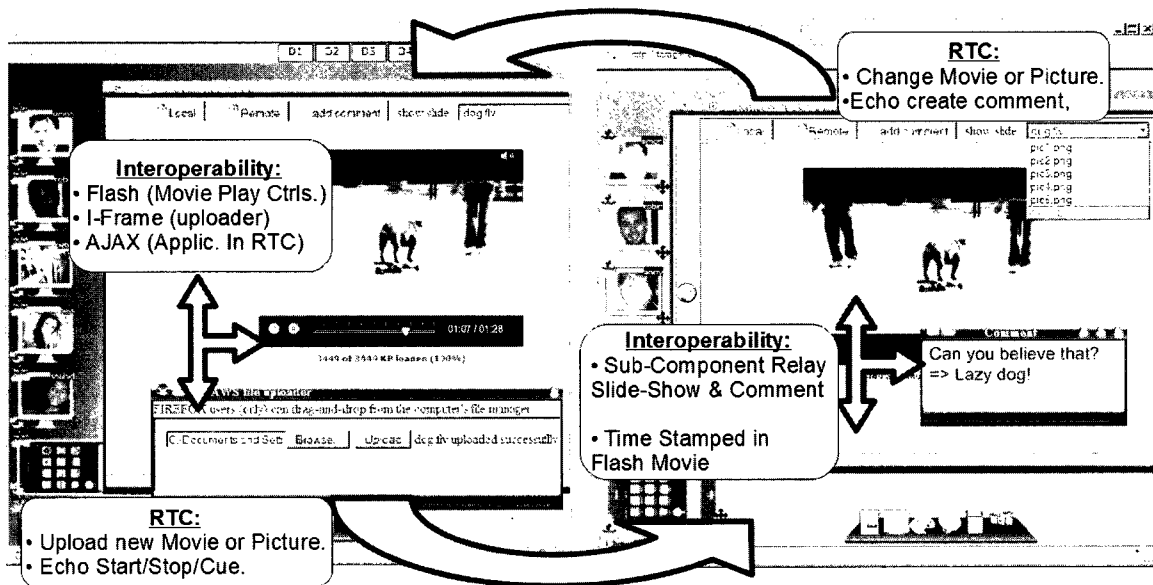


Figure 6.14: Slide Show Application: Collaboration and Interoperability

The Slide-Show application (Figure 6.14) allows any participant to add pictures or movies, navigate to any exact slide, propagate to other users playback control actions and add time-stamped comments. It is also an example of interoperability as playback controls (part of the Flash Object embedded in the showing slide) are propagated through CSOS (AJAX) to the Slide-Show application (JavaScript based MVC) which also coordinates with the Flash object to create time-stamped comments (Sticky Note application) that pop-up when the playback reaches the time marker. It is also an example of code-injection: the file uploader is a JSP contained in an I-Frame for the purpose of adding existing pictures and movies to the slide show. All the above (and more) are

propagated to all users in the real-time collaboration session.

Important Note: this is different from shared-desktops for the following reasons:

- 1.Applications in collaboration are only a subset of the applications on any given desktop; the whole set being potentially different from user to user.
- 2.Shared desktops do not support multiple concurrent real-time collaboration sessions on different sets of applications.
- 3.Echoing Actions need not reproduce all GUI components (e. g. uploading a file shows the uploader on one desktop only).

Fully functional locally synchronized copies have already been illustrated in Figure 5.7 on p. 109. Locally synchronized copies are supported by all the applications listed above and confirm the algorithm in section 5.7.2(1). Typical tests include the Draw Window scenario detailed above as well as a word-processor in a collaboration session of three users (on different hosts), each of which have four local copies on their workspaces; one on each of four desktops.

Delegation to sub-components by using contracts (section 5.7.1(2)) has been successfully tested with the NIWT Draw Window application where an empty window is the base application containing a Draw utility as a sub-component. The Draw Window defines its utility as a general delegate save for window events. The utility notifies the Window which forwards the notification to the collaboration Session object. Draw Shape objects (sub-components of the Draw Utility) provide a third level of delegation depth for the replication of creation, moving and deletion.

Figure 6.15 Shows this delegation on the Draw Window used simultaneously as locally synchronized copies – ‘pinned’ application on all desktops of a user (left window) – and in real-time collaboration with another user (right window). Whatever is done on any copy of the user on the left is reflected to all copies and the user on the right; whatever is done by the user on the right is reflected on all copies of the user on the left.

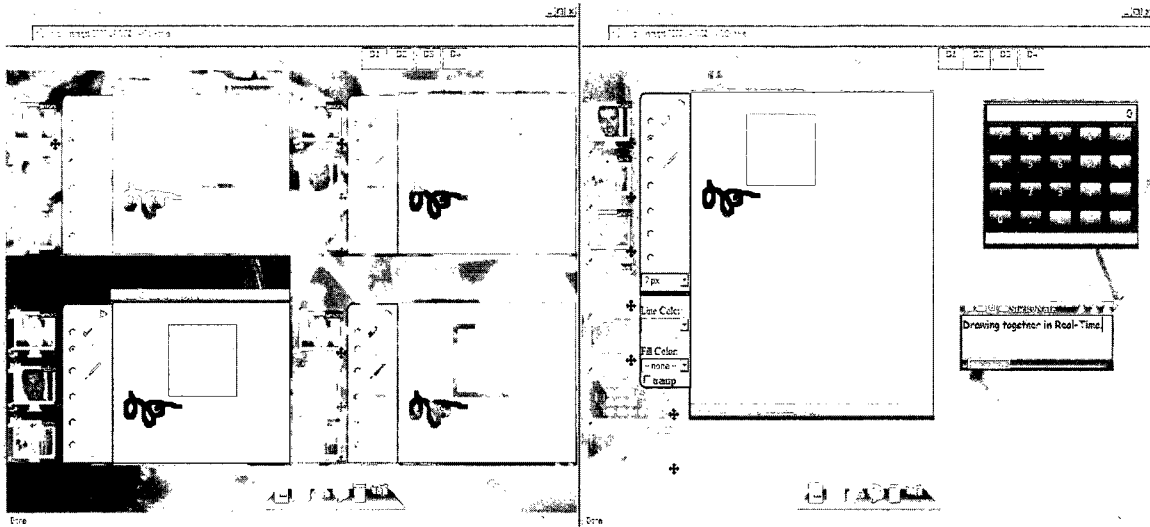


Figure 6.15: Simultaneous locally synchronized copies and real-time collaboration with 3-level delegation.

A more complex example of RPC delegation using the contract object (see section 5.7.1(2)) also proves dynamic aggregation of applications. The Mashup Container application consists of an empty window serving as a Drop Target for aggregating applications. When starting a real-time collaboration session on it, the remote copies will restore the applications already included in the mashup with their data and visual state. Any new application added to it will also be replicated on the remote copies, including all data as it was just before being mashed-in. Once a part of the mashup, these ‘sub-applications’ are automatically part of the collaboration, thus notifying all their own actions to the mashup application which then uses event forwarding and delegation concatenation to replicate the action to all other users in the session.

OLE has been proven by the test shown in Figure 5.8 on p. 109 which is programmed as a specific use-case scenario. Generalization for automation is proving difficult because:

1. Applications must specify which applications they are capable of embedding.
2. Applications must provide program logic for the placement of the embedded objects.
3. There is currently no mechanism to handle destruction in the case where keyboard deletion of the embedded object occurs.

4. Though the implementation of double-clicking to open the embedded object in a synchronized application is trivial, some such objects (e. g. a partial spreadsheet) are preferably edited in-place.

6.9 ORB Between Remote Web-Pages



Figure 6.16: Other user's Application Manager using ORB.

Figure 6.16 shows a prototype application built specifically to test ORB where a System Administrator must monitor and control user activity. The NIWT Application Manager GUI³⁷ lists applications currently running on a given user's environment, provides information about each application and its parent component and also provides a button to forcibly terminate a specific application instances. The GUI increases its content dynamically when new application instances are launched and removes listings

37 Not to be confused with the management component shown in Figure 3.14. The Application Manager GUI is a SDMD observer of the Application Manager component.

dynamically when application instances terminate. Two copies of the application are running: the one to the left (back) shows the applications running locally whereas the one to the right (front) is obtained by ORB with a remote user. Visually, only the title bar and the user fields in the application information differentiate the two application instances; however the local copy performs its functions locally whereas the ORB copy delegates its actions remotely through the browser-to-browser RPC layer and reacts to changes in its remote counterpart through the remote observer pattern.

Though browser-to-browser ORB has been successfully demonstrated (the source code for the proof-of-concept is given in Appendix K), there remain considerations for future study. These considerations can be categorized in two general scenarios that can be addressed by ORB:

1. The case where a remote object is only a logical/functional entity requiring no UI on either the proxy or remote sides.
2. The case where a remote object implies some kind of user interaction, be it as simple as a read-only display, a more complex application with user input (commands or data) or asynchronously refreshing a GUI on the proxy side in response to state changes from arbitrary causes.

Demonstrating requires an application where services are provided that do not directly relate to actions performed by the remote user. Because NIWT and CSOS are built specifically as an environment for end-user applications, this scenario remains in the realm of theory.

The second scenario can be demonstrating by echoing activity performed by a remote user. This entails a problem concerning the importance of the GUI:

1. The various functions required to build any HTML rarely need to be part of the contract object which automates the creation of the proxy object.
2. Automation mechanisms can still be built assuming the strict adherence to programming standards to overcome the problem in point 1. Nonetheless, in cases where

the GUI depends on the state of the user-environment, where it is dynamic or where it uses a sub-composition tree, the problem of acquiring HTML generators for components that are not available to the local client is not trivial.

3.To overcome points 1 and 2, it is tempting to acquire HTML generated on the remote side, but this incurs all the problems discussed in section 4.3.1(1) (General Indexing Schema).

Figure 6.16 involves two classes: the Application Manager GUI and the Application Listing. The problem of the GUI was solved by assuming the HTML generators are available locally, which is contrary to the goals of ORB. Other solutions currently supported are based on the dynamic loading of applications or of injecting specifically chosen scripts for the HTML generator only; these too contradict the principle of remote operation. Consequently, this thesis limited itself to laying-out the foundation protocols and mechanisms for browser-to-browser ORB, leaving the generation of the GUI to future work.

6.10 *Hand-Held Devices*

A version of the entire user-environment was ported to the HTC s360 smart-phone (see acknowledgements on p.2) running Opera Mobile version 6.x (Figure 6.17). In order to test the feasibility of reuse on small devices, the directives at the time specifically excluded optimizing for memory or bandwidth. Only the HTML code was redefined for the screen size and user interaction. The smart-phone version proved very long to load³⁸ and prone to time-out and memory failures, but it did succeed often enough to test and verify the successful functioning of:

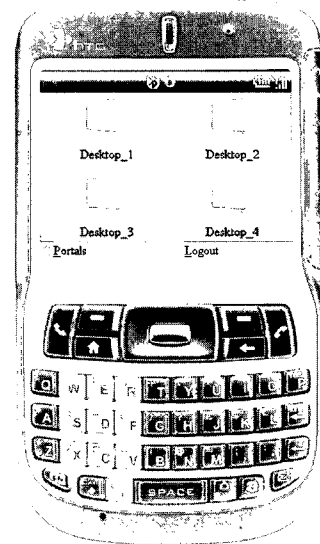


Figure 6.17 : Full user workspace on a smart phone

38 On the order of tens of minutes.

1. Applications supporting multiple instantiation: Sticky Note, Chat, e-mail, GPS map, Instant Messaging.
2. The ability to organize applications by desktop and navigate the system.
3. Full editing capability of the applications (not just read-only).
4. Sending applications to other desktops and users and the reception of applications.
5. Instant Notification from the server application regardless of the application currently showing.

This experiment proved the power of the MVC model shown here as only the HTML code was modified.

Another experiment (see acknowledgements on p. 2) consisted of porting to the iPhone the GIS Map showing RT data detailed at the end of section 6.6. This version of the application furthermore receives geo-located pictures ‘pushed-in’ from the server and places markers for these on the map, again using components and notification mechanisms built by this thesis work, whereby the pictures can be displayed in full (and closed) by the user through the iPhone’s touch pointer.

That experiment yielded three major achievements with respect to this thesis:

1. Integration of a third-party RIA (OpenLayers.js [119]) for the map application.
2. Extraction and use of only those components that are absolutely necessary for communication and application display.
3. Support of RT Streaming in unanticipated use-cases – the extracted components on an iPhone – with no substantial modification of the NIWT and CSOS packages.

Figure 6.18 Shows one user logged-in to UC-IC from a computer with the full NIWT/CSOS multi-desktop environment and another logged-in from an iPhone for the RT Map application with image reception (see acknowledgements on p.2). The inset (top right corner) shows the DnD state reflected by the cursor when dropping the image onto the Avatar for the iPhone user. Figure 6.19 shows the picture received by the iPhone user and opened on the map application.

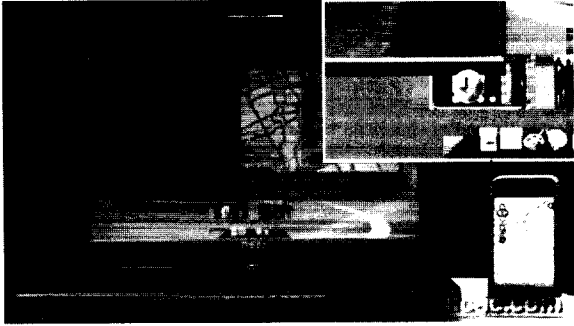


Figure 6.18: Sending geo-located picture to iPhone extracts of NIWT/CSOS.

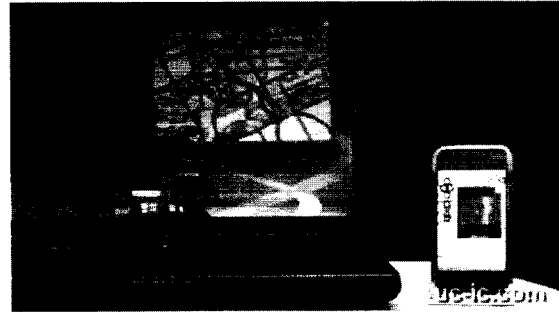


Figure 6.19: Image displayed on iPhone extracts of NIWT/CSOS.

A final experiment consisted of taking pictures with a cellphone and sending it in Real-Time to other users logged-in to UC-IC. Though mostly the work of other laboratory personnel, the experiment nonetheless fully depended on messaging protocols, components and applications designed and built in the context of this thesis.



Figure 6.20: Cell-phone sending a picture to other UC-IC users.

Figure 6.20 shows seven users, all logged-in to UC-IC, consisting of five computers an iPhone and a cellphone which has already taken and sent a picture to “all” users. This picture appears in a newly launched FilePreviewer application on the current desktop of each user.

6.11 *Unified Communications*

UC as per [12][13][14] (see section 2.1) in the context of this thesis is provided by the user-environment (client-side) itself through the support for the development of new (and reuse of existing) applications that provide the different business communication services. Applications developed for communication include, but are not limited to: two versions of a video-conferencing tool, Video players, two mashup Applications, a Chat application, a simple messaging application and an e-mail client with a mail reader which was augmented with the direct send-to feature. Presence information is provided by the User Avatars, updating lists of shared and public applications and GIS tools for RT media streaming. The RT control aspect of UC is provided by the implementation of SIP in the JAWS server and the UI elements on in NIWT and CSOS³⁹. The algorithms, protocols and architectural elements for the automation of shared event replication also benefits the RT control aspect of UC by adding control over contents and propagating controls to all participating users (permissions allowing).

6.12 *Drawbacks with the Results*

The very decision to build an applications platform using web (browser-managed) technologies brings about a number of drawbacks. This section examines these and exposes others which naturally flow from the design decisions.

39 SIP and the JAWS server belong to the scope of another thesis. See Acknowledgements on p. 2.

6.12.1 Caching and Memory Leaks

Issues with any given browser's memory management are by no means resolved. Browser caching causes configuration problems, reusing outdated JavaScript definitions when the JavaScript files have been upgraded on the applications server. NIWT and CSOS integrated all the scripts that could be found to prevent caching: they have all proven unreliable. Only explicit browser configuration done by each end-user individually can overcome the problem at this time; an unrealistic requirement of the mass population.

Contrary to what is expected, [118] has found that deleting HTML elements from the document does not free the memory. This is particularly troublesome in RT data applications that add and remove visual items (e. g. air traffic on a map). Though the same paper explains policies for closure (class) creation, what it either omitted or failed to identify is a problem that could be described as the opposite of dangling (or null) pointers. In Java and C++, if an object is destroyed, any remaining pointers to it will point to 'nothing' (and throw exceptions accordingly). In JavaScript, the contrary is true: regardless of the techniques used to destroy an object, as long as a pointer remains on it, the object will continue to exist. Using the deleted, nullified or otherwise destroyed pointers will usually throw exceptions, but the other pointers that have not been 'emptied' will still be usable and the object will continue its operations despite what was intended. This is not trivial as pointers to any object are created dynamically as discussed in Appendix I, Memory Issues in DHTML. If RIA are to become serious contenders for real distributed applications and ubiquity, the dependability of pointer destruction and garbage collection needs to be improved.

6.12.2 Cross-Domain Interoperability

Section 5.3.2 leads to the notion of integrating resources from web domains other than UC-IC. This cross-domain communication is not supported *a priori* by AJAX for reasons of security. In particular, code-injection is prohibited in such circumstances. Firefox 3.x

has implemented means to support such communication as granted by new W3C standards [127][128]. Experimentation and design in this area are left to future work.

6.12.3 Reusing Third-Party Software

Engineering practice promotes reusing resources when feasible. In the case of RIA, these resources are typically libraries for applications and utilities. This thesis imposed strict criteria on candidate resources concerning interoperability, capabilities and scalability. For any application (barring obvious cases), the support for multiple concurrent and independent instances created dynamically must be ensured. Reusability must also be supported in terms of easy combination with other resources in order to develop new libraries and/or components. For example, if a library offers a visual container for a given application interface, the container must be capable of containing other random applications developed with any other technology compatible with the user environment.

All the above has proven difficult to find in existing resources for reasons given in section 2.15.4. Many resources considered failed to exhibit proper component oriented design and thus prevented dynamic modification. Others simply would not tolerate the inclusion of any other library. This is cause for concern as some of the resources rejected are highly praised by some important industrial players⁴⁰.

6.12.4 Browser Compliance Issues

Strict compliance to W3C standards is not enough. The problem of cross-browser compatibility is well known and apparently not headed for resolution. For example, Firefox 3.x supports drag-and-drop file upload from the PC desktop and Mozilla defines several new CSS that otherwise require complex image manipulation. Other, less fundamental, problems can occur either from one browser to another or between different versions of the same browser. Case-in-point: Firefox handles the default z-index of HTML elements (which elements cover which) differently from versions 2.5 to 3.0 to

40 Noteworthy case-in-point: the DOJO package is used and promoted by IBM. [126]

3.5! This and default mouse event handlers for images have proven most problematic at development and testing times.

6.12.5 *Web MVC not Automatic*

Recall the MVC design by which JavaScript objects generate and control DHTML GUI in the web document (section 4.2). The design was intended to better conform to OOP in the style of the packages Java.awt and Javax.swing. As such, at any given time, getters and setters can be called to change visual aspects of a GUI component while the HTML may or may not already be inserted in the document. Unlike most Javax.swing components, setting a JavaScript variable corresponding to a GUI property is not automatically reflected on the visual component. This adds extra duty of care to developers who must ensure that setters attempt to change the HTML inserted, and catch exceptions when the code is not yet inserted in the document.

6.12.6 *Weak OOJS*

Though this thesis showed how intricate OOP can be achieved in JavaScript, important attributes of Java, C++ etc. are simply impossible. For example (and among many others), private and protected members are simply not supported which can lead to programmers ignoring getter and setter functions in favour of directly manipulating variables. Solid MVC cannot be enforced as direct manipulation of DOM remains available.

```
function someRoutine() {
    while( !globalArray.contains( "control value" ) ){
        modifyArray();
    }
}
function modifyArray() {
    globalArray.push( "control value" );
}
```

Listing 6.1: Looping on a volatile variable.

Extensive experiments show that JavaScript does not handle volatile variables very well. Consider Listing 6.1: debugging tools suggest that the globalArray object used in the

while loop never reflects the change from the calling `modifyArray()`, a common situation when compilers (the JavaScript engine, in this case) creates a copy of the object rather than performing a look-up every time, thus causing the condition to remain false and resulting in an endless loop.

Similarly to multiple inheritance in C++, careful design must be shown as unintended effects can result from using Plugs. If the class originally had members in common with the plug, then the class is left in a weakened state with respect to its original definition. This is a naming conflict problem which also happens with hard-plugging and known in C++ multiple inheritance [98].

Successful experiments were made to restore classes to a previous state after removing a Plug (unplugging) by keeping an inventory of members and definitions. But a problem arises when more than one Plug is used and unplugging is not done in the correct reverse sequence: to which state must the component be restored? For these and other reasons, it was decided to simply not support state restoration and leave duty of care to the programmer.

6.12.7 The Set Manager is Needlessly Complex

The Set Manager (section 5.9) was built for the AJIP project specifically and implements a registry, control structures and the delegation pattern. It can be abstracted and mechanisms can be provided for the general case, but this each case would remain static to the specific set of applications as defined by the developer

The Mashup component already partially implements a registry and observer pattern and the Contract object already provides a service descriptor. These two entities can be extended to support “Reactive Mashups” where applications in a mashup can be notified of newly added applications and use the contract in the program logic to decide if the services of the new components are useful; if they are, an observer pattern can be established between them. Furthermore, filtration of acceptable (or not acceptable) applications are already included in the Mashup Plug and independence of the set is

implicit in the mashup instance. Saving and restoration of the mashup is partially supported with the dynamic data mechanisms discussed in section 5.10.2 leaving only the re-association of applications at restoration time untested.

The benefit from this consist of combining the third type of mashups described by [18] (see section 2.2), user application mashups, with the benefits of the first type (data-flow) and extended to include event-flow. This is left to future research.

6.12.8 RPC Returns and Type-Constraining

Two aspects remain clearly missing: data typing in the contract (useful for validation or decision trees) and return values from browser-to-browser RPC. Data typing is currently done by the marshalling mechanism (section 5.5, RPC Between Browser-Contained Clients) on the assumption that arguments of correct data type are already used. Also, though the RPC module does expect a return value and implements a “return RPC” routine, the principle of deferred synchronicity discussed in section 5.6 means that a “return handler” must be specified or else the recipient of the return value will not know what to do with it. Though Listing 6.2 shows how these issues could be easily obtained, they remain unimplemented at this time.

```
this.contract.sizeEvent = {
  setSize : {
    return      : "void",
    returnFunc  : "confirmSize",
    newWidth    : "Number",
    newHeight   : "Number"
  },
  getDistance : {
    return      : "Number",
    returnFunc  : null,
    startPoint  : "Array",
    endPoint    : "Array"
  },
  ...
}
```

Listing 6.2: Contract with return and data typing.

Chapter 7

Conclusion

This thesis has proven the feasibility and scalability of a web-based real-time collaboration multi-desktop client-side platform which supports the development of new applications including the integration and interoperability of important web-document compatible technologies. The current state of AJAX provides all the necessary features for its design and implementation. Mashable applications assembled from components built in various web-based technologies open the door to a user-environment where web-services are only the beginning: the user decides the rest.

7.1 Concepts Addressed in this Thesis

This thesis dealt with:

1. The architecture of a web-based multi-desktop metaphor user environment (WebOS) to support dynamic change in composition (contents) as well as supporting asynchronous (unsolicited) commands from a server.
2. Scalability of rich internet applications in terms of supporting all browser-contained technologies in the same applications platform and providing interoperability between them transparently to developers.
3. Metamorphic properties of end-user applications to support the persistence and restoration of their data and state in a nomadic context as well as replicate metamorphic changes dynamically in a collaborative context includes action replication in object linking and embedding and mashable applications.
4. Extending well established software design patterns, protocols and a programming paradigm from compiled languages (e. g. Java and dot.NET) to web-programming and adapt them for browser to browser (client to client) interoperability. These include the observer pattern, remote observer pattern and remote procedure call.

5. Real-time collaboration between random applications on remote user workspaces as well as between application components on the same user workspace. In particular, the design of a platform for end-user application development which provides the requisite API, protocols and routines transparently to application developers.
6. Object oriented programming and component oriented design for reusability and extensibility supporting the development of user environments and applications in a web browser based context. In particular, full and correct support of the model-view-control design pattern in Rich Internet Application and complete features of object orientation (encapsulation, inheritance and introspection) in JavaScript. This is extended to a design pattern for single-data, multiple displays through local synchronization which interacts with remote replication routines.

7.2 Contributions of this Thesis

1. A new programming paradigm for programming rich internet applications comprising:
 - (a) An extension of the factory pattern to support of multiple, volatile and removable inheritance as well as full-featured introspection in object oriented JavaScript.
 - (b) Design of a dynamic data structure and an implementation in XML to support hierarchical persistence and delegable restoration of the data and state of metamorphic objects.
 - (c) A new coding practice to better implement the model-view-controller pattern in web-applications with a component oriented design that supports all-way communication and control. This includes other applications on the same client-side, server-side routines and even components on another client-side can query and/or call functions on any local component.

2. A new thick-client design for a fully browser-contained multi-desktop metaphor which supports sustained activity during network disconnections as well as a protocol for recovery from failure: a first in WebOS implementations. This is comprised of:
 - (a) The Client-Side Operational Structure built on deferred synchronicity in AJAX to better approximate RT activity which accepts asynchronous messages and commands from the server or other (remote) clients.
 - (b) The architecture and algorithmic resources for application development which supports interoperability between multiple browser-contained technologies, notably Flash and Applets and interoperability between them and the DHTML environment, including increasing web-based drag-and-drop to full-featured object association for procedure launching.
3. Real-time collaboration between random rich internet applications through:
 - (a) A new action replication algorithm for both local and remote copies of components with new mappings and new mechanisms for redundancy evasion.
 - (b) Adaptation for browser-to-browser (client-to-client) of RPC, ORB and the automation of these on random and metamorphic rich internet applications.
 - (c) Design and implementation of the new contract object for the novel automation of Browser-to-Browser Remote Procedure Call, Locally Synchronized Copies of applications (which includes Object Linking and Embedding), and Browser-to-Browser Object Request Brokering.
4. Design and implementation of a new concept: multiple sets of inter-dependent applications and laying the ground work for responsive mashups which combines the first and third models of mashup in [18].

7.3 Future Research

1. Ensuring sequential continuity in distributed computing based on AJAX.
2. HTML 5 and ECMAScript 4 (JavaScript 2): their potential benefit to programming paradigm and web-based real-time collaboration.
3. Portability to hand-held devices: Especially in light of the many new AJAX capable browsers such as Fennec by Mozilla.
4. The potential benefits of Adobe AIR for a browser-independent implementation of NIWT and CSOS.
5. Improving pointer destruction and garbage collection for both the web-page's DOM and JavaScript.
6. Merging the collaboration mechanisms (browser-to-browser RPC and locally synchronized copies) with protocols for data integrity.
7. Further study on browser-to-browser ORB.
8. Responsive Mashup as per section 6.12.7.
9. Client-initiated cross-domain integration and collaboration through new W3C standards in [127][128]

Summary of Appendices:

All appendices are available by contacting the author via e-mail at the address:

robin.tropper@alumni.uottawa.ca

Appendix A Issues with the Browser as a UI platform

Provides a discussion on the difficulties of using DHTML for the implementation of the desktop metaphor with the expectations of the features of the Java or native OS windowing toolkits.

Appendix B Browser-to-Browser Marshalling

Provides a discussion on marshalling in JavaScript, its automation with the contract object as well as the JavaScript source-code for the serialization, deserialization and function call reconstruction in browser to browser RPC and ORB.

Appendix C OOJS and Polymorphism

Discusses how polymorphism can be obtained in JavaScript but also the dangers in doing so compared with Java or C++.

Appendix D Code Conventions

Details the coding conventions adopted by NIWT in JavaScript and CSS to ensure readability of code and ensure the proper functioning of automated mechanisms like dynamic resource loading.

Appendix E String-Code Programming

A discussion on the web-programming paradigm introduced by this thesis. A comparison is made on the coding of a simple GUI using DHTML using the DOM, with tables and in Java.

Appendix F Component Hierarchy for CSOS and NIWT

Provides class diagrams and a discussion on the classes used for CSOS and NIWT and how they relate to one another.

Appendix G Multiple Concurrent CSS

Demonstrates and explains how the coding standards described in Appendix D allow multiple style-sheets to affect different copies of equivalent parts of the same page with different visual styles.

Appendix H Namespaces: a JavaScript solution to DLL hell in RIA

Discusses pitfalls with JavaScript resource inclusion and presents a design and successful experiments which enable different versions of the same JavaScript resource to operate concurrently.

Appendix I Memory Issues in DHTML

Provides a short discussion on memory issues with dynamic applications programmed in DHTML as compared to Java or C++ and explores some ways to counteract them as well as the difficulties preventing a dependable solution.

Appendix J Source-Code for the Notification Object

The source-code for the notification object shows how the problems of redundant event replication and endless loops detailed in section 5.7.2 are averted.

Appendix K Browser-to-Browser Object Request Brokering

Discusses the notion of Object Request Brokering in the state-of-the-art and demonstrates the automatic generation of proxies for browser-to-browser remote object replication.

Appendix L A brief history of UC-IC: Iterations and Purposes

Provides a description of the different iterations of the project that led to UC-IC and how the CSOS and NIWT client side evolved with a special focus on added features and architectural changes.

Appendix M Towards a Distributed Operating System: Vision on the future

Discusses the notion of a distributed operating system, how the industry currently supports or doesn't support the notion and how existing features of UC-IC can be extended to achieve that goal.

References

- [1] Zepeda, J.S.; Chapa, S.V., “From Desktop Applications Towards Ajax Web Applications”, *Electrical and Electronics Engineering, 2007. ICEEE 2007. 4th International Conference on*, pp.193-196, Sept. 2007
- [2] Crane, Dave, and Eric Pascarello. *Ajax in Action*. Greenwich: Manning, 2005. ISBN: 1932394613
- [3] Douglass, Bruce. *Real Time Uml*. Boston: Addison-Wesley, 2004.
- [4] W3C, *XMLHttpRequest*. URL: <http://www.w3.org/TR/XMLHttpRequest/>
As seen in November 2009
- [5] W3C (World Wide Web Consortium) URL:<http://www.w3.org/>
- [6] DIX, Alan et.al. *Human-Computer Interaction*. Boston: Pearson, 2004. ISBN: 0130461091.
- [7] Mclaughlin, Brett. *Java and Xml*. Sebastopol: O'Reilly, 2000.
- [8] eWeek.com: *Open XML Incompatible With GPL*
URL: <http://www.eweek.com/c/a/Enterprise-Applications/Open-XML-Incompatible-With-GPL/>
As seen in November 2009
- [9] Mahemoff, Michael. *Ajax Design Patterns*. Sebastopol: O'Reilly, 2006. ISBN: 0596101805
- [10] Douglas C. Schmidt, Department of Electrical and Computer Engineering, University of California, Irvine.
URL: <http://www.dre.vanderbilt.edu/~schmidt/reuse-lessons.html>
As seen in November 2009

- [11] OMG Unified Modeling Language™ (OMG UML), *Infrastructure Version 2.2*.
OMG Document Number: formal/2009-02-04 Standard document
URL: <http://www.omg.org/spec/UML/2.2/Infrastructure>
As seen in November 2009
- [12] Adaptive Path: *Aurora Concept*. URL: <http://adaptivepath.com/aurora/>
As seen in November 2009
- [13] Mozilla Labs: *Introducing the Concept Series; Call for Participation*
URL: <http://labs.mozilla.com/2008/08/introducing-the-concept-series-call-for-participation/>
As seen in November 2009
- [14] Cisco: *Unified Communications Create Adaptive Workspaces*
URL: http://www.cisco.com/en/US/netsol/ns151/networking_solutions_unified_communications_home.html
As seen in November 2009
- [15] Schreiner, K., “Unified Messaging: Will It Finally Meet Its Promise?,” *IT Professional* , vol.9, no.4, pp.56-60, July-Aug. 2007
- [16] Andrews, C. 2001. “Unified communication systems.” *Crossroads* vol. 8, no. 1 (Sept. 2001), pp. 13-17.
- [17] Imieliński, T. and Badrinath, B. R. 1993. “Data management for mobile computing.” *SIGMOD Rec.*, vol. 22, no. 1 (Mar. 1993), pp. 34-39.
- [18] Murugesan, S., “Understanding Web 2.0,” *IT Professional* , vol.9, no.4, pp.34-41, July-Aug. 2007.
- [19] Hye-Jin Jin; Hong-Chul Lee, “Web services development methodology using the mashup technology”, *Smart Manufacturing Application, 2008. ICSMA 2008. International Conference on*, pp.559-562, 2008

- [20] Hong Chen; Ikeuchi, N.; Qun Jin, “Implementation of Ubiquitous Personal Study Using Web 2.0 Mash-up and OSS Technologies”, *Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008. 22nd International Conference on*, pp.1573-1578, 2008
- [21] The Java Tutorials: Lesson: *Drag and Drop and Data Transfer*
URL: <http://java.sun.com/docs/books/tutorial/uiswing/dnd/index.html>
As seen in November 2009
- [22] Quirksmode: *Drag and Drop*. URL: <http://www.quirksmode.org/js/dragdrop.html>
As seen in November 2009
- [23] Rico JavaScript for Rich Internet Applications. URL: <http://openrico.org/>
As seen in November 2009
- [24] Mozilla Developer: *Drag and Drop*.
URL: https://developer.mozilla.org/En/DragDrop/Drag_and_Drop
As seen in November 2009
- [25] Microsoft Developer Network: *Drag and Drop*.
URL: <http://msdn.microsoft.com/en-us/library/ms749074.aspx>
As seen in November 2009
- [26] Microsoft Help and Support: *OLE Concepts and Requirements Overview*
URL: <http://support.microsoft.com/kb/86008/en-us>
As seen in November 2009
- [27] Hao Han; Tokuda, T., “A Method for Integration of Web Applications Based on Information Extraction”, *Web Engineering, 2008. ICWE '08. Eighth International Conference on*, pp.189-195, 2008
- [28] Yahoo Pipes. URL: <http://pipes.yahoo.com/pipes/>
As seen in November 2009
- [29] Microsoft Popfly. URL: <http://www.popfly.com/>
As seen in November 2009

- [30] *What is Google Mashup Editor?* URL: <http://code.google.com/gme/index.html>
As seen in November 2009
- [31] Mozilla Labs: *Introducing Ubiquity*
URL: <http://labs.mozilla.com/2008/08/introducing-ubiquity/>
As seen in November 2009
- [32] Microsoft Developer Network: *About Dynamic Data Exchange*
URL: [http://msdn.microsoft.com/en-us/library/ms648774\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms648774(VS.85).aspx)
As seen in November 2009
- [33] Microsoft Systems Journal, May 1996. *How OLE and COM Solve the Problems of Component Software Design*. URL:
<http://www.microsoft.com/msj/archive/S2EC.aspx>
As seen in November 2009
- [34] Gani, S.; Picuri, P., “The object revolution. How COM technology is changing the way we do business”, *Computing & Control Engineering Journal*, vol.6, no.3, pp.108-112, Jun 1995
- [35] JavaDde: *Windows DDE Protocol for Java*
URL: <http://www.nevaobject.com/jddedetails.htm>
As seen in November 2009
- [36] Zimbra Ajax Linking and Embedding
URL: <http://www.zimbrablog.com/blog/archives/2006/04/a-pint-of-ale-ajax-linking-and-embedding.html>
As seen in November 2009
- [37] Elsaesser, Derek Shawn. *Metamorphic Objects for Dynamic Reconfiguration of CORBA-based Applications*. M.A.Sc. Thesis submitted to the University of Ottawa, 2000

- [38] EPC Global Inc. (*Whitepaper*)
URL: http://www.epcglobalinc.org/standards/ale/ale_1_0-standard-20050915.pdf
As seen in November 2009
- [39] Mitel Presence Magazine, January 2008
URL: http://www.mitel.com/presence/current_issue/pdf/300_842__presence_Jan%202008_FINAL.pdf
As seen in November 2009
- [40] Zhang, Y. and Boncz, P. 2007. "XRPC: interoperable and efficient distributed Xquery", *Proceedings of the 33rd international Conference on Very Large Data Bases* (Vienna, Austria, September 23 - 27, 2007). Very Large Data Bases. VLDB Endowment, 99-110.
- [41] Moshchuk, A., Gribble, S. D., and Levy, H. M. 2008. "Flashproxy: transparently enabling rich web content via remote execution", *Proceeding of the 6th international Conference on Mobile Systems, Applications, and Services* (Breckenridge, CO, USA, June 17 - 20, 2008). MobiSys '08. ACM, New York, NY, 81-93.
- [42] Hui-Chieh Lu; Yen-Ping Chu; Ruey-Kai Sheu; Win-Tsung Lo, "A Generic Application Sharing Architecture Based on Message-Oriented Middleware Platform", *Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference on*, pp.1-5, 2006
- [43] Imine, A. 2008. "Decentralized concurrency control for real-time collaborative editors", *Proceedings of the 8th international Conference on New Technologies in Distributed Systems* (Lyon, France, June 23 - 27, 2008). NOTERE '08. ACM, New York, NY, 1-9.
- [44] Stallings, William. *Operating Systems*. Englewood Cliffs: Prentice Hall, 2001.
ISBN: 0130319996

- [45] Runhe Huang; Jianhua Ma, “A general purpose virtual collaboration room”, *Engineering of Complex Computer Systems, 1999. ICECCS '99. Fifth IEEE International Conference on*, pp.21-29, Oct 1999
- [46] Baomin Xu; Michael Rose; Zongkai Lin, “A Novel Approach to Convert Single-user Applications into Collaborative Applications”, *Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference on*, pp.1-5, 2006.
- [47] Seacord, R. C. and Hissam, S. A. 1998. “Browsers for distributed systems: Universal paradigm or siren’s song?”, *World Wide Web* 1, 4 (Apr. 1998), 181-191.
- [48] Vahdat, A., Eastham, P., Yoshokawa, C., Belani, E., Anderson, T., Culler, D., and Dahlin, M. 1997 *Webos: Operating System Services for Wide Area Applications*. Technical Report. UMI Order Number: CSD-97-938., University of California at Berkeley.
- [49] Lawton, G., “Moving the OS to the Web”, *Computer* , vol.41, no.3, pp.16-19, March 2008.
- [50] Plasmeijer, R., Jansen, J. M., Koopman, P., and Achten, P. 2008. “Declarative Ajax and client side evaluation of workflows using iTasks”, *Proceedings of the 10th international ACM SIGPLAN Conference on Principles and Practice of Declarative Programming* (Valencia, Spain, July 15 - 17, 2008). PPDP '08. ACM, New York, NY, pp. 56-66.
- [51] Stamey, J. and Richardson, T. 2006. “Middleware development with AJAX.” *Journal of Computing Sciences in Colleges*, Volume 22, Issue 2 (Dec. 2006), pp. 281-287.
- [52] Hanakawa, N.; Ikemiya, N., “A Web Browser for Ajax Approach with Asynchronous Communication Model”, *Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on*, pp.808-814, 2006.

- [53] Maki, D. and Iwasaki, H. 2007. “A portable javascript thread library for Ajax applications”, *Companion To the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion* (Montreal, Quebec, Canada, October 21 - 25, 2007). OOPSLA '07. ACM, New York, NY, 817-818.
- [54] Direct Web Remoting. URL: <http://directwebremoting.org/>
As seen in November 2009
- [55] Dojo Toolkit. URL: <http://www.dojotoolkit.com/>
As seen in November 2009
- [56] Checiu, Laurentiu. *A new XML Schema Mapping Architecture*.
M.A.Sc. Thesis submitted to the University of Ottawa, 2008
- [57] Microsoft Developer Network: Windows API
URL: <http://msdn.microsoft.com/en-us/library/aa383750.aspx>
As seen in November 2009
- [58] Microsoft .NET Framework. URL: <http://www.microsoft.com/net/>
As seen in November 2009
- [59] RAD Application Development Software | Delphi from Embarcadero Technology.
URL: <http://www.embarcadero.com/products/Delphi/>
As seen in November 2009
- [60] QT White Paper.
URL: <http://www.qtsoftware.com/products/files/pdf/qt-4.4-whitepaper>
As seen in November 2009
- [61] GTK+ Project. URL: <http://www.gtk.org/>
As seen in November 2009
- [62] The Java Tutorials: *Getting Started with Swing*
URL: <http://java.sun.com/docs/books/tutorial/uiswing/start/index.html>
As seen in November 2009

- [63] Deitel, Harvey, and Paul Deitel. *Java How to Program*. Englewood Cliffs: Prentice Hall, 1999. ISBN: 0130125075
- [64] JBoss Portal: *The open source and open standards Java Portal*.
URL: <http://www.jboss.org/jbossportal/>
As seen in November 2009
- [65] The Yahoo! User Interface Library (YUI)
URL: <http://developer.yahoo.com/yui/>
As seen in November 2009
- [66] GWT Google Web Toolkit – Google Code.google.com
URL: <http://code.google.com/webtoolkit/>
As seen in November 2009
- [67] Google Wave API Overview - Google Wave API
<http://code.google.com/intl/fr/apis/wave/guide.html>
As seen in November 2009
- [68] Java Server Faces Technology Tutorial
URL: <http://java.sun.com/javaee/javaserverfaces/docs/archive/JSF.pdf>
As seen in November 2009
- [69] Zend. URL: <http://www.zend.com/en/>
As seen in November 2009
- [70] Ext JS. URL: <http://extjs.com/>
As seen in November 2009
- [71] Dynarch. URL: <http://www.dynarch.com/>
As seen in November 2009
- [72] Laszlo Dashboard Demo URL: <http://www.laszlosystems.com/lps/sample-apps/dashboard/dashboard.lzo?lzt=html>
As seen in November 2009

- [73] Six Revisions, *15 Free Tools for Web-based Collaboration*
URL: <http://sixrevisions.com/tools/15-free-tools-for-web-based-collaboration/>
As seen in November 2009
- [74] YouOS: People. URL: <http://www.youos.com/html/static/team.html>
As seen in November 2009
- [75] Bindows. URL: <http://www.bindows.com/>
As seen in November 2009
- [76] Cloudo. URL: <http://www.cloudo.com/>
As seen in November 2009
- [77] Desktop Two. URL: <http://desktoptwo.com/>
As seen in April 2009
- [78] Google Docs: *Create documents, spreadsheets and presentations online*
URL: <http://www.google.com/google-d-s/intl/en/tour1.html>
As seen in November 2009
- [79] Zoho. URL: <http://www.zoho.com/>
As seen in November 2009
- [80] Windows Vista Enterprise Application Virtualization. URL:
<http://www.microsoft.com/windows/enterprise/products/app-virtualization.aspx>
As seen in November 2009
- [81] Richardson, T.; Stafford-Fraser, Q.; Wood, K.R.; Hopper, A., “Virtual network computing”, *Internet Computing, IEEE* , vol.2, no.1, pp.33-38, Jan/Feb 1998
- [82] uVNC *Remote Control Software for all*
URL: <http://www.uvnc.com/>
As seen in November 2009
- [83] Brambilla, M.; Origgi, A., “MVC-Webflow: An AJAX Tool for Online Modeling of MVC-2 Web Applications”, *Web Engineering, 2008. ICWE '08. Eighth International Conference on* , pp.344-349, 2008

- [84] Techinline Remote Desktop
URL: http://www.techinline.com/?gclid=CNKh4f_19JUCFRKIXwod3HzzMA
As seen in November 2009
- [85] Sun Virtual Desktop Infrastructure Software
URL: <http://www.sun.com/software/vdi/index.jsp>
As seen in November 2009
- [86] Citrix. URL: <http://www.citrix.com/lang/English/home.asp>
As seen in November 2009
- [87] Product Overview: GoToMyPC
URL: https://www.gotomypc.com/en_US/forYou.tmpl
As seen in November 2009
- [88] Citrix GoToMeeting
URL: https://www2.gotomeeting.com/en_US/pre/productOverview.tmpl
As seen in November 2009
- [89] Citrix GoToAssist
URL: http://www.gotoassist.com/en_US/technology.tmpl
As seen in November 2009
- [90] Citrix XenDesktop
URL: <http://www.citrix.com/English/ps2/products/product.asp?contentID=163057>
As seen in November 2009
- [91] An in-depth look at Citrix XenApp
URL: <http://www.citrix.com/English/ps2/products/feature.asp?contentID=1684340>
As seen in November 2009

- [92] Leff, A.; Rayfield, J.T., “Web-application development using the Model/View/Controller design pattern”, *Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International* , vol., no., pp.118-127, 2001.
- [93] Selfa, D.M.; Carrillo, M.; Del Rocio Boone, M., “A Database and Web Application Based on MVC Architecture”, *Electronics, Communications and Computers, 2006. CONIELECOMP 2006. 16th International Conference on* , vol., no., pp. 48-48, 27-01 Feb. 2006
- [94] Yonglei Tao, “Component- vs. application-level MVC architecture”, *Frontiers in Education, 2002. FIE 2002. 32nd Annual* , vol.1, 2002
- [95] Goodrich, Michael, and Roberto Tamassia. *Data Structures and Algorithms in Java*. New York: John Wiley, 2001. ISBN: 0471383678
- [96] Taivalsaari, A. 1996. On the notion of inheritance. *ACM Comput. Surv.* 28, 3 (Sep. 1996), 438-479.
- [97] Wilkens, L. 2002. Objects with prototype-based mechanisms. *J. Comput. Small Coll.* 17, 3 (Feb. 2002), 131-140.
- [98] Nystrom, N., Chong, S., and Myers, A. C. 2004. “Scalable extensibility via nested inheritance”, *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Vancouver, BC, Canada, October 24 - 28, 2004). OOPSLA '04. ACM, New York, NY, 99-115.
- [99] Rice University, Computer Science Department: *The Strategy Design Pattern*
URL: <http://www.exciton.cs.rice.edu/>
As seen in November 2009
- [100] HP Online Support Resources: HP Remote Support Pack
URL: <http://h20219.www2.hp.com/services/cache/452199-0-0-225-121.html>
As seen in November 2009

- [101] Solution Powered by Mitel, *Sun Ray Unified IP client*
URL: <http://www.mitel.com/resources/Sun%20Ray%20Unified%20IP%20Client.pdf>
- [102] A. Imine. “Decentralized concurrency control for real-time collaborative editors”, *NOTERE '08: Proceedings of the 8th international conference on New technologies in distributed systems*, pp. 1-9, New York, NY, USA, 2008. ACM.
- [103] Burns, Alan et.al. *Real-Time Systems and Programming Languages*. Upper Saddle River: Pearson Education, 2001. ISBN: 0201729881
- [104] Alessandro Alinone. *Changing the web paradigm - moving from traditional web applications to streaming-ajax*. (2005)
URL: [http://www.lightstreamer.com/Lightstreamer Paradigm.pdf](http://www.lightstreamer.com/Lightstreamer%20Paradigm.pdf)
As seen in November 2009
- [105] Microsoft TechNet: *NetMeeting and Internet Communication*
URL: <http://technet.microsoft.com/en-us/library/cc773353.aspx>
As seen in November 2009
- [106] Windows Help and How-to: *What is Windows Meeting Space?*
URL: <http://windowshelp.microsoft.com/Windows/en-US/Help/bb91b26f-7a9b-40d3-b397-b3c1cfac94411033.mspx>
As seen in November 2009
- [107] Captain's Universe: *AJAX/Javascript Browser Operating System*
URL: <http://www.captain.at/howto-ajax-browser-operating-system.php>
As seen in November 2009
- [108] PC World: *Hackers expand massive IFRAME attack to prime sites*
URL: http://www.pcworld.idg.com.au/article/210515/hackers_expand_massive_iframe_attack_prime_sites?fp=2&fpid=1
As seen in November 2009

- [109] Centre d'Expertise Gouvernemental de Réponse et de Traitement des Attaques informatiques. URL: <http://www.certa.ssi.gouv.fr/site/CERTA-2008-INF-001/>
As seen in November 2009
- [110] Google-mobwrite: *Real-time Synchronization and Collaboration Service*
URL: <http://code.google.com/p/google-mobwrite/>
As seen in November 2009
- [111] Theory - google-mobwrite - *How MobWrite works*.
<http://code.google.com/p/google-mobwrite/wiki/Theory>
As seen in November 2009
- [112] Vancea, A.; Grossniklaus, M.; Norrie, M.C., “Database-Driven Web Mashups”,
Web Engineering, 2008. ICWE '08. Eighth International Conference on, vol., no.,
pp.162-174, 14-18 July 2008
- [113] Makki, S.K. and Sangtani, J., “Data Mashups & Their Applications in
Enterprises.” Univ. of Toledo, Toledo, OH. In: Third International Conference on
Internet and Web Applications and Services, Athens, June 2008
ISBN: 978-0-7695-3163-2
- [114] Jin Yu; Benatallah, B.; Casati, F.; Daniel, F., “Understanding Mashup
Development”, *Internet Computing, IEEE*, vol.12, no.5, pp.44-52, Sept.-Oct.
2008
- [115] URL: Douglas Crockford: *Private Members in JavaScript*
<http://www.crockford.com/javascript/private.html>
As seen in November 2009
- [116] EU Dock. URL: <http://eudock.jules.it/index-eudock2.0.php>
As seen in November 2009
- [117] Orfali, Robert, and Dan Harkey. *Client/Server Programming with Java and
Corba*. New York: J. Wiley & Sons, 1998. ISBN: 047124578x,

- [118] Kiciman, E. and Livshits, B. 2007. "AjaxScope: a platform for remotely monitoring the client-side behavior of web 2.0 applications." *SIGOPS Oper. Syst. Rev.* vol. 41, no. 6 (Oct. 2007), pp. 17-30
- [119] OpenLayers: *Free Maps for the Web*
URL:<http://openlayers.org/>
As seen in November 2009
- [120] Chuang, T., Kuo, Y. S., and Wang, C. 1998. "Non-intrusive object introspection in C++: architecture and application", *Proceedings of the 20th international Conference on Software Engineering* (Kyoto, Japan, April 19 - 25, 1998). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 312-321.
- [121] Deitel, Harvey, and Paul Deitel. *C++ How to Program*. Englewood Cliffs: Prentice Hall, 2001. ISBN: 0130895717
- [122] Direct Web Remoting: Reverse AJAX
URL: <http://directwebremoting.org/dwr/reverse-ajax>
As seen in November 2009
- [123] Okika, J.C.; Ravn, A.P., "Classification of SOA Contract Specification Languages", *ICWS '08. IEEE International Conference on*, pp.433-440, 23-26 Sept. 2008
- [124] OMG: *ORB Basics*. URL: http://www.omg.org/gettingstarted/orb_basics.htm
As seen in November 2009
- [125] JavaScript Menu. URL: http://www.softcomplex.com/products/tigra_menu/
As seen in November 2009
- [126] Two tools bring Ajax to Eclipse's Ajax Toolkit Framework
<http://www.ibm.com/developerworks/opensource/library/os-ecl-ajax/>
As seen in November 2009

- [127] Mozilla Developer: *Cross-Site XMLHttpRequest*
URL: https://developer.mozilla.org/en/Cross-Site_XMLHttpRequest
As seen in November 2009
- [128] W3C *Cross-Origin Resource Sharing*
URL: <http://www.w3.org/TR/access-control/>
As seen in November 2009
- [129] Prototype is a JavaScript Framework
URL: <http://www.prototypejs.org/>
As seen in November 2009
- [130] Pellegrini, N-C, “Dynamic reconfiguration of Corba-based applications”,
Proceedings Technology of Object-Oriented Languages and Systems (TOOLS 29)
IEEE Computer Society, 1999, pp.329-40. Los Alamitos, CA, USA.
- [131] Purtilo, J.M., and Hofmeister, C., “Dynamic reconfiguration of distributed
programs”, *Proceedings of the 11th International Conference on Distributed
Computing Systems*, pp. 560-571, 1991.
- [132] Schmidt RW, “Dynamically extensible objects in a class-based language”,
*Proceedings of Technology of Object-Oriented Languages and Systems TOOLS-
23, IEEE Computer Society*, Los Alamitos, CA, USA, pp. 294-305, 1998.
- [133] Skarra, A.H., and Zdonik, S.B., “Type Evolution in an Object-Oriented
Database”, *Research Directions in Object-Oriented Programming, MIT Press*,
1988.
- [134] Monk, S.R., and Sommerville, I., “Schema Evolution in OODBs Using Class
Versioning”, *SIGMOD RECORD*, vol. 22 no.3, September 1993.
- [135] Evans, H., and Dickman, P., “DRASTIC: A Run-Time Architecture for
Evolving, Distributed, Persistent Systems”, *ECOOP'97 Proceedings*, 1997.

- [136] Latest SOAP versions
URL <http://www.w3.org/TR/soap/>
As seen in November 2009
- [137] Downes, S. 2008. *Ten web 2.0 things you can do in ten minutes to be a more successful e-learning professional*. eLearn 2008, 3 (Mar. 2008), 1-1.
- [138] Lenk, A., Klems, M., Nimis, J., Tai, S., and Sandholm, T. 2009. "What's inside the Cloud? An architectural map of the Cloud landscape", *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing - Volume 00 (May 23 - 23, 2009)*. International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 23-31.
- [139] Dearman, D. and Pierce, J. S. 2008. "It's on my other computer!: computing with multiple devices" *Proceeding of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy, April 05 - 10, 2008). CHI '08. ACM, New York, NY, pp.767-776.
- [140] Wright, J. and Dietrich, J. 2008. "Survey of existing languages to model interactive web applications" *Proceedings of the Fifth on Asia-Pacific Conference on Conceptual Modelling – Vol, 79* (Wollongong, NSW, Australia, January 01 - 01, 2008).
- [141] A. Hinze and M. Kirchberg, Eds. *Conferences in Research and Practice in Information Technology Series, vol. 325*. Australian Computer Society, Darlinghurst, Australia, 113-123.
- [142] Bernick, P.; Palazzolo, J., "Validating Wikis (and other Collaborative Writing Tools)", *Professional Communication Conference, 2007. IPCC 2007. IEEE International*, Oct. 2007