

Resource Sharing Platform Architecture for an Information Product Factory

by

Abel Sanchez

Master of Science (1998)
Massachusetts Institute of Technology

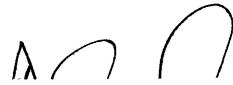
Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in the Field of
Information Technology

at the

Massachusetts Institute of Technology
[September 2003]
August 2003

© 2003 Massachusetts Institute of Technology
All rights reserved



Signature of Author

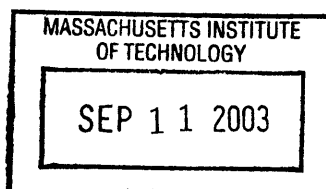
Department of Civil and Environmental Engineering
August 15, 2003

Certified by

 John R. Williams
Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by

 Heidi Nepf
Chairman, Department Committee on Graduate Students



BARKER

Resource Sharing Platform Architecture for an Information Product Factory

by

Abel Sanchez

Submitted to the Department of Civil and Environmental Engineering
on August 15, 2003 in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy in the Field of
Information Technology

ABSTRACT

Efforts to share resources in collaborative pursuits are hindered by differing data representations, redundant applications, and software incompatibilities. Members of a collaborative effort often span different computational environments and the heterogeneity of contexts disrupts software interoperability.

Sharing computational resources has become the focus of many research efforts. Efforts in the 1980s led to the Component Object Model (COM) [Williams, 1998a, 1998b, 1990] and the Common Object Request Broker (CORBA) architectures [Offall et al 1996; OMG, 2003]. In the 1990s both technologies were extended for network support. In recent years, Ian Foster, has phrased the distributed computing problem in terms of sharing computational resources. The grid problem is, “*coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*”, “*to support collaborative problem solving in industry, science, and engineering in a data rich environment*”. [Foster et al, 2001]

This work presents a new resource sharing platform architecture for information products that leverages the lessons learned from physical product platforms, the concepts of web services, and grid computing. The platform developed in this thesis integrates the contributions of these three areas into a system that is shown to be more efficient and effective at producing software products.

Thesis Supervisor: John R. Williams

Title: Professor of Civil and Environmental Engineering and Engineering Systems Division

Acknowledgments

Professor John R. Williams for his endless curiosity, sense of wonder, contagious excitement, knowledge, and guidance. For always taking time out of his busy schedule to discuss my projects. For always making sure I had the hardware and software I needed. It has been a pleasure having him as my advisor.

My wife, my better half (media naranja), for her undevoted support through out the years. For being my best friend and always being in my cheering section. Thank you for knowledge, understanding, and love.

My father, who always believed that education is the most important gift a person can receive and who instilled in me the value of continuous learning. To my parents, I owe what I am today and what I can achieve in the future to them.

My friends, for making MIT fun. The IESL bunch for being more than lab mates. For their support in IESL projects. To Hai for his support in IESL projects. Hai's sense of responsibility and professionalism always raised the bar.

Joan MacCusker, for her help and advice on administrative issues.

Lotus Development Corporation, IBM, Microsoft Corporation, Ford Motor Company, the Systems Design and Management program, and Sandia National Laboratories for providing funding for my research.

CHAPTER 1	PREFACE.....	11
1.1	INTRODUCTION.....	11
1.2	CHAPTER GUIDE.....	11
CHAPTER 2	PROBLEM OVERVIEW.....	14
2.1	GOALS.....	15
2.2	METHODOLOGY.....	16
2.3	DESIGN PRINCIPLES.....	18
2.4	GRID COMPUTING	18
2.5	XML AND HTTP ARCHITECTURE	19
2.5.1	<i>RPC and ORPC</i>	19
2.5.2	<i>Problems with ORPC</i>	20
2.5.3	<i>HTTP, XML, and SOAP</i>	21
2.5.4	<i>Email Service Sample</i>	22
2.6	APPLICATION DEFINITION LAYER	30
2.7	PRODUCT PLATFORMS.....	31
2.8	PROBLEM STATEMENT	32
CHAPTER 3	COURSE SUPPORT AS AN EXAMPLE OF AN INFORMATION PRODUCT ...	33
3.1	INTRODUCTION.....	33
3.2	COMMAND GOALS	34
3.3	CENTRALIZED MODEL.....	35
3.4	DECENTRALIZED MODEL	35
3.5	PROPOSED MODEL	35
3.6	ARCHITECTURE	36
3.7	USERS	37
3.8	WEB MODEL	38
3.8.1	<i>Centralized Model</i>	38
3.8.2	<i>Decentralized Model</i>	39
3.8.3	<i>Proposed Model</i>	39
3.9	SAMPLE COMMAND DATA.....	40
3.10	COMMAND DESIGN GUIDELINES	41
3.11	OBSERVATIONS	41
CHAPTER 4	GRIDS.....	43
4.1	INTRODUCTION.....	43
4.2	GALACTIC NETWORK	43

4.3	GRID TYPES.....	47
4.4	THE GRID	49
4.5	WHY IS THE GRID IMPORTANT?	51
4.6	THE GRID ARCHITECTURE SOLUTION.....	53
4.7	GRID TECHNOLOGY.....	55
CHAPTER 5 INTER-ORGANIZATION COLLABORATION.....		56
5.1	COLLABORATION ORGANIZATIONS SAMPLES.....	56
5.2	ORGANIZATION DIFFERENCES.....	57
5.3	SOFTWARE & DATA	57
5.4	PERMISSIONS.....	58
5.5	SCALABILITY	59
5.6	VIRTUAL ORGANIZATIONS	59
CHAPTER 6 WEB ARCHITECTURES.....		64
6.1	INTRODUCTION.....	64
6.2	WEB-LIKE DESIGN	64
6.3	XML	65
6.4	XML BASED PROGRAMMING.....	66
6.5	SERVICE ORIENTED PROGRAMMING AND ARCHITECTURE	69
6.5.1	<i>Background</i>	69
6.5.2	<i>OOP</i>	70
6.5.3	<i>OOP Disadvantages</i>	70
6.5.4	<i>The SOA Alternative</i>	71
6.5.5	<i>OOP SOA Differences</i>	72
6.5.6	<i>SOA Observations</i>	73
6.5.7	<i>SOA Fabric</i>	73
6.6	PROTOCOL INDEPENDENCE AND SOA	75
6.7	BUSINESS PROCESSES	75
CHAPTER 7 XML ARCHITECTURE.....		76
7.1	SCHEMA DEFINITION LANGUAGE	76
7.2	SERIALIZATION	80
7.3	OBJECT TECHNOLOGIES	82
7.3.1	<i>Object Technologies</i>	82
7.3.2	<i>HTTP Architecture Sample</i>	85
7.3.3	<i>HTTP Architectures</i>	86
7.3.4	<i>SOAP</i>	90

7.3.5	WSDL.....	91
CHAPTER 8	DISCOVERY SERVICES.....	96
8.1	DATA MODEL.....	98
CHAPTER 9	PRODUCT PLATFORMS.....	107
9.1	ARCHITECTURE	108
9.2	BLACK AND DECKER - ARCHITECTURE RENEWAL EXAMPLE	109
9.3	PRODUCT PLATFORM	110
9.4	PROCESS PLATFORM.....	111
9.5	PRODUCT FAMILIES	114
9.6	PRODUCT PLATFORM EVOLUTION	114
9.7	TECHNOLOGICAL LEVERAGE.....	115
9.8	MARKET LEVERAGE.....	116
9.9	DISTINCTIONS FROM PHYSICAL PRODUCTS	117
9.10	BARRIERS FOR COMPETITION.....	118
9.11	PLATFORM CONCLUSIONS	119
CHAPTER 10	PORTAL FACTORY	121
10.1	INTRODUCTION.....	121
10.2	THE PORTAL INFRASTRUCTURE.....	122
10.3	USERS	124
10.4	RENDERING	127
10.5	REFLECTION	130
10.6	PORTAL DATA MODEL	131
10.7	LANGUAGE AND CULTURE	133
10.8	PORTAL ARCHITECTURE.....	136
10.9	GOING MULTI-PORTAL	140
10.10	PORTAL TYPES	141
10.11	PORTAL MANAGERS.....	143
10.12	PORTAL FACTORY	144
CHAPTER 11	METRICS.....	148
11.1	EFFICIENCY	151
11.2	EFFECTIVENESS	153
11.3	DECLINING PLATFORM EFFECTIVENESS.....	154
11.4	OBSERVATIONS	154
CHAPTER 12	FUTURE WORK AND CONTRIBUTIONS.....	156

12.1	FUTURE WORK.....	156
12.1.1	XML Transactions.....	156
12.1.2	Workflow.....	157
12.1.3	Data Mining.....	160
12.2	CONTRIBUTIONS.....	160
12.2.1	Grid Extension and Implementation.....	160
12.2.2	Information Product Platform.....	161
12.2.3	Service Oriented Architecture.....	161
12.2.4	Lower cost for information products.....	161
 APPENDIX 1 COMMAND DATA		162
A.1.1	DOCUMENT COUNTS PER CATEGORY – SYSTEM WIDE.....	162
A.1.2	DIRECTORY SIZES	170
A.1.3	FILE TYPES.....	173
A.1.3.1	File Type Count per Course.....	173
A.1.3.2	File Type Count Totals for the Entire System	192
 APPENDIX 2 DESIGNING FOR COMMUNICATION: LAYOUT, STRUCTURE, AND NAVIGATION. 196		
A.2.1	STRUCTURAL GUIDELINES	196
A.2.2	EXPLICATE STRUCTURE	197
A.2.3	EMPHASIZE CENTRAL STRUCTURE.....	197
A.2.4	GRAPHICS DESIGN.....	198
A.2.5	TEXT DESIGN	200
A.2.6	THE GRAPHICAL USER INTERFACE.....	201
A.2.6.1	The Principle of User Correctness.....	201
A.2.7	THE PRINCIPLE OF LEAST ASTONISHMENT.....	202
A.2.7.1	Navigational Bars	202
A.2.7.2	Clarity	203
A.2.7.3	Completeness	203
A.2.7.4	Consistency.....	204
A.2.7.5	Robustness.....	204
A.2.7.6	User-Centeredness.....	204
 APPENDIX 3 TECHNOLOGY AND EDUCATION.....		206
A.3.1	INFORMATION BANKS	206

A.3.2	SYMBOL PADS	206
A.3.3	CONSTRUCTION KITS	207
A.3.4	PHENOMENARIA	207
A.3.5	TASK MANAGERS	207
A.3.6	CAVEATS FOR APPLICATIONS OF INFORMATION TECHNOLOGY	208
A.3.7	INFORMATION TECHNOLOGY	208
A.3.8	EDUCATIONAL MODELS	209
A.3.8.1	<i>Theory One</i>	209
A.3.8.2	<i>Tutorial Cycle</i>	210
A.3.8.3	<i>Present Information</i>	210
A.3.8.4	<i>Elicit Student Action Toward Goals</i>	210
A.3.8.5	<i>Asses Students Action</i>	210
A.3.8.6	<i>Provide Feedback to the Student</i>	211
A.3.8.7	<i>Offer Strategic Guidance to the Student</i>	211
A.3.8.8	<i>Manage and Motivate the Process</i>	211
A.3.9	DESIGNING CLEAR INFORMATION	211
A.3.10	CONTENT GUIDELINES.....	211
A.3.10.1	<i>The Learner</i>	211
A.3.10.2	<i>The Experts</i>	212
A.3.10.3	<i>Structural Guidelines</i>	213
A.3.11	EXPLICATE STRUCTURE.....	214
A.3.12	EMPHASIZE CENTRAL STRUCTURE	215
APPENDIX 4	WSDL GENERIC REGISTRATION SAMPLE.....	215
APPENDIX 5	FEDERATION.....	229

FIGURE 2-1 ORPC [BOX, 2000]	20
FIGURE 2-2 ORPC REQUESTS [BOX, 2000]	20
FIGURE 2-3 SOAP MAPPED TO ORPC [BOX, 2000]	22
FIGURE 3-1 MODELS OF TEACHING AND WEB USE	38
FIGURE 4-1 COMPUTER LINKING	44
FIGURE 4-2 DOCUMENT LINKING	45
FIGURE 4-3 COMPUTING RESOURCES LINKING.....	46
FIGURE 5-1 COMPUTATIONAL RESOURCES.....	61
FIGURE 5-2 FLEXIBLE SHARING RELATIONSHIPS.....	62
FIGURE 5-3 DIVERSE CONTROL LEVELS	63
FIGURE 5-4 DIVERSE USAGE MODES.....	63
FIGURE 6-1 XML FAMILY OF SPECIFICATIONS [SALL, 2002]	65
FIGURE 6-2 DIVERSE COMPUTING ENVIRONMENT	68
FIGURE 7-1 SCHEMA TYPES [W3C-SCHEMA-TYPES, 2003]	78
FIGURE 7-2 COMPONENT OBJECT MODEL (COM).....	83
FIGURE 7-3 CORBA	84
FIGURE 7-4 RMI.....	84
FIGURE 7-5 SAMPLE HTTP ARCHITECTURE	85
FIGURE 7-6 SOAP	88
FIGURE 7-7 WEB SERVICE.....	89
FIGURE 7-8 SOAP.....	91
FIGURE 7-9 WSDL	92
FIGURE 7-10 WSDL DESCRIPTION	93
FIGURE 7-11 WSDL SAMPLE	94
FIGURE 8-1 UDDI BASIC DATA MODEL.....	98
FIGURE 8-2 tMODELS	100
FIGURE 8-3 UDDI DATA MODEL	101
FIGURE 8-4 UDDI EXTENDED SAMPLE (SOURCE MSDN.MICROSOFT.COM)	102
FIGURE 10-1 USERS DATA MODEL.....	125
FIGURE 10-2 RENDERING PANES	128
FIGURE 10-3 LATE BINDING USING REFLECTION	131
FIGURE 10-4 PORTAL DATA MODEL	131
FIGURE 10-5 MODEL PAGES MAPPING	132
FIGURE 10-6 MAPPING SAMPLE	133
FIGURE 10-7 CULTURES AND LANGUAGES	134
FIGURE 10-8 CULTURE LAYERS	135

FIGURE 10-9 LANGUAGE EDITOR	136
FIGURE 10-10 HIGH LEVEL PORTAL ARCHITECTURE	136
FIGURE 10-11 VIRTUAL FABRIC	137
FIGURE 10-12 RENDERING	139
FIGURE 10-13 PORTAL COMPOSITION	140
FIGURE 10-14 PORTAL RENDERING.....	140
FIGURE 10-15 PORTAL AUTOMATION CYCLE.....	142
FIGURE 10-16 PEER SERVERS IN A SERVICES GRID	143
FIGURE 10-17 PORTAL MANAGERS	144
FIGURE 10-18 PORTAL TYPES	145
FIGURE 10-19 PORTAL INSTANCES.....	145
FIGURE 10-20 MULTI-PORTAL GRID	146
FIGURE 10-21 REFERENCING THE GLOBAL GRID	147
FIGURE 10-22 SOURCING THE GLOBAL GRID	147
FIGURE 12-1 WEB SERVICE SPECIFICATIONS [IBM-WS, 2003].....	156
FIGURE 12-2 WORKFLOW DIMENSIONS.....	158
APPENDIX FIGURE 1 LECTURE NOTES	162
APPENDIX FIGURE 2 HOMEWORK.....	163
APPENDIX FIGURE 3 DISCUSSIONS	164
APPENDIX FIGURE 4 READINGS.....	165
APPENDIX FIGURE 5 ASSIGNMENTS.....	166
APPENDIX FIGURE 6 ABOUT COURSE	167
APPENDIX FIGURE 7 SOLUTIONS	168
APPENDIX FIGURE 8 VIDEO	169
APPENDIX FIGURE 9 DIRECTORY SIZES	172
APPENDIX FIGURE 10 FILE TYPE.....	194
APPENDIX FIGURE 11 NAVIGATIONAL STRUCTURES.....	202
APPENDIX FIGURE 12 NAVIGATIONAL AIDES.....	203
APPENDIX FIGURE 13 TUTORIAL CYCLE	210

Chapter 1 Preface

This chapter begins with a brief introduction followed by a chapter guide.

1.1 Introduction

Sharing computational and information resources has become the focus of many research efforts. Efforts in the 1980s led to the Component Object Model (COM) [Williams, 1998a, 1998b, 1990] and the Common Object Request Broker (CORBA) architectures [Offall et al 1996; OMG, 2003]. In the 1990s both technologies were extended for network support. In recent years, Ian Foster, has phrased the distributed computing problem in terms of sharing computational resources. The grid problem is, “*coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*”, “*to support collaborative problem solving in industry, science, and engineering in a data rich environment*”. [Foster et al, 2001]

This work presents a new resource sharing platform architecture for information products; leveraging the lessons learned from physical product platforms; leveraging the concepts of grid computing and adding a binding (presentation) layer. Several bodies of research help address this problem: grid computing, service oriented architectures, and physical product platform research. The problem is addressed with the contributions of these three areas and a prototype architecture is proposed and implemented.

1.2 Chapter Guide

This work starts by introducing the problem in *Chapter 2, Problem Overview*. It presents the problem, the goals of this work, and the methodology. It also overviews the contributing fields of research on which this thesis is based and details a sample XML service.

Chapter 2 is followed by an early sample of a software system for collaboration. *Chapter 3 Course Support as an Example of an Information Product*. This chapter presents the

Course Management and Delivery system research findings that illustrate the problems posed in writing collaboration software. It details the functionality necessary to facilitate course content delivery and class administration via the web, enable collaboration between students, foster communication between student and instructor, empower users to create their own content, and facilitate dissemination and institutionalization of web-based teaching tools.

Teams, in academia, come together to find solutions to contemporary and classic problems. The teams' objectives and character are varied, ranging from course work projects to social activities.

The collaboration scenario is not limited to education. Observations made on the software systems Command and TeamSpace lead the author to the Grid problem definition.

Chapter 4 Grids. This chapter discusses early grid-like ideas, current incarnations of grid thought, .grid types, the importance of the grid, and the grid architecture solution.

Central to the grid problem is the sharing of resources. In each case, a number of distrustful participants with varying degrees of prior relationship want to share resources in order to perform some task.

Chapter 5 Inter-Organization Collaboration. This chapter discusses inter-organization collaboration. It addresses virtual organizations and the need to share computational resources, to have flexible sharing relationships, to have diverse control levels, and diverse usage modes.

Chapter 6 Web Architectures. This chapter introduces web inspired architectures that use XML based programming (XBP) as the glue to enable interoperability between heterogeneous independent distributed computing platforms; a service oriented programming model and a mesh of service oriented architectures (SOA).

Chapter 7 XML Architecture. This chapter introduces XML architectures and technologies. It contrasts SOAP, web services, and WSDL with COM, CORBA, and RMI.

Chapter 8 Discovery Services. This chapter introduces the Universal Description Discovery and Integration (UDDI) protocol. UDDI is used for description and discovery in the portal grid. Samples of the UDDI XML documents are provided.

Chapter 9 Product Platform. This chapter discusses product platforms, product families, and derivate products. It addresses the design and manufacturing of information products. It investigates how information technology can be used to support the creation of new products, create the manufacturing platforms, and modularize the product design. It proposes an architecture for information products and it will addresses the strategic and technical implications of the architecture.

The concept of Grid Computing is powerful but more is needed: a home for our services, a way to define applications (portal types) quickly and inexpensively, a presentation layer to the pool of computational resources, and a stateless version of the grid's abstract framework for communication. Based on these needs the design of a Portal Factory can be developed.

Chapter 10 Portal Factory. The concept of a portal factory is that of a core platform of software from which individual products can be developed. The portal infrastructure data model is composed of portals, component instances, component definitions, users, and roles. The portal infrastructure also contains a rendering component, a module loader, a portal loader, the portal context, satellite assemblies. In this chapter the infrastructure component, data models, and designed are discussed.

Chapter 11 Metrics. This section proposes metrics to assess the efficiency and effectiveness of the research, development, and lifetime management of information product platforms.

Chapter 12 Future Work and Contributions. This chapter introduces potential future directions for work in the portal factory. The most direct paths of future work are transaction support layers for the XML architecture and workflow management systems. Other venues are mining the vast collected data of the command system. Similarly, it presents contributions: grid extensions and implementation, information product platform, and service oriented architecture.

Appendix 1, lists extensive data collected from the Command software system deployed at MIT. *Appendix 2 and 3* present the design guidelines used for the Command system.

Appendix 2. Designing for Communication: Layout, Structure, and Navigation. This chapter presents guidelines for the design of site structure. It also goes on to discuss graphic and text design in the context of graphical user interfaces.

Appendix 3. Technology in Education. An overview of technology in education.

Appendix 4. WSDL Generic Registration Sample. Web services description language sample for a generic registration service.

Appendix 5 Federation. Federation is a term used for grouping in large decentralized systems. This appendix discusses federation in the context of security systems.

Chapter 2 Problem Overview

Efforts to share resources in collaborative pursuits are hindered by differing data representations, redundant applications, and software incompatibilities. As the members of a collaborative effort span computational environments the heterogeneity of contexts

disrupts sharing attempts. The problem will only worsen as network connectivity becomes ubiquitous and more resources are converted to software form.

Sharing computational resources has become the focus of many research efforts. Efforts in the 1980s led to the Component Object Model (COM) [Williams, 1998a, 1998b, 1990] and the Common Object Request Broker (CORBA) architectures [Offall et al 1996; OMG, 2003]. In the 1990s both technologies were extended for network support. In recent years, Ian Foster, has phrased the distributed computing problem in terms of sharing computational resources. The grid problem is, “*coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*”, “*to support collaborative problem solving in industry, science, and engineering in a data rich environment*”. [Foster et al, 2001]

This work presents a new resource sharing platform architecture for information products; leveraging the lessons learned from physical product platforms; leveraging the concepts of grid computing and adding a binding (presentation) layer. Several bodies of research help address this problem: grid computing, service oriented architectures, and physical product platform research. The problem is tackled with the contributions of these three areas and a XML architecture is proposed and implemented.

This chapter begins listing the goals, the methodology, and the design guidelines. It then overviews grid computing, XML architectures, product platforms, the portal factory, and ends with the problem statement.

2.1 Goals

The goals of this thesis are:

1) A grid extension - A presentation layer for the grid concept

An Application Definition Layer (ADL) for the available pool of web services.

An XML application definition of distributed XML services on the network.

2) A modular platform for collaborative work

Loosely coupled XML components that encapsulate the functionality for a service.

3) Lower cost for information products

Lower the cost, time, and efficiency of constructing new information products.

4) Framework to analyze information product platforms

A framework to study the stages in the production of information product.

5) Metrics to gage platform efficiency and effectiveness

Metrics to compare platforms and determine successful information product families.

6) Service oriented platform design

A platform based on XML contracts and messages.

2.2 Methodology

This work is based on a series of experiments, beta trials, and implementations made at the Massachusetts Institute of Technology (MIT). The initial observations were made on the efforts to provide web based course support in academe.

Five years ago many professors, groups, and departments were putting course content on the web. However, developers were programming and authoring independently. Everyone's course looked different. Everyone's content data model was different. The functionality across efforts varied according to savvy, resources, budget, and time.

Groups with a lot of time, a large budget, and computer savvy built high quality course support. However, many faculty did not have the resources needed and offered minimal or none existent web support. Even in the case of a high quality resource, there was still the issue of sharing the information product programmatically across platforms and external groups.

The authoring and programming explosion of the web happened for course support but reuse was missing. Everyone was solving the same set of problems over and over. Time

and time again groups devised independent systems for authentication/authorization, schemas, and data structures to support courses.

Five years ago, a framework to address the instructional integration of technology was designed and tested using three prototype web applications deployed in the schools of Engineering, Business, and Architecture.

The systems scaled well and became the largest course management and delivery system at MIT. The system reached over 16,000 users, over 250,000 pages, and over 60,000 attached files. The most common request from users was to leverage parts of the system for collaborative efforts outside of courses. In time an extension was made for generic team collaborations. The team support system also had a wide adoption and acceptance.

The heavy use of the system flagged an unmet need in the MIT community. Web course support was indicative of a broader need to share resources. In other words, the collaboration scenario is not limited to education. The course platform observations and specially the team platform observations lead the to the same conclusion as the Grid problem definition: the need for individuals and organizations to share computational resources [Foster et al, 2001]. That is, the course platform provided a consistent and predictable service for courses. However, the platform fell short in leveraging existing services and sharing services generically.

The problem focus is the redundant work, redundant services, and the inability to leverage existing resources on the network. Examples are course sites, department sites, laboratory sites, student activities sites, virtual engineering, and remote experimentation sites. Several bodies of research help address this problem: grid computing, service oriented programming, service oriented architectures (e.g. SOA [Cabrera, 2003], OGSA [Foster et al, 2002]), and physical product platform research. The problem is tackled with the contributions of these three areas and a XML architecture is proposed and implemented. The observations were made at MIT but collaborative efforts and resource sharing are prevalent though all organizations.

2.3 Design Principles

This work follows the emerging service oriented architecture (SOA) and web services programming paradigm.

The design principles are modular and composable components that can work standalone or as part of an application. The services are general purpose services that are abstracted by a layer of XML to be agnostic to the execution environment. The protocols are based on HTTP, XML, and the SOAP standards. The web service layer is federated. There is not central point of administration, control, or failure.

2.4 Grid Computing

The fundamental problem of grid computing is:

- *Coordinated resource sharing and problem solving in a dynamic, multi-institutional virtual organizations.*
- *To support collaborative problem solving in industry, science, and engineering in a data rich environment.*

as defined by Ian Foster in the “*Anatomy of the Grid*” [Foster et al, 2001], where a virtual organization is a set of individuals and/or institutions defined by sharing rules.

The grid computing literature presents interoperability as the key to enable resource sharing architectures. The grid needs to support sharing relationships among arbitrary parties, accommodating new participants dynamically, across different platforms, languages, and programming environments. Without interoperability, VO applications are forced to enter into bilateral sharing agreements. Without interoperability, dynamic VO formation is all but impossible, and the types of VOs that be formed are severely limited. Interoperability is thus the central issue to be addressed. In a networked environment interoperability means common protocols.

The Grid architecture is first and foremost a protocol architecture, with protocols defining the basic mechanisms by which VO users and resources negotiate, establish, manage, and exploit sharing relationships. For an in depth discussion of the grid see the “Grids” chapter.

This work proposes a protocol architecture based on XML and HTTP. The XML and HTTP architecture addresses shortcomings of the dominant distributed object technologies of the past twenty years.

2.5 XML and HTTP Architecture

HTTP architectures have scaled like no other in history. Coupled with XML the Simple Object Access Protocol (SOAP) [W3C-SOAP, 2003] presents a new opportunity to lower the barriers of interoperability. SOAP messages are encoded using XML and uses HTTP as Remote Procedure Call (RPC) style transport.

The kernel of the architecture is XML, HTTP, and SOAP. XML parsers and HTTP processors are ubiquitous and at the time of writing there were over 50 implementations of SOAP [SOAPLite-Toolkits, 2003]. With a SOAP handler any server becomes a SOAP object request broker.

2.5.1 RPC and ORPC

In 1980s the dominant RPC protocols were Sun RPC and the Distributed Computing Environment (DCE) RPC. The most visible implementations were the Network File System (NFS) for UNIX systems for Sun and Windows NT for DCE [Box, 2000].

In the 1990s, the proliferation of Object Oriented Programming (OOP) led to Object RPC (ORPC). ORPC coupled objects with a transport. Likewise, ORPC constructed the mechanism needed to expose a handle for the remote object down into the programming language.

Request	Response
Object End Point ID (which object)	Status Code (did it work?)
Interface Identifier (which interface)	Extension Headers (what did you forget to build into the protocol?)
Method Identifier (which method)	Parameter Data (here are the out and inout parameters)
Extension Headers (what did you forget to build into the protocol)	
Parameter Data (here are the in and inout parameters)	

Figure 2-1 ORPC [Box, 2000]

Currently, the dominant ORPC protocols are the Distributed Component Object Model (DCOM) and the Component Object Request Broker Architecture's (CORBA) Internet Inter-ORB Protocol (IIOP) [Box, 2000].

2.5.2 Problems with ORPC

ORPC implementations were successful within the organization in controlled computational environments. However, both CORBA and DCOM had problems going across organizational boundaries and onto the internet.

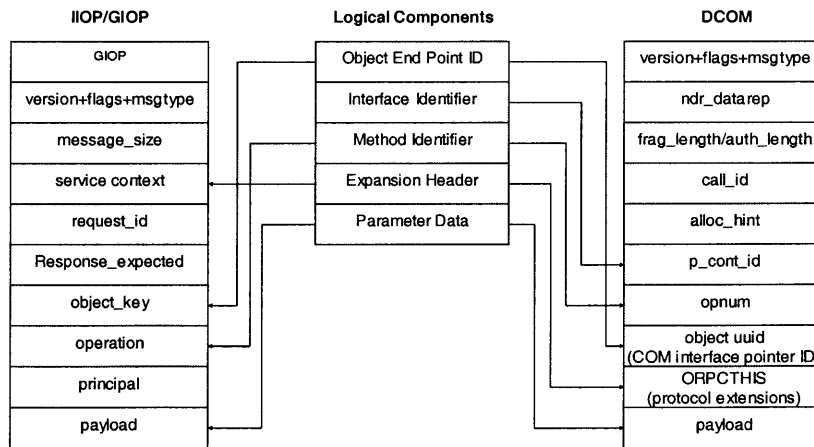


Figure 2-2 ORPC Requests [Box, 2000]

The successful implementations of ORPC were platform dependent. Although Independent Software Vendors (ISV) made adaptations to other platforms, the cost of maintenance and complexity of the solution limited the adoption rate.

Each ORPC implementation also had specific implementation issues. DCOM had scalability problems with DCE's keep alive messages on the network. The security issues as systems spanned computational environments and the required message volume restricted the deployment scenarios.

CORBA is turn had ORB problems. The number of ORBs in the market led to incompatibilities. ORBs did not interoperate. Software developed for one ORB had no guarantee to work in another ORB.

Another barrier was the technical complexity of the technologies. The complexity of the technology limited to development to senior staff and experts. The plumbing, the programming languages, and protocols restricted the number of people qualified to design applications.

2.5.3 HTTP, XML, and SOAP

The distributed architecture using HTTP, XML, and SOAP is very loosely coupled. Many parts are not specified to promote interoperability. For example, SOAP does not specify a programming model. SOAP does not specify an API or ORB. Instead with a XML parser and minimal code any HTTP server can become a SOAP ORB. The local mapping from the incoming XML message to the local object is left as implementation detail.

The missing part of HTTP is a standardized format for RPC calls, SOAP and XML answer this need. XML plays the message encoding role. XML is a platform independent, language independent, text based, and human readable representation protocol. Outgoing messages can be serialized into XML and deserialized to the local object representation upon arrival.

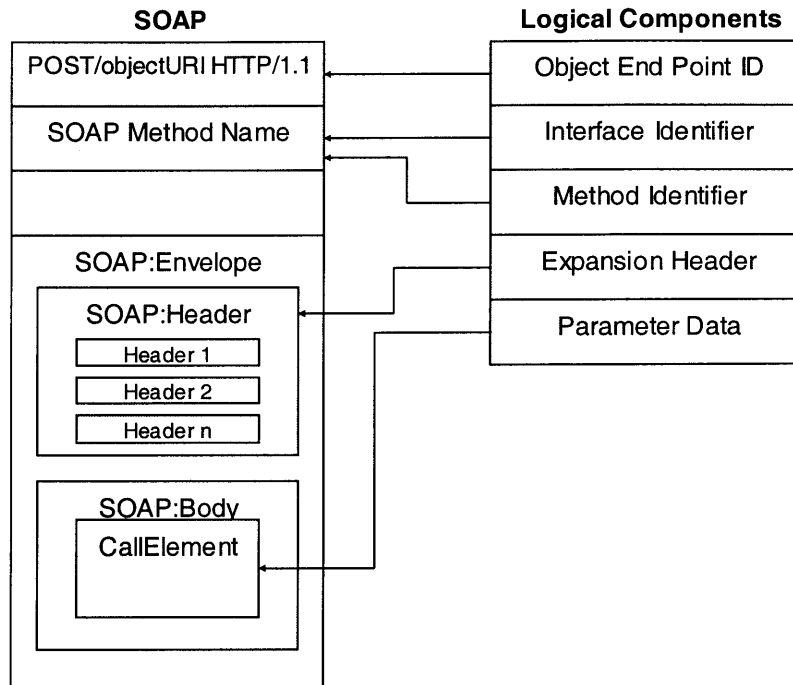


Figure 2-3 SOAP Mapped to ORPC [Box, 2000]

HTTP, XML, and SOAP is presented as better kernel for RPC: HTTP as a Better RPC and XML as a Better Network Data Representation (NDR).

Next is a detailed example using HTTP, XML, SOAP, the Web Service Description Language (WSDL), and the Universal Description Discovery and Integration (UDDI) [UDDI, 2003].

2.5.4 Email Service Sample

The first step is to write the service – edited for clarity. The email service is a simple service with one method. The “Send” method sends an email from the consuming client. The message is passed in the “Message” class.

```

namespace email
{
    // -----
    // class to hold the email message
    // -----

    public class Message
    {
        public String to = "";
        public String cc = "";
        public String from = "";
        public String bcc = "";
        public String body = "";
        public String subject = "";
        public String contentType = "TEXT";
    }

    // -----
    // email class with "send" method
    // -----

    public class Email : System.Web.Services.WebService
    {
        [WebMethod]
        public bool Send(Message msg)
        {
            Smtplib.Smtplib.SmtpServer = "outgoing.mit.edu";
            MailMessage mailMsg = new MailMessage();
            mailMsg.Cc = msg.cc;
            mailMsg.Body = msg.body;
            mailMsg.Bcc = msg.bcc;
            mailMsg.Subject = msg.subject;
            mailMsg.To = msg.to;
            mailMsg.From = msg.from;
            Smtplib.Send(mailMsg);
            return true;
        }
    }
}

```

Next the service needs to be described using the Web Service Description Language (WSDL) [W3C-WSDL, 2003]. Note that the "Send" method and the "Message" class are in the WSDL types.

The HTTP GET, POST, and SOAP are supported protocols in this WSDL document. SOAP encoding is document and literal as opposed to RPC.

The end points are the URL where the XML will be sent.

```

<definitions
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:s0="http://tempuri.org/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  targetNamespace="http://tempuri.org/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!--=====-->
  <!--          TYPES          -->
  <!--=====-->

  <types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">

      <s:element name="Send">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="msg" type="s0:Message" />
          </s:sequence>
        </s:complexType>
      </s:element>

      <s:complexType name="Message">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="to" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="cc" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="from" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="bcc" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="body" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="subject" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="contentType" type="s:string" />
        </s:sequence>
      </s:complexType>

      <s:element name="SendResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="SendResult" type="s:boolean" />
          </s:sequence>
        </s:complexType>
      </s:element>

    </s:schema>
  </types>

  <!--=====-->
  <!--          MESSAGES          -->
  <!--=====-->

  <message name="SendSoapIn">
    <part name="parameters" element="s0:Send" />
  </message>

  <message name="SendSoapOut">
    <part name="parameters" element="s0:SendResponse" />
  </message>

  <!--=====-->
  <!--          PORT TYPES - INTERFACE          -->
  <!--          port now called interface          -->
  <!--=====-->

  <portType name="EmailSoap">
    <operation name="Send">
      <input message="s0:SendSoapIn" />
      <output message="s0:SendSoapOut" />
    </operation>
  </portType>

```

```

</portType>

<portType name="EmailHttpGet" />

<portType name="EmailHttpPost" />

<!--=====-->
<!--          BINDING          -->
<!--=====-->

<binding name="EmailSoap" type="s0:EmailSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <operation name="Send">
    <soap:operation soapAction="http://tempuri.org/Send" style="document" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>

<binding name="EmailHttpGet" type="s0:EmailHttpGet">
  <http:binding verb="GET" />
</binding>

<binding name="EmailHttpPost" type="s0:EmailHttpPost">
  <http:binding verb="POST" />
</binding>

<!--=====-->
<!--          SERVICE AND END POINT          -->
<!--          port now called endpoint          -->
<!--=====-->

<service name="Email">
  <port name="EmailSoap" binding="s0:EmailSoap">
    <soap:address location="http://biztalk.mit.edu:81/email/email.asmx" />
  </port>
  <port name="EmailHttpGet" binding="s0:EmailHttpGet">
    <http:address location="http://biztalk.mit.edu:81/email/email.asmx" />
  </port>
  <port name="EmailHttpPost" binding="s0:EmailHttpPost">
    <http:address location="http://biztalk.mit.edu:81/email/email.asmx" />
  </port>
</service>
</definitions>

```

Next the service is entered into the Universal Description Discovery and Integration (UDDI) database.

The business entity corresponds to the provider, MIT in this case. The web service is “Notifications”. “Email” is only one type of notification. The binding template is the destination of the XML message. The tModel holds the metadata for the service. In this case, the WSDL document for the email service.

```

<UniversalDescriptionDiscoveryIntegration>

<!--=====-->
<!-- Business Entity entry - THE PROVIDER -->
<!--=====-->
<businessEntity
  businessKey="00000000-0000-0000-0000-000000000000"
  operator="Massachusetts Institute of Technology"
  authorizedName=" Abel Sanchez : 86">
  <discoveryURLs>
    <discoveryURL useType="Home Page">http://portals.mit.edu/training/</discoveryURL>
    <discoveryURL
useType="businessEntity">http://uddi.mit.edu/discovery?businessKey=00000000-0000-0000-
0000-000000000000</discoveryURL>
    </discoveryURLs>
    <name>Academic Portal</name>
    <description>Academic Portal - An instance of the Academic Portal type in the Portal
Factory</description>

  <!--=====-->
  <!-- The Notification service - THE WEB SERVICE -->
  <!--=====-->
  <businessService
    serviceKey="11111111-1111-1111-1111-111111111111"
    businessKey="00000000-0000-0000-0000-000000000000">
    <name>Notification</name>
    <description>Notification service in the portal factory</description>

    <bindingTemplates>
      <!--=====-->
      <!-- first binding: Email Service - THE ENDPOINT OR PORT -->
      <!--=====-->
      <bindingTemplate
        serviceKey="11111111-1111-1111-1111-111111111111"
        bindingKey="22222222-2222-2222-2222-222222222222">
        <description>Email</description>
        <!-- The service's endpoint URL -->
        <accessPoint URLType="http">
          http://portals.mit.edu/Email.asmx
        </accessPoint>
        <tModelInstanceDetails>
<!-- reference to the tModel describing the Email Send interface -->
          <tModelInstanceInfo
            tModelKey="uuid:55555555-5555-5555-5555-555555555555">
            <description>Email tModel</description>
            <instanceDetails>
              <description>SDK document</description>
              <overviewDoc>
                <description/>
                <overviewURL/>
              </overviewDoc>
              <instanceParams>http://portals.mit.edu/SDK</instanceParams>
            </instanceDetails>
          </tModelInstanceInfo>
        </tModelInstanceDetails>
      </bindingTemplate>

      <!-- second binding: Instant Messaging Service - THE ENDPOINT OR PORT -->
      <bindingTemplate
        serviceKey="11111111-1111-1111-1111-111111111111"
        bindingKey="33333333-3333-3333-3333-333333333333">
        <description>Instant Messaging</description>
        <!-- The service's endpoint URL -->
        <accessPoint URLType="http">
          http://portals.mit.edu/InstantMessaging.asmx
        </accessPoint>
        <tModelInstanceDetails>
<!-- reference to the tModel describing the Instant Messaging interface -->
          <tModelInstanceInfo tModelKey="uuid:44444444-4444-4444-4444-444444444444">
            <description>Instant Messaging tModel</description>
            <instanceDetails>

```

```

        <description>SDK document</description>
        <overviewDoc>
            <description/>
            <overviewURL/>
        </overviewDoc>
        <instanceParms>http://portals.mit.edu/SDK</instanceParms>
    </instanceDetails>
    </tModel InstanceInfo>
</tModel InstanceDetails>
</bindingTemplate>

</bindingTemplates>
</businessService>
</businessEntity>

<!--=====-->
<!--      tModel for the Email Service - METADATA STATEMENT      -->
<!--=====-->
<tModelDetail>
    <tModel
        tModelKey="uuid:55555555-5555-5555-5555-555555555555"
        operator="MIT"
        authorizedName="Abel Sanchez">
        <name>Email Send Service</name>
        <description>This interface is to be implemented by services that send
email.</description>
        <overviewDoc>
            <description>WSDL document</description>
            <overviewURL>http://portals.mit.edu/Email.asmx?WSDL</overviewURL>
        </overviewDoc>
        <categoryBag>
            <keyedReference
                tModelKey="uuid:01010101-0101-0101-0101-010101010101"
                keyName="Specification for a web service described in WSDL"
                keyValue="wsdlSpec"/>
            </categoryBag>
        </tModel>
    </tModelDetail>

<!-- tModel for the Instant Messaging Service -->
<tModelDetail>
    <tModel
        tModelKey="uuid:44444444-4444-4444-4444-444444444444"
        operator="MIT"
        authorizedName="Abel Sanchez">
        <name>Instant Messaging Service</name>
        <description>This interface is to be implemented by services that Instant
Messages.</description>
        <overviewDoc>
            <description>WSDL document</description>
            <overviewURL>http://portals.mit.edu/InstantMessaging.asmx?WSDL</overviewURL>
        </overviewDoc>
        <categoryBag>
            <keyedReference
                tModelKey="uuid:01010101-0101-0101-0101-010101010101"
                keyName="Specification for a web service described in WSDL"
                keyValue="wsdlSpec"/>
            </categoryBag>
        </tModel>
    </tModelDetail>
</UniversalDescriptionDiscoveryIntegration>

```

The next step is to write a proxy to consume the email service and send a message. Note that the proxy is a local representation of the email service “Send” method.

```

namespace wsclient.email {

    [System.Web.Services.WebServiceBindingAttribute(Name="EmailSoap",
    Namespace="http://tempuri.org/")]
    public class Email : System.Web.Services.Protocols.SoapHttpClientProtocol {

        /// <remarks/>
        public Email() {
            this.Url = "http://biztalk.mit.edu:81/email/email.asmx";
        }

        /// <remarks/>
        public bool Send(Message msg) {
            object[] results = this.Invoke("Send", new object[] {
                msg});
            return ((bool)(results[0]));
        }
    }

    /// <remarks/>
    [System.Xml.Serialization.XmlTypeAttribute(Namespace="http://tempuri.org/")]
    public class Message {

        /// <remarks/>
        public string to;

        /// <remarks/>
        public string cc;

        /// <remarks/>
        public string from;

        /// <remarks/>
        public string bcc;

        /// <remarks/>
        public string body;

        /// <remarks/>
        public string subject;

        /// <remarks/>
        public string contentType;
    }
}

```

The next step is to build the client to the email service. The consumer codes against the local proxy as if the method was local. Serialization and messaging is handled by the proxy.


```

namespace wsclient
{
    /// <summary>
    /// Email client class
    /// </summary>
    class EmailClient
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            Email.Message msg = new Email.Message();
            msg.to = "abel_user@hotmail.com";
            msg.from = "doval@mit.edu";
            msg.subject = "Xml is Stringy";
            msg.cc = "abel_user@hotmail.com";
            msg.body = "Xml is human readable";

            Email.Email email = new Email.Email();
            email.Send(msg);
        }
    }
}

```

When the client is executed, the message below goes on the wire. The top 8 lines are HTTP headers. In this platform, SOAP action signals the web server this is web service call.

```

<!--=====-->
<!--          MESSAGE REQUEST TO SERVICE          -->
<!--=====-->

POST /email/email.asmx HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol
1.0.3705.288)
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://tempuri.org/Send"
Content-Length: 451
Expect: 100-continue
Connection: Keep-Alive
Host: biztalk.mit.edu

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <Send xmlns="http://tempuri.org/">
      <msg>
        <to>iesl@mit.edu</to>
        <cc>abel_user@hotmail.com</cc>
        <from>doval@mit.edu</from>
        <body>Xml is human readable</body>
        <subject>Xml is Stringy</subject>
      </msg>
    </Send>
  </soap:Body>
</soap:Envelope>

```

The service returns the following response after a successful message.

```
<!--=====-->
<!--          MESSAGE RESPONSE FROM SERVICE          -->
<!--=====-->

HTTP/1.1 100 Continue
Server: Microsoft-IIS/5.0
Date: Tue, 12 Aug 2003 04:57:31 GMT

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 12 Aug 2003 04:57:31 GMT
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 332

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <SendResponse xmlns="http://tempuri.org/">
      <SendResult>true</SendResult>
    </SendResponse>
  </soap:Body>
</soap:Envelope>
```

This sample has shown the cycle from writing the service, describing it using WSDL, publishing it using UDDI, writing a proxy, consuming the proxy, and the SOAP messaging between the service and client. For more on this topic see the chapter on Xml Architecture.

2.6 Application Definition Layer

Grid computing and XML architectures are powerful but more is needed: a home for the web service pool; a way to define applications (portal types) quickly and inexpensively; a presentation layer to the pool of computational resources; a stateless version of the grid's abstract framework for communication.

This thesis presents an Application Definition Layer (ADL). The ADL is a new resource sharing platform. The ADL is a presentation layer for a web services federation, leveraging lessons from physical product platforms (see next section), leveraging the grid computing architecture to create a platform for information products - the Resource Sharing Platform Architecture for an Information Product Factory.

The design of the portal factory grid is a XML architecture for a stateless grid computing implementation. An implementation to enable reusable services and in a information products assembly line.

To see a detailed description of the ADL see the portal factory chapter. The next section discusses the contributions of physical products platform research to this work.

2.7 Product Platforms

The “platform” term refers to the basic product or process design from which derivate products can be produced. Two types of platforms are singled out: the product and process.

The product platform is the physical form of the architecture, where as the architecture is the design concept of the product. The process platform is the end-to-end process encompassing technologies, facilities, and processes.

Derivative products from a platform are called product families. Products families share a number of subsystems and technologies. The commonality leads to efficiency and effectiveness.

The portal factory uses a product platform called a repository. The repository is a directory of available web services from which information products can be created.

The process platform is a refinery. The refinery is composed of five steps: acquisition, refinement, storage/retrieval, distribution, and presentation.

For complete coverage on products platforms see the chapter on platforms, the one on metrics, and the Universal Description Discovery and Integration (UDDI).

2.8 Problem Statement

This work presents a new resource sharing platform architecture for information products; leveraging the lessons learned from physical product platforms; leveraging the concepts of grid computing and adding a binding (presentation) layer.

The motivation of this work is to address the needs of virtual organizations (VOs) which are defined as a set of individuals and/or institutions bound together by sharing rules. In each case, a number of distrustful participants with varying degrees of prior relationship want to share resources in order to perform some task.

The design of the portal factory grid is an XML architecture for a stateless grid computing implementation. The new architecture presents the following innovations:

- Lower cost for information products
- A new way to organize development
- A new way to develop and organize distributed development
- A platform for collaborative work
- Metrics to gage platform effectiveness and efficiency.

Chapter 3 Course Support as an Example of an Information Product

This chapter presents selected portions of the Course Management and Delivery system research findings. It covers the goals, implementation model, design analysis, and design foundations.

3.1 Introduction

Among the most important issues facing educators today are the instructional integration of technology and user support. These issues, coupled with the trend toward using web sites as a point of comparison for courses as well as schools, provide strong motivations for investigating how the web can be used to support and augment classroom instruction.

Five years ago, a framework to address the instructional integration of technology was developed and tested using three prototype web applications deployed in the schools of Engineering, Business, and Architecture. These courses involved faculty, teaching staff, and students, as well as professionals in industry. The prototypes sought to identify the following: (a) educational opportunities afforded by the web based on various teaching styles; (b) tasks that lent themselves well to its asynchronous, albeit ubiquitous, nature; (c) issues involved in web site preparation and management, including web authorship and student use.

Emphasis was placed then, as in this work, on developing interactive features and designing general templates or containers that could accommodate various subjects and empower both professors and students, while shielding them from complexity, to create and maintain web content without having to rely on an intermediary.

This section discusses the requirements, design, development and implementation, of web sites, as a course delivery system scales up and plans for institutionalization.

The first part of this chapter briefly details the design approach and presents observations based on data of the running system.

3.2 Command Goals

Develop 'generic' web-based models/templates that can be easily customized and extended to meet the needs of various classes and students at a distance.

Design general templates and continuously refine these to create a base set of containers that are scalable and customizable.

Facilitate dissemination and institutionalization of web-based teaching tools and ensure quality control in course module development.

Use the standard set of templates to initiate a “mass customization” effort targeted at enabling teachers to set up home pages quickly and effectively, while shielding them from complexity, and institutionalizing this practice.

Empower users to create their own content.

Users refers to professors as well as their teaching assistants, and students. One goal is to develop a model that does not rely on a central authority, such as a webmaster, to post materials. There are two benefits of this approach, namely: (a) professors need not rely on technical support at the institutional or even departmental level; and (b) they can control what is published and how it appears on the web. The latter is important because they are the domain experts and know best how to present the material. In addition, empowering the students to contribute directly to the class web site lessens the overhead involved in administering the class and gives more room for creativity.

The models considered for the production release are listed bellow.

3.3 Centralized Model

- Hardware, software and human (application development and technical support) resources maintained at a central site. Content provider needs to go through this intermediary for application design changes and possibly publication of content.
- *Advantages:* Economies of scale resulting from sharing of resources, control of standards, availability of critical mass of skills.
- *Disadvantages:* Author does not have full control over application design and publication of content, larger start-up costs (for central site), slower response/turnaround time for user requests.

3.4 Decentralized Model

- Content provider has full control over the design and content of the web site. In a fully decentralized model, the ‘course developers’ (i.e. professor, teaching staff) maintain the hardware and server software needed to keep the web site up and running. A deviation of this model has the hardware and server software maintained by a central resource, such as Athena, but leaves development of the web application and html pages to the course developers. The latter model may constrain the application design capabilities. For instance, relying on the Athena web server prevents course developers from using cgiscripts in their web sites.
- *Advantages:* Author has full control over publication and can see results immediately, no need to rely on an intermediary
- *Disadvantages:* Lack of hardware/software support and technical expertise, larger overall costs to the university, reinvention of wheels, variable standards, no synergy and integration.

3.5 Proposed Model

The templates have the following characteristics:

- Support a common set of course delivery and management tasks;
- Be easily customized and extended to meet to the unique needs of the various classes;
- Empower both the teaching staff and students to publish and edit content without the intervention of a webmaster;

- Run on a development and production environment that supports multiple platforms, is scaleable, provides industrial-strength security and database management support.

This paradigm lends itself well to prototyping, rapid application development and mass customization, and can be adapted to both the centralized and decentralized models. The use of generic templates that can be easily replicated and customized enables, but does not require, development of standards, promotion of best practices, flexibility in both application design and publication of content. Given the short lead times and limited technical resources available to most professors in developing web content, the generic toolkit described above enables them to get a jumpstart on their web site and focus on development of course material.

3.6 Architecture

The architecture of command was designed in an object oriented paradigm and well encapsulated components. However, as many software today the components all had proprietary interfaces. The data stores, the authentication and authorization model, the rendering model, the agents, the clients, and servers were all based on interfaces that were proprietary.

As the system began to gain acceptance and was deployed in more places the architecture constraints started to become apparent. One of the first was the registration agent. When the system was deployed integrating users automatically was desired when registering large numbers of students. The types of integration were different at each deployment. Some had proprietary systems, others had databases, and some had the increasingly popular directories like LDAP.

Interfacing the registration agents was done on a case by cases basis. Each institution created a custom solution that only worked for them and could not be reused unless the exact information technology infrastructure existed somewhere else.

A similar issue arose around the data stores. Each institution localized data exchange to relational database management systems (RDBMS), proprietary stores, and even flat text files with value pairs. The lack of a common data exchange format limited leveraging peer institutions work.

The programmatic agents themselves could only be called through proprietary protocols. Although the platform provided interfaces in several popular programming languages, C, C++, and Java, developers still had to interface the platform, create a session, access the platform classes, and create objects, all of which constrained customization and ultimately adoption.

3.7 Users

The target audience was students and faculty, both on-campus and remote. Consequently, users connected from commercial internet service providers (ISP) and local area networks (LAN). The typical bandwidth for these connections was 14.4 kbps – 56 kbps for ISPs, 128 kbps for ISDN¹, and 1 mbps for LANs.

As the course delivery system scaled up the audience diversity increased. The different users came from the schools of Engineering, Science, Humanities, Architecture, and Management. User needs, both students and faculty, varied across disciplines due to academic culture, teaching models, computer facilities, and the degree of computer literacy.

These classes, conducted in different schools within MIT, subscribed to different teaching models, thereby providing a representative cross-section of the academic population from which to base the functional requirements and system design. The findings are shown in the following diagram.

¹ integrated services digital network

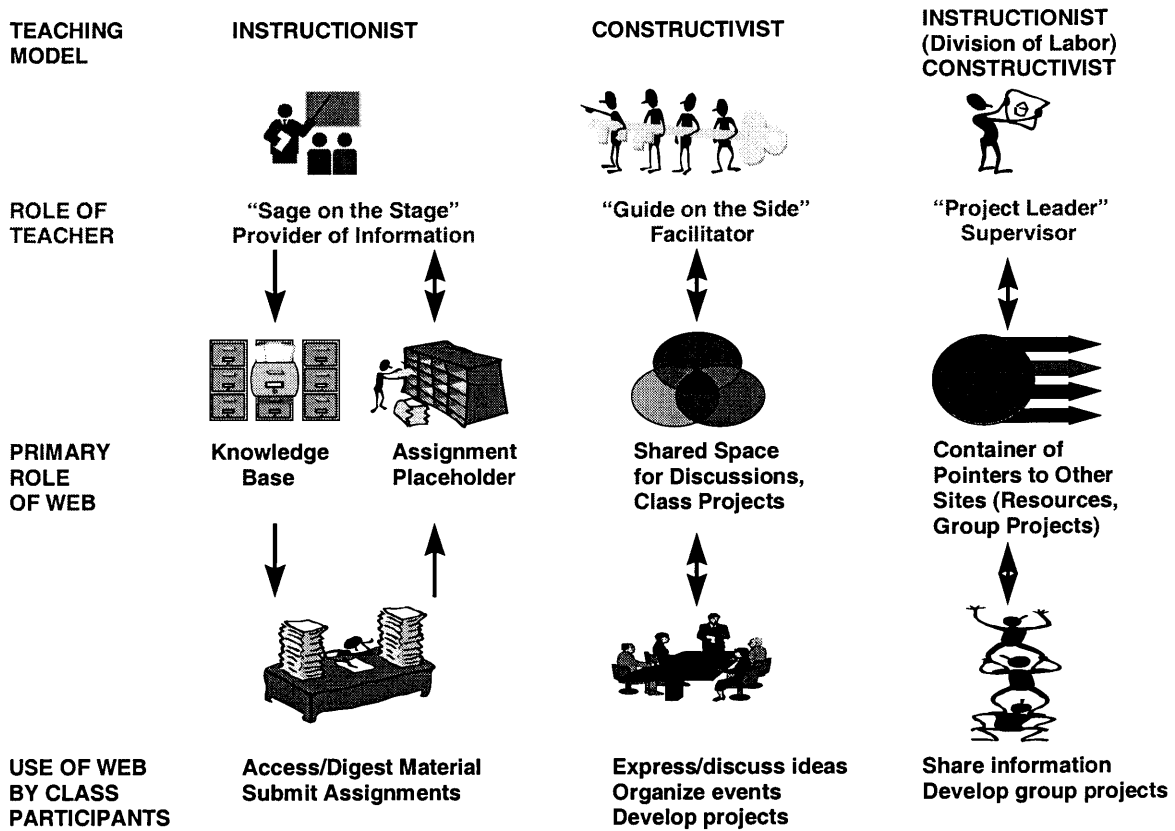


Figure 3-1 Models of Teaching and Web Use

3.8 Web Model

Currently most courses follow a centralized or decentralized web site management model [Hidalgo, 1997].

3.8.1 Centralized Model

- Hardware, software and human (application development and technical support) resources maintained at a central site. Content provider needs to go through this intermediary for application design changes and possibly publication of content.
- Advantages: Economies of scale resulting from sharing of resources, control of standards, availability of critical mass of skills.

- Disadvantages: Author does not have full control over application design and publication of content, larger start-up costs (for central site), slower response/turnaround time for user requests.

3.8.2 Decentralized Model

- Content provider has full control over the design and content of the web site. In a fully decentralized model, the ‘course developers’ (i.e. professor, teaching staff) maintain the hardware and server software needed to keep the web site up and running. A deviation of this model has the hardware and server software maintained by a central resource, such as Athena, but leaves development of the web application and html pages to the course developers. The latter model may constrain the application design capabilities. For instance, relying on the Athena web server prevents course developers from using cgi-scripts in their web sites.
- Advantages: Author has full control over publication and can see results immediately, no need to rely on an intermediary
- Disadvantages: Lack of hardware/software support and technical expertise, larger over-all costs to the university, reinvention of wheels, variable standards, no synergy and integration.

3.8.3 Proposed Model

The templates have the following characteristics:

- Support a common set of course delivery and management tasks;
- Be easily customized and extended to meet to the unique needs of the various classes;
- Empower both the teaching staff and students to publish and edit content without the intervention of a webmaster;

- Run on a development and production environment that supports multiple platforms, is scalable, provides industrial-strength security and database management support.

This paradigm lends itself well to prototyping, rapid application development and mass customization, and can be adapted to both the centralized and decentralized models. The use of generic templates that can be easily replicated and customized enables, but does not require, development of standards, promotion of best practices, flexibility in both application design and publication of content. Given the short lead times and limited technical resources available to most professors in developing web content, the generic toolkit described above enables them to get a jumpstart on their web site and focus on development of course material.

3.9 Sample COMMAND Data

At the time of writing the COMMAND had 16,199 users, 476 course instances, 84 team spaces, 2749 administrative email messages, and over 200,000 pages of content.

Sample course sizes in megabytes:

Course Size (MB)			
– 1.00	Size: 5.44 MB	– 1.041	Size: 30.91 MB
– 1.00Fall01	Size: 7.43 MB	– 1.070	Size: 0.18 MB
– 1.00Spr02	Size: 14.6 MB	– 1.103	Size: 4.05 MB
– 1.010	Size: 0.17 MB	– 1.118	Size: 59.33 MB
– 1.012	Size: 127.8 MB	– 1.118_F01	Size: 112.08 MB
– 1.013	Size: 118.41 MB	– 1.120F99	Size: 39.1 MB
– 1.030	Size: 23.32 MB	– 1.124	Size: 20.11 MB
– 1.033	Size: 104.29 MB	– 1.124_F00	Size: 6.47 MB
– 1.040	Size: 43.39 MB	– 1.125	Size: 11.62 MB
– 1.051	Size: 0.19 MB		

Table 3-1 Sample Course Sizes

Sample course file type counts:

File Types Counts – Course 1.118 (MB)	
– Document folder: Assignments	– Document folder: LectureNotes

<ul style="list-style-type: none"> - html, 1 - Document folder: By Category <ul style="list-style-type: none"> - html, 2 - Document folder: CourseMaterial <ul style="list-style-type: none"> - html, 1 - Document folder: HWByNo <ul style="list-style-type: none"> - xls, 3 - txt, 6 - pdf, 1 - zip, 14 - html, 50 - htm, 1 - gz, 2 - vsd, 3 - total section files: 80 - 	<ul style="list-style-type: none"> - ppt, 24 - pdf, 3 - zip, 1 - html, 1 - total section files: 29 - Document folder: Readings <ul style="list-style-type: none"> - xls, 2 - txt, 1 - pdf, 14 - ps, 2 - zip, 7 - exe, 2 - html, 2 - htm, 14 - total section files: 44 - Document folder: Solutions <ul style="list-style-type: none"> - html, 1 - htm, 1 - total section files: 2
---	--

Table 3-2 Sample File Type Counts

For comprehensive data on command data stores and content see the appendix.

3.10 COMMAND Design Guidelines

The appendix presents guidelines for the design of site structure. It also goes on to discuss graphic and text design in the context of graphical user interfaces.

3.11 Observations

Teams in academia come together to find solutions to contemporary and classic problems. The teams objectives and character are varied, ranging from course work projects to social activities. Regardless of the group’s objective, there are common collaboration objectives.

Sample common collaboration objectives:

- Problem identification

- Informing and knowledge exchange
- Events
- Evaluating alternatives
- Choice
- Implementing
- Review/feedback
- Coordination
- Socialization
- Motivation
- Negotiation

The collaboration scenario is not limited to education. Command and specially TeamSpace observations lead us to the same conclusion as the Grid problem definition. The need for individuals and organizations to share computational resources.

Chapter 4 Grids

4.1 Introduction

There is no universally agreed upon definition of Grid Computing. There are many systems that have grid like qualities. There are many systems that use grid technologies and there are many systems that use grid specifications. A simple Google search will return hundreds of hits on grid “like” systems.

Adding to the mix, the original and still most recognized grid technology [Globus, 2003] had undergone considerable changes in its short history. The original implementation has evolved so much there are legacy issues with early adopters. The current implementation has moved to XML architectures while the initial code base was based on the Java platform.

The grid domain presents many new ideas and possibilities. However, the unique aspect of the grid is the *web service and XML protocols*. The advantages of the architecture and protocols are discussed in the sections to follow.

4.2 Galactic Network

The early days of supercomputing saw many high performance computing centers be funded by the national science foundation and the like. As time went on the number of supercomputing centers was reduced to a few well known centers. The big machine problem received less funding and there was a focus change to distributed computing. The next challenge was to aggregate, link, and marshal distributed computers and computational resources. Many of these ideas resonate with the long standing goal to link computers and resources.

The idea of connecting computers and sharing resources is not new. Back in 1962, J.C.R Licklider from MIT introduced the idea of a “Galactic Network” a network to enable

social interactions and collaboration. By the end of the sixties, in pursuit of such goals, a contract was awarded to BBN by the Defense Advanced Research Projects Agency (DARPA) to build a packet switching network. The developments continued, in the mid seventies the work on the Transmission Control Program (TCP) began and by the eighties TCP/IP and the Domain Name System had been implemented. TCP/IP and DNS brought about a connectivity boom as record numbers joined the internet.

By the end of the eighties the number of internet hosts had surpassed 100,000 [PBS, 2003]. Widespread of local area networks and lowered technological barriers enabled the internet to flourish. However, even though many in academe and industry were connected collaboration was still complex enough that sharing resources remained for the computer savvy. Tools for the personal computer user were few and not very friendly. Text based interfaces with monochrome displays were the norm and Graphical User Interfaces (GUI) were the exception.

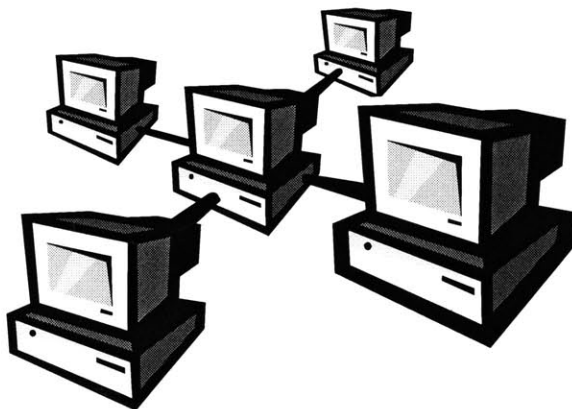


Figure 4-1 Computer Linking

The great advances in internetworking protocols of the past twenty plus years would be introduced to the mainstream by a killer application: enter the World Wide Web. The nineties started with Tim Berners Lee introduction of the World Wide Web (WWW). The web, based on the Hypertext Transfer Protocol, was the revolutionary and transcendental next step in connectivity. The HTTP step, starting out slow in the early nineties, had an exponential take off blast with the introduction of the web browser, Mosaic in late 1993.

The web browser and the simplicity of the web publishing had a take off like nothing before it in history. By the end of the nineties every country in the world was connected to the web and the technology penetrated every facet of society.

TCP/IP had connected computers, HTTP connected documents. Technically, what HTTP offered was not very different than what was possible with TCP/IP. A user could just as well pick up a document from specified a TCP/IP port given he had the right tools and knowledge of the programmatic interface. However, the exchange would require a bilateral agreement and information exchange between the client and server teams. HTTP standardized the exchange. As long as the server adhered to the protocol the client could consume documents from the standardized interface with having to talk to the service owners.

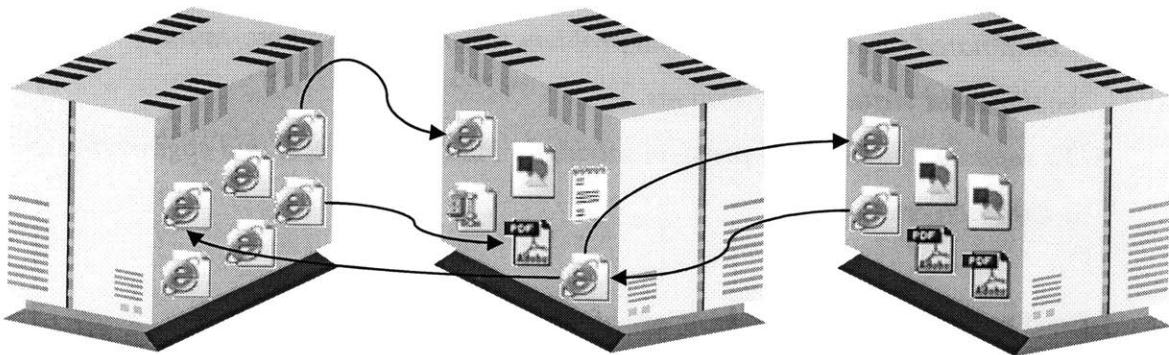


Figure 4-2 Document Linking

Starting in the late nineties Grid computing and Grid computing protocols are vying to take the next step/leap in connectivity. Grid protocols link software, databases, simulations and visualizations tools, and the number crunching power of computers – see diagram bellow. If successful, the impact of grid protocols could be even bigger than linking documents (HTTP).

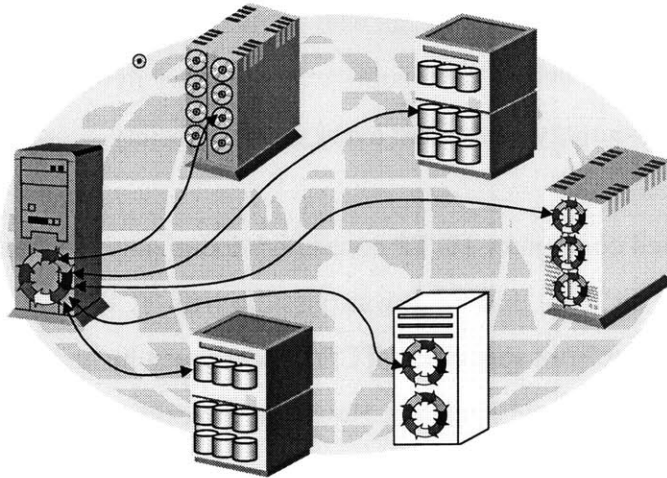


Figure 4-3 Computing Resources Linking

The fundamental problem of grid computing is:

- *Coordinated resource sharing and problem solving in a dynamic, multi-institutional virtual organizations.*
- *To support collaborative problem solving in industry, science, and engineering in a data rich environment.*

As defined by Ian Foster in the “*Anatomy of the Grid*” [Foster et al, 2001].

Resource sharing has three dimensions that can be singled out for discussion. The first is client type; operating systems in academe and industry vary with in families and across vendors. For example to mention a few with in Linux there is Red Hat, Mandrake, Debian, Knoppix, Slackware, Caldera, Turbolinux, ELX, and Lycoris to name a few. At last count there were 131 available distributions online from DistroWatch, distrowatch.com. Likewise, UNIX and Windows have arrays of operating systems. The plethora of operating systems necessitates connectivity that is independent of the operating system, much like browsers.

A second dimension is access type. There is a need to connect to computers, software, and data. For example, in cases of distributed computation there is need to distribute

computation over the pool of available hosts. Computing hosts need to communicate status and overall computation may need to be coordinated. The access must be granular enough to be able to connect to the software running on each machine and interact with the data. The access must also be done in a platform independent manner to promote scalability and easy of deployment.

The third dimension is control. Control refers to more than permissions; it refers what is shared such as software on a machine, and/or computational cycles, and/or data. Who is allowed to share, users or groups, conditional situations in which sharing will occur, delegation of sharing, and policies of sharing.

A set of individuals and/or institutions defined by sharing rules is a Virtual Organization (VO). For more on VOs see the section on virtual organizations.

More and more so, collaborative efforts involve individuals from different parts of an organization – or across organizations – where the information technology infrastructure is different enough that online collaboration is a challenge. On each effort, a number of individuals whose level of trust needs to be established with varying degrees of prior relationship want to share computational resources in the course of a collaborative effort.

4.3 Grid Types

Most of the discussion on the grid is centered on computation, data, and services grids. The computational grid has three major variations. In the simplest case existing applications run on an available machine on the grid rather than locally. In the second case applications are broken down into parts in order to execute each part in parallel among machines on the grid. The third is an application that needs to run many times on many machines on the web [Jacob, 2003].

While on one type of grid high performance servers are set aside to provide computing power on the other desktop machines are scavenged for available CPU cycles and other

resources. Sample applications of the concept are SETI and Google. Depending on the type of application, a larger number of machines or higher computing power on a single machines maybe a more advantageous grid solution.

In the computational grid, the user or customer is requesting raw resources. The grid can perform everything from brokering computing cycles to complete solutions such as an application service provider. Much like the super computing of past decades but offered on new protocols and architectures. Grid proposals are leveraging the new scalable architectures of web programming and publishing web capability to expose the application programming interfaces of grid applications. Though enabling solutions exist today, there are still many more in the making that will round out the web programming model like protocols for transaction, coordination, addressing, and security.

In the data grid each available machine provides some storage for grid use. Storage can be in memory or secondary storage in hard disk or other storage device. The trade-off between the storage types is speed and persistence. Secondary storage can be used to increase capacity, reliability, and performance. Capacity can be increased by using storage in multiple machines and making it available through a unifying file system or protocol. Distributed implementations of data striping can improve performance and redundancy for data safety.

The third type of grid is a “federation of services”. The grid offers a number of services for consumption. Each grid having a number of domain specific services as well as a common set that is shared among grids of the same type. In an academic grid, examples of the common services are registration, document store, calendar, and forums. Grid users deliver data to an algorithm for processing at the service end point. This work presents a federation of services architecture.

4.4 The Grid

Ian foster, from Argonne National labs coined the “grid” name and defined the grid problem, "integrate services across virtual organizations formed from disparate resources within and with out an enterprise" [Foster et al, 2001]. The grid problem is also stated in terms of protocols and services.

A service approach is important because a service is defined solely by the communication protocol and the implemented functionality. The services offered can be used as is or extended to create new services. That is, the user is not burdened with setup or installation issues. The resource specific dependencies on the server side are hidden and invisible to the service consumer. For more on services architectures, see the section on web and XML architectures.

Application programming interfaces (API) and software development kits (SDK) are important but secondary to protocols and services. API and SDKs can improve application robustness, development costs, and maintenance costs and are important. However, with out common protocols interoperability can only be achieved by having a single implementation or by having every consumer learn the details of every interface it wants to consume. Thus, while APIs and SDKs are beneficial they are even more so when coupled with open protocols.

Protocols are the lingua franca used between applications. A protocol specifies the structure of information used in messaging between the protocols. The protocol does not involve any internal applications issues, instead it acts as a go between for application messaging. Protocols are key to the grid because a protocol defines how a distributed system interacts with another and the structure of the information exchanged during this interaction. The focus on the descriptive external resource hooks facilitates an engagement foregoing the preoccupation of implementation issues. Resources being shared or consumed must still integrate into local policies and allow individual institutions to maintain ultimate control over their own resources. Since protocols focus

on the information exchange and not the implementation, the control of local resources is maintained.

Another critical component of the grid problem is trust and heterogeneous distributed computing. The issue of trust is a difficult problem in distributed computing. Grid participants must be able to trust a remote service with their data and/or application. The returning result must be trusted to be uncompromised. The remote service must be trusted to not copy the data. The remote services must be trusted to not copy the executing code. Finally, the remote service and the calling application must devise and implement a security protocol to prevent interception, guarantee message integrity, and prevent identity impersonation.

The challenge of trust is compounded by the need for a heterogeneous computing solution. Trust architectures need to exist independent of programming model, operating system, or component technologies. An early effort for a federated security architecture in XML has already been proposed by IBM and Microsoft.

This work presents the grid as federation of web services. The implementation is for information products. However, given the architecture, applications can also be made to large scale science and engineering data problems. Possible scenarios are web service taking data from other programs or users. Another possibility is data mining and data stores.

The architecture is the next step in easy data publication and access – where data is complex. The architecture of web services, WSDL, and UDDI makes it easy to publish, find, and explore computing resources. The wire protocols are text based making possible for every operating system and platform to interoperate. The other big advantage is the architecture now has internet scale distributed computing.

What is different?

- Programming interoperable systems is easier.

- Leveraging services is easier.
- Having access to the wire's text based protocols makes it easier to debug.
- The architecture has web scalability.

A central focus of the information products portal grid architecture is data. The grid can be thought of as a data federation of web services whose focus is the manipulation of data.

4.5 Why is the grid important?

Today's computing environment is very diverse. Within an organization, say a university, the major information technology infrastructure applications vary significantly across departments. Starting with operating systems (OS), organizations tend to have OS camps, the most notable being Apple, UNIX, Linux, and Windows. The variety surpasses even the major pockets since within each category there is an array of additional offerings. Other major applications like Relational Database Management System (RDBMS), web servers, security systems, groupware, and workflow servers follow the same pattern. When crossing organization boundaries the problem is magnified. Not only is there the possibility of yet more technical variations but there are added complexities such as organizational culture and security policies to overcome. Many of these issues exist even within one organization but multi-organization compound the problem as each additional organization carries with it baggage of integration issues.

The variety in information technology packages discussion resonates with the business-to-business (B2B) pursuit, the exchange of services, information and/or products from one business to another. The problem stems from businesses representing information and/or products differently. The B2B differences can be seemingly small issues as different representations for date fields. On other cases, the data models for transactions maybe incongruous or non-existent. Efforts for businesses to exchange data have been underway for some time. One of the earliest efforts was the Electronic Data Interchange

(EDI) with roots in the seventies and gaining traction in the eighties, still being used today by legacy applications and some business communities. A number of other standards have followed, such as Biztalk, ebXML, and RosettaNet. However, businesses still face the issue of exchanging information between Enterprise Planning Systems (ERP), exchange standard such as EDI and other business languages. The problem of integration persists. As companies come together to do business they must come up with a solution to share/exchange data.

There are several shortcomings with today's dominant distributed computing technologies. The World Wide Web's (WWW) Hypertext Transfer Protocol (HTTP) has a built in post and request mechanism that can be used to create a sharing framework. A big problem though is the absence of formalized standardized semantics for transferring complex data. If company "A" is to collaborate with company "B" what posting mechanism would they use (HTTP has two)? How would data structures, objects, be represented? There are a lot of choices. Companies could not share computational resources with out engaging in a bilateral agreement and establishing a communication standard and a protocol. The necessary coordination affects scalability. The WWW could not function at the same scale if meetings and agreement were needed every time two parties needed to exchange documents. Other technologies designed purposefully for distributed objects like the Common Object Request Broker Architecture (CORBA) and the Distributed Component Object Model (DCOM) have also had scalability difficulties. Both technologies added a layer of plumbing over TCP/IP to enable calls to remote methods as if they were local. DCOM was limited to windows and calls outside the organization ran into firewall barriers. CORBA, choose to use a different network protocol than DCOM preventing interoperability between the two object technologies. CORBA also had difficulty establishing a dominant ORB and received competition from the Remote Method Invocation (RMI) ORB that was not CORBA compliant. Both technologies had success with an organization but stumbled crossing organizational boundaries. At issue were platform interoperability, complexity in the interface definition language (IDL), firewall blocks, and the knowledge transfer that had to occur – in the

form of phone calls, meetings, agreements – before one organization could remotely use another’s remote objects.

Reviewing the technologies discussed in the previous paragraphs there are several motivations for a computational grid, a resource sharing network for computational resources in collaborative activities. The web lacks a standardized protocol to transfer computational objects and make remote method invocations. Distributed object technologies formalized remote object access but as discussed have stumbled in scale. The B2B problem is a microcosm of the general need for the grid, organizations coming together to exchange data with issues of interoperability, legacy, and competing standards. Next we discuss what the grid architecture solution.

4.6 The Grid Architecture Solution

The root of the grid effort goes back to 1997. Along the years of development there have proposals for distributed management, storage, large datasets, scheduling, coordination, replication, asynchronous, synchronous, sharing, authentication, and authorization solutions. During this time the sample implementations have evolved into the current architecture for sharing computation resources.

Protocols are at the heart of communication between computational resources. A protocol is an agreed-upon format for transmitting data between two devices. The protocol determines the type of error checking to be used, the data compression method, and message delivery reliability. Protocols have had a tremendous impact on connectivity between computational resources.

The grid architecture is a protocol solution:

“The Grid architecture is first and foremost a protocol architecture, with protocols defining the basic mechanisms by which VO users and resources

negotiate, establish, manage, and exploit sharing relationships." [Foster et al, 2001]

Protocol architectures have given way to the most scalable architectures to data, namely the internet and the World Wide Web. Starting with the transmission control protocol (TCP) coupled with the internet protocol (IP) computing had a robust protocol for routing, packing, and messaging. The implementation of TCP/IP spread from the research lab to academe and industry as the protocol became widely adopted. In time TCP/IP became the standard for the internet and later coupled with the Hypertext Transfer Protocol (HTTP) the World Wide Web. In the mid to late nineties, learning from the interoperability difficulties of past, exchange protocols moved to what has become the lingua franca between platforms, the Extensible Markup Language (XML).

The grid needs to be able to establish relationships among any of the participating members. Being able to put all participant on the same computational context sharing resources and enabling resource sharing relationships is the most the most important and distinct aspect of the grid architecture. Thus, interoperability is the central issue the architecture needs to address. In the current computational environment of linked computer systems interoperability takes the form of common protocols. XML protocols are human-readable, text-based, and can be parsed on any platform. XML is information enclosed in tags, bracket delimited. XML allows designers to create their own customized tags enabling the definition, transmission, validation, and interpretation of data between applications and between organizations. The flexibility and simplicity of XML has been leveraged in the specification of recent protocols to access software in the network, distributed software.

The architecture has adopted XML protocols to establish sharing relationships.

4.7 Grid Technology

The underlying protocol of remote method invocation is the Simple Object Access Protocol (SOAP). SOAP originally conceived for remote object access has been generalized as a lightweight protocol for the exchange of information in a decentralized, distributed environment. The XML based protocol, much like HTTP, is very simple and allows extensibility. SOAP consists of three parts: the envelope, the header, and the body. The envelope is the top most or root element in the XML document that represents the SOAP message. The header is a generic container where additions can be made to SOAP in a decentralized way. The header allows you to place a payload header in the top of your SOAP message. The additional information is used to better describe the payload in the SOAP body. The body is mandatory information for the message receiver. In RPC, the body of a SOAP envelope is where the information for the remote method call is placed.

Chapter 5 Inter-Organization Collaboration

Inter-organization collaborations have always had a number of challenges. One of them is the lack of a common software applications or information technology infrastructure.

Applications to support collaboration typically evolve slowly over time with in single organization [Anson, 2001]. A workflow application, to name one, is derived from the organizational process is trying to support. The task of gathering requirements, design, architecture, integration to the organization's existing infrastructure, and deployment is a lengthy and expensive process [McConnell, 1996]. Along the way there are logistical, technical, and cultural hurdles to overcome.

Inter-organization collaborations compound the challenges and bring to bear new ones. Part of the challenge is that a "new" organization is being created with a multi-organization membership [Blair, 1991]. The diversity of interests and stake holders raises the number of issues in creating the new organization. Often, these organizations are short lived and hence have no physical or virtual infrastructure. For now, let us call these "Collaboration Organizations" (COs).

5.1 Collaboration Organizations Samples

Life Long Learning (LLL)

Boos Allen Hamilton engaged by MIT's Council of Educational Technology (CET) to recommend a strategy for Life Long Learning (LLL)

MIT Course

1.124J Foundations of Software Engineering, Information technology course, MIT, course 1.

Product Development

CIPD, MIT, ONR, and Xerox; collaborative product development

5.2 Organization Differences

COs need an information technology infrastructure that goes beyond document exchange and webs together computers, software, and data.

Even in the cases where document exchange is the core of the collaboration - say government, academia, and industry groups writing new law – document exchange is only part of the resources needed. Version control, changes audit, history tracking, roll back to earlier versions, integration manager, and publishing model are some of the additional computing resources that would be needed.

There are many differences among the collaboration organizations. The organizations vary in purpose, scope, size, duration, structure, community, community, and sociology [Foster et al, 2001]. However, collaboration organizations also have a common set of requirements.

The common requirements are discussed in the sections to follow.

5.3 Software & Data

Each CO member brings different expertise, skills, and computing resources to the collaboration effort. To leverage the COs resources there is a need integrate computers and software. The IT industry has a niche of system integrators, companies dedicated to integrating software to an organization's information technology infrastructure. The market exists because there is tremendous variety in the information technology used by each organization and even within each corporation. The variety comes in the form of programmatic interfaces, protocols, and operating systems among others. Integration in a company is not trivial, across organizations it is a formidable task.

Many of the same challenges of software integration exist in data integration. Members must open part of their data stores and connect to the rest to create a distributed data model. Normalization of data, data integrity, and foreign/private key relationships are modeled as part of one system. Current protocols and languages to model data do not address the distributed inter-organization scenario. As with software there also interoperability issues to address. Ownership of data is another issue to resolve among the members.

5.4 Permissions

The integration and sharing that occurs needs to be very granular. Permissions are determined by each organization and then each resource. Prescribed is who can access, when access is permitted, what level of access is authorized, and under which conditions access is permitted. Each resource implements organization permissions or a custom set. Therefore, the integrated set of resources will have an array of permissions sets to manage.

Two core components of permissions are authentication and authorization.

Authentication is the process of verifying the user credentials; making sure the user is who he claims to be. Once a user is authenticated, authorization determines what the user is permitted do. For convenience functionality and authorization are grouped into roles. Example roles are: administrator, editor, author, reader.

A user may have multiple roles and the roles can be dynamic. As a user works through an application permissions maybe granted and then removed. For example, a user submitting material for evaluation must have authoring/editing permission during creation. However, once the material is submitted, the user must loose the ability to author/edit the material.

Another sought functionality is system policies. At a global or local scope the application of system rules establishing default behavior. Roles are resource specific while policies are system wide.

There are situations in which is necessary to defer to another authority the access decision. It maybe that the current resource does not have the credentials to verify access or that the user does not belong to the domain. In other cases it is advantageous to not replicate data and simply provide a pointer to the authentication authority.

5.5 Scalability

The resulting mesh of shared resources must be able to scale and be extended to encompass new resources and new organizations. The collaborative application design must choose protocols, interfaces, clients, and data stores that favor scalability. The design must also be flexible; new resources should bolt on with out the need for redesign or reconstruction. And authentication and authorization needs to be able integrate new organizations into the system.

5.6 Virtual Organizations

The collaboration organization bound by the sharing rules derived from the programmatic resources they share has come to be known as a Virtual Organization (VO) [Foster et al, 2001].

The information technology challenge of the Virtual Organization is the focus of this work. Moreover, the infrastructure required to build applications for collaboration organizations and the unique challenges of resource authentication, authorization, access, discovery, presentation, and scalability.

The diversity of participants brings a cornucopia of differences among the VOs. Starting with the objectives and purpose; each participant is apt to have different stakeholders, affecting the scope and duration of engagement and commitment of resources.

Over the lifecycle of a project the duration of the participation will vary among participants. Some members will be involved in the formative process and decide to fade out as the project continues. Likewise, other members will only join the advanced stages of the collaboration. The length of duration and degree of involvement will change based on the interests and motivation of the members.

VOs will differ in structure and community. The structure and matching software will be chosen based on the domain of the project. For example, projects collecting experimental data are apt to have a longer duration than ones coming together for a practical implementation. Similarly, the communities formed around collaborative projects will vary in expertise, character, and interaction levels reflecting the personality of the domain.

As might be expected, just as there are many differences there is also a fundamental set of common requirements for VO collaborative efforts. The commonalities have four dimensions that maybe singled out for discussion.

The first is the need for sharing of varying resources. Collaborative efforts need to be able to share resources at various levels of granularity. Probably one of the most visible is documents. The hierarchical structuring of documents, the programmatic representation of the document, the metadata associated with each document are some of the design decisions to store and share documents. Another resource to be shared is data.

Collaborative efforts often have data stored that needs to be shared. The organization of the data to be shared, how it is shared, and the format of the data are all sharing considerations. Software running on machines would another desired resource to share. In collaborative efforts there is often domain software that needs to be shared with the rest of the group. One example of such software is simulators. These programs are

typically exotic enough that distribution is unrealistic. The best solution in these cases is to share the software from where it lives. Moreover, often it is convenient to share unsophisticated software. A common example is a calendar, once one is written and implemented resources would be saved by reusing it. In case of a collaborative a calendar application has the added advantage of centralizing the group's calendar events.

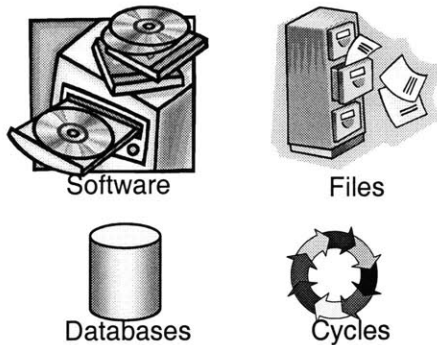


Figure 5-1 Computational Resources

A second group of common requirements are flexible sharing relationships. The resources shared need to be accessed from more than one client type and more than one client server model. Clients need to connect from an unknown number of platforms and collaborative efforts need to accommodate the variety. Beyond platform flavors there is the client server model. Resources need to be shared on a traditional client server model as well as a peer-to-peer architecture where every client is also a server.

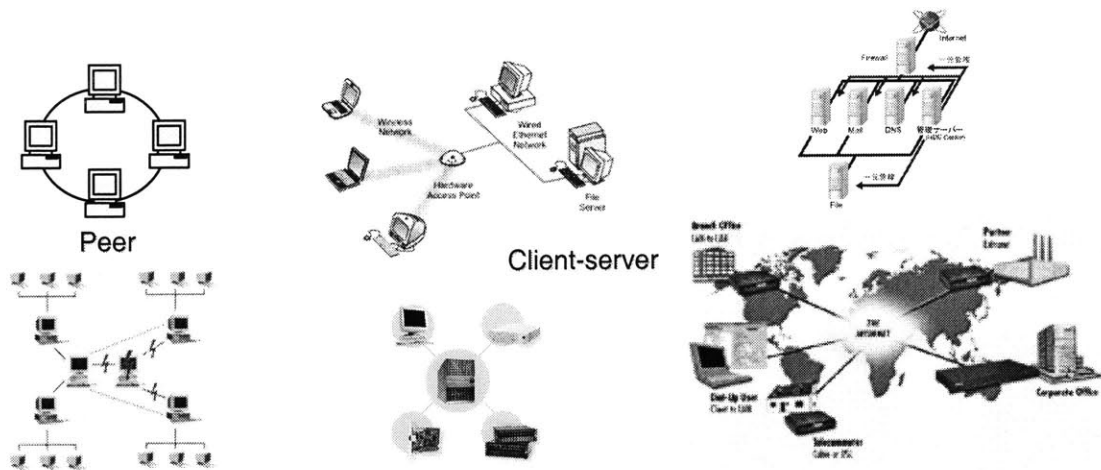


Figure 5-2 Flexible Sharing Relationships

A third common requirement is the need for diverse control levels. The granularity and diversity of control must be very rich to accommodate sharing collaborative resources. Resources being shared quickly run into the need of the concept of roles for users sharing the same functionality. Implicit is the need to have a mechanism to create and manage users and groups. The user management component is necessarily modular, accessible, and portable enough that it can be used by all the resources. For resources or users that span security zones and additional requirement will be the ability to delegate permission credentials to other security systems. To facilitate management resource control should implement policies. Given the distributed nature of resources the collaboration solution needs to have the concept of local as well as global policies.

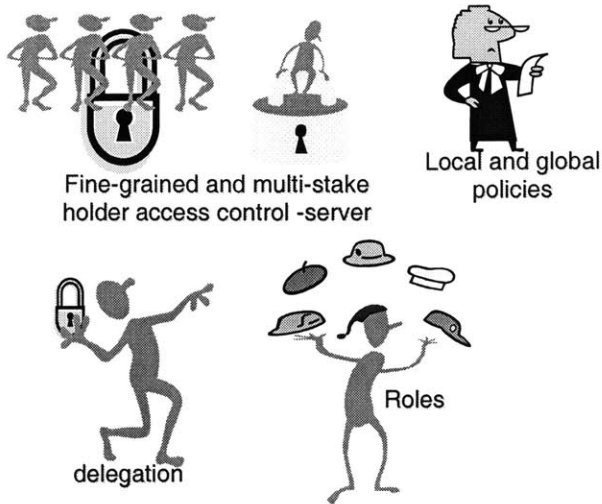


Figure 5-3 Diverse Control Levels

The fourth common requirement is diverse usage modes. For example, resources need to allow usage by individuals or groups. That is, users need to be able to access resources and perform operations individually or as a group. The usage modes exceed user access and include operating modes that favor or conserve performance. A resource at times will need/want to operate in a cost sensitive context. The context will seek to minimize the resource execution tax on the host. The host may also want to do the opposite, devote all the host resources to the resource being shared. Furthermore, there are bound to be unforeseen usage modes the sharing framework must be extensible to accommodate future modes.

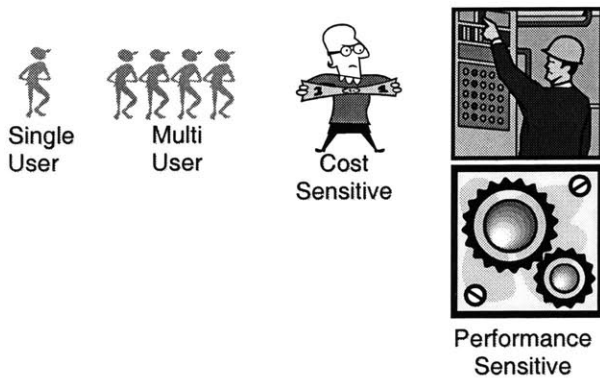


Figure 5-4 Diverse Usage Modes

Chapter 6 Web Architectures

6.1 Introduction

This chapter introduces web inspired architectures. XML based programming (XBP) as the glue to enable interoperability between heterogeneous independent distributed computing platforms. A service oriented programming model and a service oriented architecture.

6.2 Web-Like Design

As many observers have pointed out, the web was not well designed as a protocol. The web lacked persistent connections, central tag control, and it ignored earlier hypertext's theoretical work like the need for bidirectional links. However, as many have also pointed out, the weaknesses were also the strengths of the protocol.

Other protocols of the day like Gopher were too rigid, strong, and robustly attached to a designed implementation. The web in contrast was a minimalist, weak, and extensible protocol. It was no more than a pointer collection at the start but it was the weakness that was the strength of the protocol. For example, all hypertexts models at the time called for bidirectional links but the level of coordination and complexity added by this feature would have been prohibitive for the web. Dangling links in contrast scaled like no prior architecture in computing history.

The lesson appears to be that in the fast changing world of computing weak evolvable designs will win over strong, focused, and rigid designs.

Looking closer there are a number of additional observations. Instant satisfaction, implementations must work immediately even if only partly implemented. A bottom up approach, the implementation can start with a few pockets and then can scale to the network.

Evolvable applications are ever changing, adapting, and for the same reason not at the bleeding edge. Critics are fast to point out the shortcomings of the design but one would do well to remember the lessons of the web architecture where the shortcomings are the strength. Sam Ruby, IBM architect, said it well, “Centrally designed protocols start out strong and improve logarithmically. Evolvable protocols start out weak and improve exponentially. Infrastructure built on evolvable protocols will always be partially incomplete, partially wrong and ultimately better designed than its competition.”

6.3 XML

This work is based on web architectures and extensively in XML. XML has evolved and matured extensively from its origins in the late 1990s. Below is an image presenting the current breath of XML.

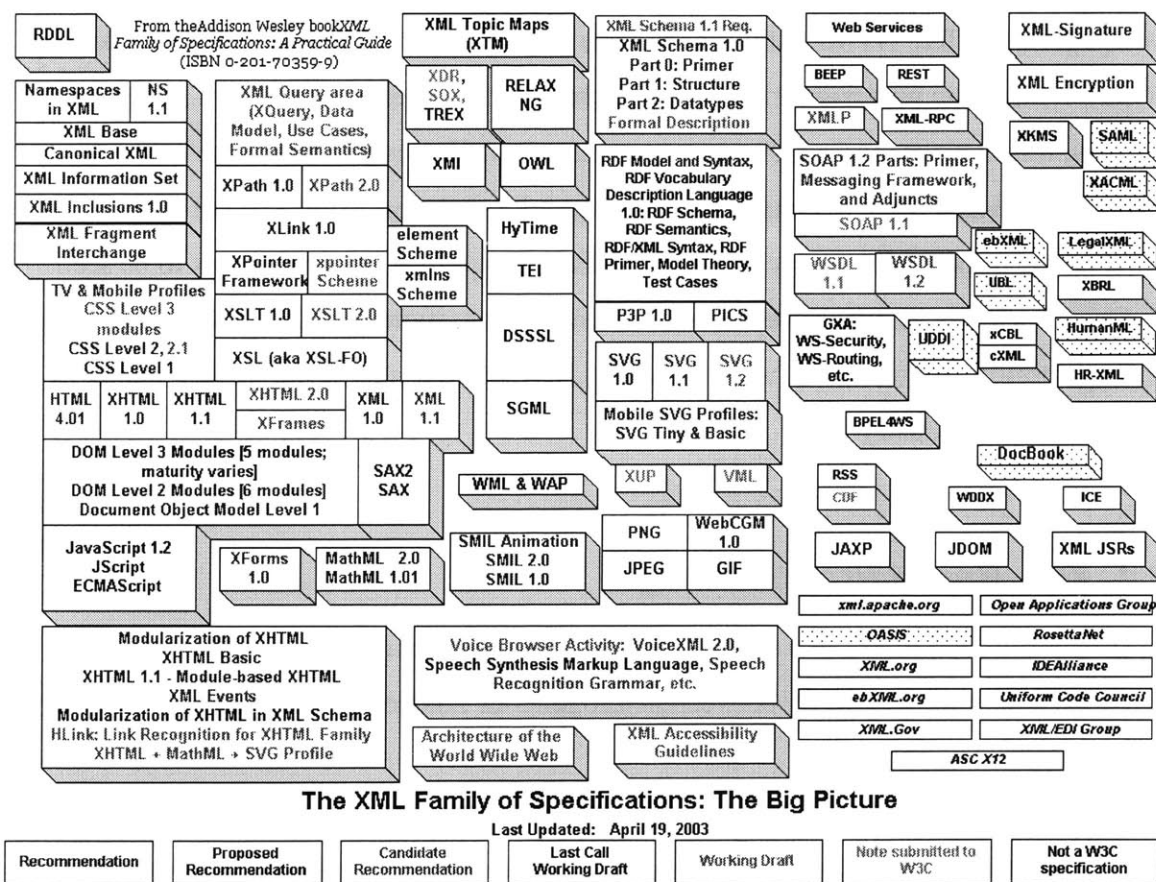


Figure 6-1 XML Family of Specifications [Sall, 2002]

One of the differentiating characteristics of XML is its flexibility. The flexibility can be noted on data models where XML data does not have the limitations of Relational Database Management Systems (RDMS). At any point, new attributes can be added, data can be incomplete, and there is always the capability of carrying “<any>” tags where unplanned XML can go. Even though XML can be bound to object models and made rigid, the underlying tagged data flexibility is always be there.

It is the flexibility, adaptability, interoperability, and extensibility that have caused XML to spread at a phenomenal rate. XML has penetrated databases, querying, office suites, web servers, security, messaging, component technologies, and many other technologies as depicted in the image above.

In basic XML web services there is a close analogy to basic web pages in HTTP. Basic web pages use URL, HTML, and HTTP. Web services use a URI, XML, schema, and SOAP. As in basic web pages there is much that is not addressed. There are issues like security, transactions, and attachments to name a few. However, even at the minimalist level web services work well just like web pages did in the early days of the web.

Weather the basic functionality is enough to have ubiquitous buy in remains to be seen. So far web service implementations abound and organizations like the Web Services Interoperability Organization (www.ws-i.org) have representation from every major player in the IT industry.

6.4 XML Based Programming

XML is positioned to become the lingua franca between computational resources. The glue to enable interoperability between heterogeneous independent distributed computing platforms.

Component technologies architects design computational abstractions which become infrastructure to facilitate component development. Building on the abstractions

leverages computational building blocks that shorten the construction time, enables component reuse, lowers construction costs, and once a pool of components exists enables greater product variety. However, in the heterogeneous computing environment of the market place, competing component technologies have been a hotly debated subject. The division lines have mirrored the already established divisions among operating systems, programming languages, and development environments. The corporations and organizations behind the technologies have fought hard to gain market share, developers, and become the dominant component building platform.

The leading component technologies at the time of writing are: the Component Object Model (COM), the Common Request Broker Architecture (CORBA), and Java technology. Although the technologies act as the glue between the computational resources built on the platform, the degree of interoperability beyond it can be discussed in terms of four layers:

- *In-memory interoperation* standardization can offer excellent performance and still offer rich component management services.
- Standardizing on an application programming interface (API) to access computational resources the platform can provide *source code interoperation*.
- By providing a standardized programmatic description of the component consuming platforms can localize the type information gaining *type information interoperation*.
- Wire interoperation, using network protocols component technologies aim to leverage remote components with the same ease of local components.

There are a number of choices for leveraging computational resources and achieving interoperability. Organizations and corporations have made platform choices and invested people, training, and software among others to make the most of the platform. Changing

the component technology requires reinvestment and will be resisted. However, the promise of interoperability has proven difficult in the current platform incarnations and moving to a more scaleable, loosely coupled, and firewall friendly protocol namely HTTP and XML is very attractive.

Xml in the context of interoperability standardizes the text based wire protocol to exchange data and messages. In so doing using Xml builds a minimal standard for component communication. Although XML applications, schemas, and extensions are growing rapidly, the fundamental XML used for documents is human readable, the w3C standard is stable, and widely adopted.

One of the biggest appeals of XML is platform indifference. Operating systems are not an issue. Neither is programming languages, application servers, or application domain. The absence of application programming interface conformances is added flexibility to the messaging client/server. The host receiving the wire message can pick up the data document, parse it, and under the hosts control determine how to act on the message.

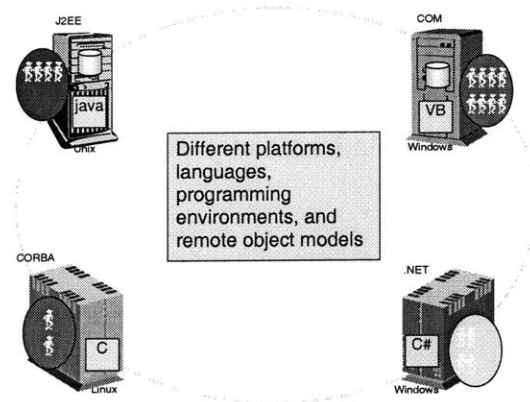


Figure 6-2 Diverse Computing Environment

Much like the Hyper Text Transfer Protocol (HTTP) XML is user friendly. XML documents are human readable and can be composed by any text editor. The hierarchical representation of data like HTTP tags reveals the document structure and content to the document reader. By contrast COM, CORBA, and Java are binary protocols that require

component construction using specialized programming environments and low-level programming languages.

Another big plus of XML is the flexibility of the message document. As time passes organizations needs evolve and the XML messages need to be extended. Making changes to an application programming interface is difficult if the interface is widely consumed. However, adding extra information to an XML message document need not affect the consuming client. The component picking up the wire message can simply ignore the additional information. Inclusion of common identifiers can be cleaned up with namespaces. Namespaces can also be applied to document sections to explicitly declare the message context.

Given the large investment by organizations into their current computing platforms major changes are unlikely. COM, CORBA, and Java will continue to be used in the software industry. However, for interoperability between component platforms XML is fast becoming the lingua franca of choice.

6.5 Service Oriented Programming and Architecture

6.5.1 Background

The emergence of XML web services has initiated discussion on a service oriented programming model. The term most commonly used is SOA (Service Oriented Architecture) not to be confused with SOAP (Simple Object Access Protocol). In some instances SOAP is repurposed to mean Service Oriented Architecture and Programming. In this section SOAP refers to the protocol definition.

Like no other technology before it, with the exception of HTTP, software vendors are uniting to support service oriented architectures. SOA does not necessarily mean web services. However, practically, SOA is enabled by SOAP like the internet is enabled by TCP/IP. A service in this section is referred to as a SOAP callable operation.

6.5.2 OOP

Object oriented programming (OOP) introduced by Brian Cox introduced a new paradigm to the world of procedural programming. In time objects were embraced by the software industry. And objects were used for modularization and really shined in creating abstraction layers. Architects grouped abstractions to create common visions leading to many of the platforms in existence today.

Objects lowered the cost of abstraction. The purpose of OOP designers was not executables but to architect new abstractions. As systems were developed and deployed it was noted that the tight coupling of the programming model meant that objects were not built for field replacement. The idea of reusable/replaceable software parts has been difficult to realize. There are some widely used libraries like the Microsoft Foundation Classes (MFC). However, in general libraries rarely span the birth project. The time and cost required to make abstractions generally useful make it prohibitive. As a consequence the resulting libraries require extensive customization to be leveraged by new projects.

6.5.3 OOP Disadvantages

Time makes abstraction expensive. When creating objects at design time abstraction is cheap. However, when building *new systems* trying to leverage existing software, abstraction is expensive.

Distance makes abstraction expensive. Local in memory abstractions do not always translate to the distributed case.

Extensibility makes abstraction expensive. Multiple extensibility coupled with unknown extensibility is very difficult to manage. Open extensibility is even worse.

Standardization makes abstraction expensive. Design by committee is a barrier for speed and focus. The interests of multiple stake holders waters down the original use case focus

as more interests influence the specification. The politics of standard bodies also lengthen the creation process.

6.5.4 The SOA Alternative

SOA offers a new architecture for software reuse. SOA brings a new set of tools and a new approach to software reuse.

The XML kernel addresses the shortcomings of OOP. The XML kernel, the core for applications, is SOAP, HTTP, and XML. SOAP limits the number of abstractions required for interoperability. Much like HTML the XML header is minimal. And like HTTP much more can be added if needed.

Service oriented architectures are about having a small set of fixed abstractions and no more. Services assume a minimal shared kernel: XML, HTTP, and SOAP. Unlike prior architectures SOA expects high latency and low fidelity communication channels. All communication between hosts and clients is made in XML documents. As such, the operating system and platform is irrelevant.

New XML layers take care of transactional elements. However, the architecture is bottom up and granular. The additions to the XML message are simply more text in the document being passed between members of the transaction.

New systems are composed of services at the boundary of coarse grained messages. Instead of changing the application, the service “A” can become part of a new service “B” that encompasses service “A”. A previous consumer of “A” continues to use the service with out interruption of change. The consumers of “B” will have the functionality of “A” plus any that “B” may have added.

Services share schemas and contracts but not types. The services contract metadata details the interface contract exposed by the service. Knowledge of the contract is

necessary to build the remote service proxy. The schema validates the data on the wire. Integration is not about sharing types it is about sharing machine readable validation instructions.

6.5.5 OOP SOA Differences

There a number of differences between the OOP and SOAP paradigms. Object oriented development and testing lifecycle models are well known and managed. Construction is done on abstraction driven integrated development environments. Objects are platform dependent and additional abstractions are needed to port modules. Each vendor implements security privately.

In contrast, services are designed for an unknown integration partner. As such, input/output constraints are the key to service consumption. Services are also built to be consumed by unknown platforms. It follows that the sine qui non of SOAP is interoperability.

Given such requirements, service based applications have a potential to be distributed and more universal than any prior architecture. Given the number of vulnerability exposure points, service oriented architectures need comprehensive security at all phases of transactions. In contrast to OOP, SOAP is just starting and the construction tools are segmented, myopic, and rudimentary.

The shift from OOP to SOAP is a focus shift from types and abstractions to schemas and contracts. The binary specification of the past are replaced with simple, minimalist, and granular XML based protocols. The shift to XML document based messaging between systems offers a looser coupling than remote procedure calls architectures and offers new levels of flexibility in the long enterprise supply chains.

6.5.6 SOA Observations

Thinking in terms of software oriented architectures has motivated industry and government to work towards an enterprise architecture. That is, business capabilities, functions, and processes that are supported by an application architecture. An example of such a system is the Federal Procurement Data System from the federal government. The system is a services-oriented architecture employing web services, open standards, and XML, and is targeted to be base for a federal enterprise system. The government is undergoing an effort to create a common architecture for all key agencies. The effort is being driven by the current information technology disarray. The current collections of legacy applications correspond to generations of incompatible architectures, non-interoperable platforms, ever increasing incongruent data silos, and incompatible technology.

The next challenge is to create a service oriented architecture virtual fabric of web services. The fabric is constructed on the services that solve holistic business problems. The thread that connects systems is web services. The glue between threads is the overarching applications and roles. The difference to traditional glue is that it behaves more like velcro. A completely new fabric can be constructed by rearranging the threads.

Thinking in terms of services is conducive to product line architectures (PLA). PLA thinking has been employed successfully in embedded systems and real time. The productivity gains come when a platform is designed to be leveraged to create new products. See the chapter on platforms for an in depth discussion on this topic.

6.5.7 SOA Fabric

One of the insights of service oriented architecture is that the power of SOA does not come from the services themselves but instead from how the services are composed into new applications. Services are not required to do more than before. Instead, SOA is extended through services (threads) woven into new applications by creating a new fabric

(messaging routes). The services interfaces remain constant but the applications composed from the services and the services using the threads are free to grow.

Small, robust, well designed services can perform tremendously complex tasks as a group (fabric). The simpler the service (thread) the lower the tax on the service consumer. The limitation is the ability to model the messaging routes through the service fabric application. At the same time simpler components lower the cost of maintenance. The abstraction boundary forces a cleaner encapsulation of functionality and replacement of a service (thread) is more modular.

SOA is akin to software agents in behavior. A SOA service is a program that performs some information gathering or processing task independently. The services, as discussed, are very small and well defined. Complexity beyond the discrete agent-like service comes in the form of the web services infrastructure and applications defining the message routes for the application.

Message routes themselves are included in the service promoting independence further decoupling the infrastructure and negating the need for a coordinator.

SOA posits that it is much easier to manage change in the composition of many small web services who share few well known abstractions than it is to manage change in varying abstractions exposed by the same set of services. In the former the difficulty lies in composing messaging routes. In the latter the barrier lies in propagating the knowledge to consume each abstraction. The two approaches highlight the two possible programming models discussed in this section. The case has been made for the high cost of abstraction in programming models. SOA in contrast seeks to minimize the abstractions, maximize the connectivity, minimize services complexity, and push complexity into the connections. The tradeoff can be labeled 'Fat Versus Thin'. In one case the service is "fat"; the service has heavy built in logic, many abstractions, and many interfaces. In the "thin" case the service is minimal, has few and well known abstractions and interfaces, and complexity is pushed into the connections. The vision is thus:

flexibility, extensibility, and interoperability enabled by software oriented architectures (SOA) and XML web services.

6.6 Protocol Independence and SOA

The service oriented architecture approach can also be followed independent of the implementation protocol. The application can add a messaging abstraction to generalize communication. Abstracting messaging and protocol to a separate layer allows the implementation to replace the protocol without having to touch the application. As mentioned, currently SOAP messaging has most industry support and it is used in this work.

6.7 Business processes

One of the roots for component technologies are business processes. Business processes are distributed and need distributed software. Distributed systems within an organization have been challenging; distributed systems across organizations are one of the big goals of industry, government, and academe.

Systems need to adapt to the host devices, scale up on high performance computers, or distribute across computing farms, and span geography and organizations. To meet such requirements services need to be interoperable across platforms and devices. Applications must be composed of reusable service components that can work individually or coupled with other services. And to realize inter-organization applications participating organizations must federate and standardize.

Chapter 7 XML Architecture

7.1 Schema Definition Language

Types are composed of both properties and behaviors. Properties contain the data that is exposed by the type. Behaviors are the functionality of the instance of the type, the object, and are defined by the public methods (also called member functions) and events of the type. Collectively, the public properties and methods of a class are known as the object interface.

A type system is the common infrastructure defining base value and operation space. Using type information compilers can validate code and can generate code for referenced types. Similarly, compilers and runtime engines use the type system information to manage memory during program execution.

Reflection, a feature of attribute-based programming is the process that allows a type to query its own metadata. Reflection allows an application to discover information about itself so that it may display this information to the interfacing application, modify its behavior at runtime, and loading components at runtime to build dynamic extensible applications on the fly. Reflection underlies much of the infrastructure type support in managed code applications like Microsoft .Net Framework and Java [Campione, 2000; Liberty, 2003].

The Schema Definition Language (XSD) introduces the type system lacking in XML 1.0. XSD is the metadata that describes types in XML [W3C-Schema, 2003]. A schema is analogous to a class and an instance of a schema, a XML document, is analogous to an object in the language of object orient programming (OOP).

XSD is analogous to the programming language you use to define classes. The C# definition of a user class is:


```
public class User
{
    public int userId;
    public string firstName;
    public string lastName;
    public string Password;
}
```

The equivalent declaration in XSD is:

```
<complexType name="User">
    <sequence>
        <element name="UserId" type="int" />
        <element name="FirstName" type="string" />
        <element name="LastName" type="string" />
        <element name="Password" type="string" />
    </sequence>
</complexType>
```

XSD type system is composed of simple and complex types. Elements are complex if they have attributes or children as in our “User” declaration. Conversely simple elements are scalar values with out attributes or children.

All types in the XSD type system derive from a base type. XSD has a hierarchy of types with “anyType” at the root. Both user defined and built in types derive from another type. The recommended built-in hierarchy tree of types specified by W3C is:

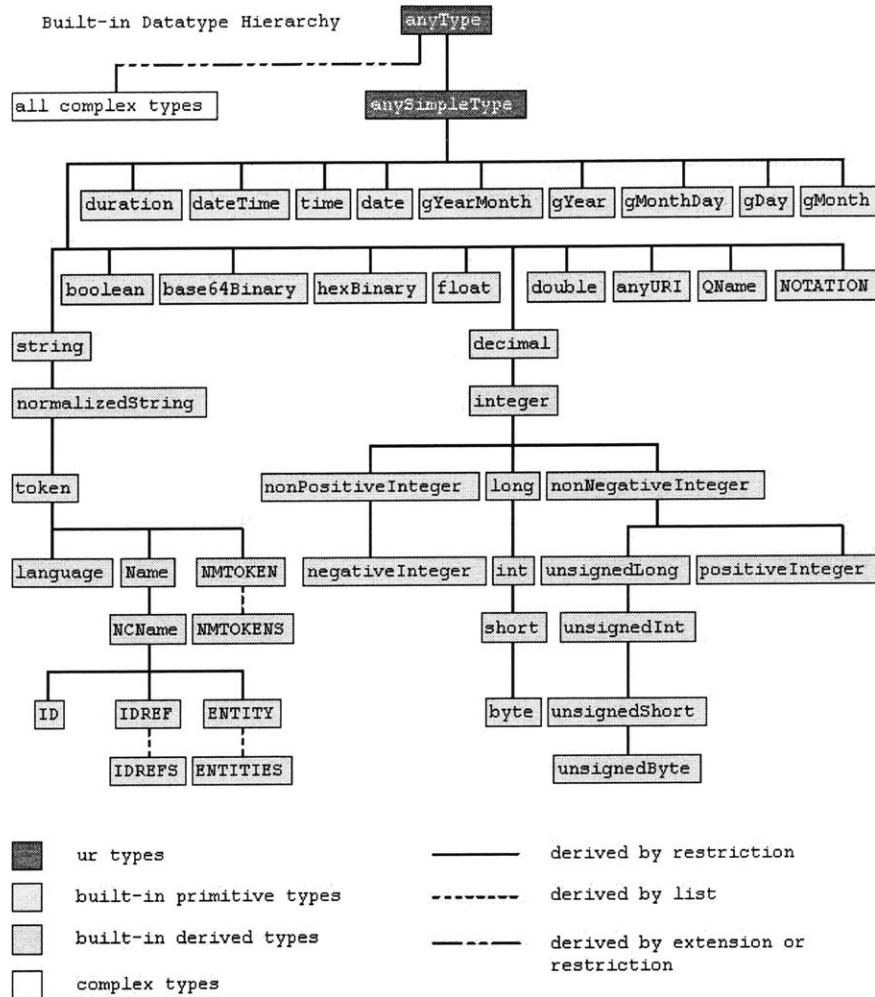


Figure 7-1 Schema Types [W3C-Schema-Types, 2003]

Types that derive directly from “anySimpleType” are called *primitives*. All complex types are called *derived*.

Schemas are composed of user defined types, built-in types, elements, attributes, and properties. An element definition needs a name and a type, for example:

```
<element name="Age" type="int">
```

The element can be qualified with maximum/minimum, for example, but must be unique.

When it comes to types, XSD permits designers to construct simple and complex types. Simple types have a value set that is a subset of the built-types. A simple type is defined by specifying a type and a restriction, using the restriction element. For example:

```
<simpleType name="AdultAge">
  <restriction base="int">
    <maxLength value="150" />
    <minLength value="18" />
  </restriction base>
</simpleType>
```

Restriction is often done using *facets*. The standard has predefined a number of facets for each of the built-in types. To follow is an example using enumeration.

```
<simpleType name="PortalTypes">
  <restriction base="string">
    <enumeration value="Team">
    <enumeration value="Course">
    <enumeration value="Department">
    <enumeration value="University">
    <enumeration value="Event">
    <enumeration value="Notification">
  </restriction>
</simpleType>
```

For a complete list of facets visit [W3C](#).

Complex types are used to represent data types. The complexType uses a compositor used to describe the type's content model. XSD defines three compositors: sequence, choice, and all. Compositors hold repeating data. Attributes are not considered compositors because they do not repeat and are declared outside of compositors. A complex type example follows.

```
<complexType name="User">
  <sequence>
    <element name="UserId" type="int" />
    <element name="FirstName" type="string" />
    <element name="LastName" type="string" />
    <element name="Password" type="string" />
  </sequence>
  <attribute name="PortalType" type="string" use="required"/>
  <attribute name="PortalID" type="string" use="required"/>
</complexType>
```

The Schema Definition Language (XSD) offers a rich platform independent type system to express classes and interfaces.

7.2 Serialization

There is currently some discussion over XML RPC and XML messaging – exchanging data between components using XML documents. The web service description language (WSDL) supports both XML RPC and XML messaging.

Interestingly, RPC can also be used for document style messaging and conversely. RPC specifies the remote method and arguments but it could still be used to exchange messages using XML documents. Since XML messaging is even more flexible it can certainly be used as RPC. “Document/literal used by XML messaging is a superset of RPC/literal. For any given RPC/literal WSDL description, an equivalent document/literal WSDL description can be created such that the wire messages are identical. All that's needed is some schema manipulation to formally describe the RPC/literal message.” [Shohoud,2003]

Going by flexibility, XML messaging is a better choice than XML RPC. Messages are used to exchange objects and interfaces by distributed components. Before the exchange objects need to be transformed from the computing language and in memory representation in case of instances to a XML representation. The conversion to XML is called *serialization*. Once the message reaches the destination it needs to be reconstructed. The XML message needs to be parsed, the object reconstructed, and the data mapped to the component. The process of remapping the XML document to an object is called deserialization. Serialization is supported by both .Net and Java [JavaSoft, 2003; Obermeyer, 2001].

Considering a “user” class represented in C# as an example:

```

public class User
{
    public string userId;
    public string firstName;
    public string lastName;
    public string Password;
}

```

The class is represented in an XML document schema as:

```

<complexType name="User">
  <sequence>
    <element name="UserId" type="string" />
    <element name="FirstName" type="string" />
    <element name="LastName" type="string" />
    <element name="Password" type="string" />
  </sequence>
</complexType>

```

An instance of the “user” class in XML:

```

<User>
  <UserId>johnd@mit.edu</UserId>
  <FirstName>John</FirstName>
  <LastName>Doe</LastName>
  <Password>John'sPassword</Password>
</User>

```

Components exchanging messages need to agree on a schema before hand. The schema serves as the template for serialization and deserialization. When a target component receives an XML document it validates the document using the schema. The receiving host then needs to create a new class to represent the “user”. In the case where a schema does not exist, one can be derived as long as the XML file contains enough data to accurately derive the data structure.

Since the creation of the “user” class is controlled by the receiving host it can be extended or customized if desired. In the example we are considering, the receiving host may wish to add attributes like address, telephone, and birthday. The loose coupling provided by XML messaging gives the target host complete freedom to manipulate and bind the message.

7.3 Object Technologies

Distributed computational systems are attractive for a number of reasons. The encapsulation that happens as components are distilled and distributed often has the nice effect of producing a collection of reusable modules. Once the computational logic has been modularized and is listening for requests it can be consumed by more than one application. For example, once a user registration module is built all systems requiring registration need only reference the one that already exists. Furthermore, the distributed components themselves can then be referenced and grouped in new ways to produce new computational products. Other benefits are: updating source with out affecting the rest of the system; distributing computational load among machines; and moving application logic close to data stores.

In a move to facilitate building computational objects and later distributed remote objects, several software companies initiated efforts to build object technologies.

7.3.1 Object Technologies

The leading object technologies in the ninety's were Microsoft's COM (Component Object Model Architecture) and OMG's (Object Management Group) CORBA (Common Object Request Broker Architecture) [Box, 2000]. Both technologies added a unifying layer to access binary self-contained entities (components) that consist of properties and procedures to manipulate data.

As more and more computers joined the internet in the ninety's it became increasingly important to leverage computational objects in a distributed environment. Developers could write components that connected an application to a network protocol, typically the Transmission Control Protocol/Internet Protocol (TCP/IP). Developers sent and received TCP/IP messages by opening a socket and reading and writing data to and from the socket. Sockets programming required the developer to write a client server in every case, implementing low level protocols for encoding and decoding messages. Having to code a socket solution every time was wasteful and error-prone.

Microsoft extended COM to DCOM (Distributed Component Object Model). DCOM is a network protocol based on DCE (Distributed Computing Environments). DCOM enabled execution COM components on other machines. DCOM replaced the interprocess communication with a network protocol. A local component could call a remote COM component as if it was local.

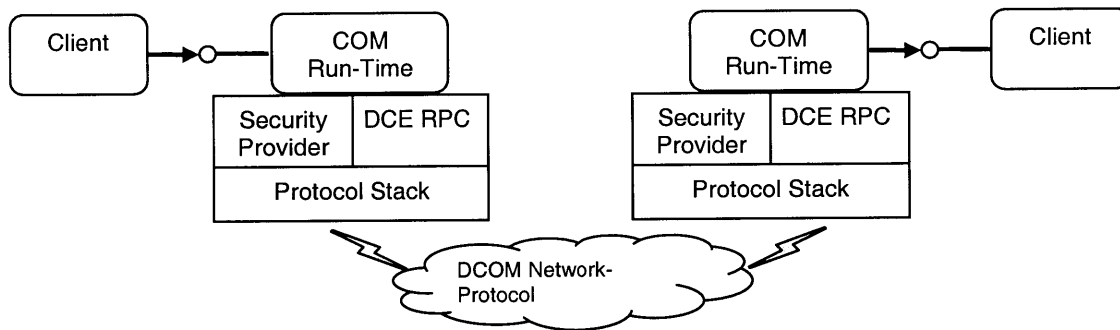


Figure 7-2 Component Object Model (COM)

DCOM unlike the World Wide Web was designed to be a stateful protocol. DCOM requires constant keep-alive messages to maintain state. Without keep-alive messages the server can become tied up holding components for clients whose session has ended. However, as the number of clients increase, the amount of network traffic causes congestion and limits scalability [Rammer, 2002].

Object Request Brokers in the CORBA solution played the connectivity role of COM in the Microsoft solution. Many expected CORBA to use DCE (Distributed Computing Environment) for the network protocol. However, a new protocol, IIOP(Inter-ORB), was developed for CORBA's remote objects. IIOP much like DCE specified how ORBs communicate over the network.

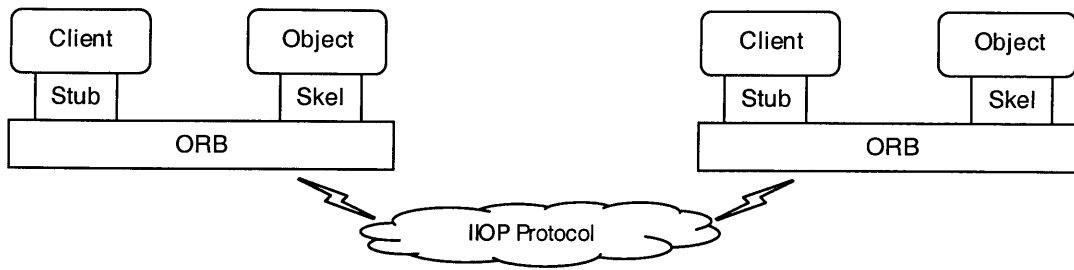


Figure 7-3 CORBA

Much like the many flavors of UNIX/Linux has created consistency, implementation, and adoption challenges the same was true of CORBA. The implementation variety and lack of a dominant ORB plagued the platform [Rammer, 2002]. To make things worse Sun developed a proprietary ORB called RMI (Remote Method Invocation) based on the Java language. RMI was the obvious choice for the Java programmers. However, the Java platform brought even more variations and integration points. What is more, the Java solution is not even CORBA compliant.

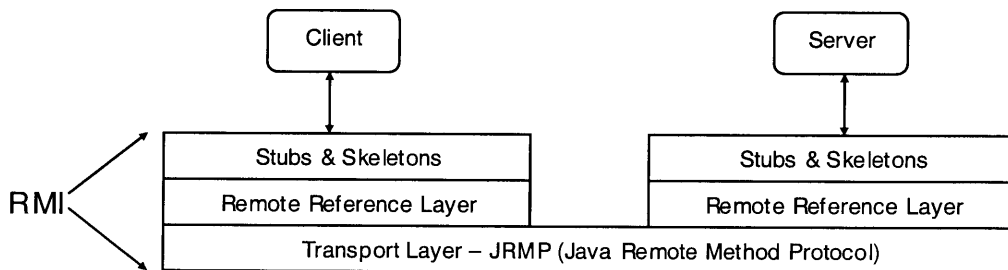


Figure 7-4 RMI

One of the shortcomings of both technologies, COM and CORBA, was interoperability. Meaning CORBA could not use COM objects and conversely – the same was true for RMI. Although, independent software vendors developed bridges to connect COM and CORBA; the addition was yet another integration point, adding to the cost and complexity of the solution.

The object technologies for remote object access had a number of additional disadvantages that hindered adoption. Towards the latter part of the ninety's HTTP

became the dominant protocol for remote access, neither DCOM or CORBA used HTTP. Encoding in DCOM was in NDR (Network Data Representation) and in CDR (common Data Representation) in CORBA while the software industry moved to XML (Extensible Markup Language).

Another barrier, neither object technology was cross-platform. Implementations remained with the supported operating system creating an operating system barrier between the technologies. Then there were practical issues like firewall friendliness. Both CORBA and DCOM were blocked by firewalls.

As time went on CORBA and COM we most successful as intranet implementations. And as the ninety's pushed to a close the industry moved toward HTTP architectures for remote computational objects.

7.3.2 HTTP Architecture Sample

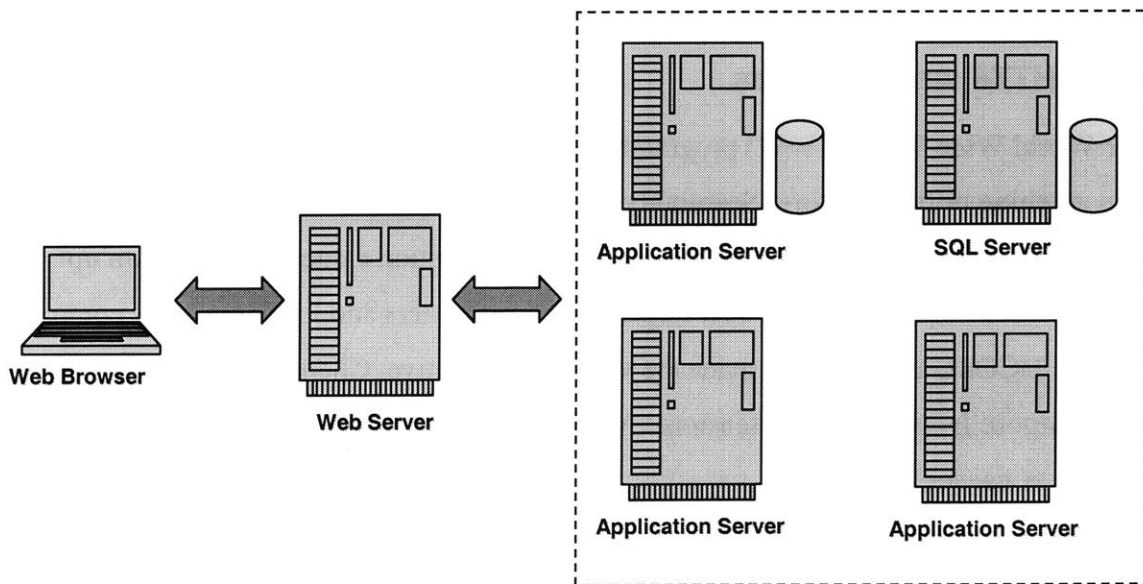


Figure 7-5 Sample HTTP Architecture

The most widely used clients are web browsers. Currently the most widely used browsers are the ones from Microsoft and Netscape. Both make requests on behalf of the user to the web server in the form of universal resource locator (URL) addresses. Web Server

side the URL identifies a resource to be served. The resource can be a static web page. The resource can be a dynamic page that is built on the fly and maybe tailored to the user based on his/her identity and the transaction context. The page being served may also source several another server(s).

The previous figure illustrates the architecture may be 3 tier or “n” tier. One of the most common setups is the 3 tier architecture where the web browser is the first tier, the web server is the second, and the database server is the third. The pages are dynamic and the data on each page comes from the database server. Other web applications compose pages from the information gathered from several servers. For example, a portal page like “My Yahoo” interfaces a weather server, a financial stocks server, a news server, and so on for every component of the portal page. The source servers themselves may have a multi-tier architecture to compose replies to the requesting web applications. Thus, serving a seemingly simple web page maybe the result of a complex process spanning several servers being orchestrated by the web server receiving the request.

7.3.3 HTTP Architectures

The World Wide Web (WWW) programming model has been adopted at a faster rate than anything prior in history. Seemingly overnight everyone could extend the largest distributed system today, the web. The exponential adoption came from a bottom up decentralized architecture. The low implementation barriers and loose coupling compared to RPC, DCOM, and CORBA made the web very attractive. Clients and server exchange Multipurpose Internet Mail Extensions (MIME) typed messages accompanied by a number of header fields.

The absence of centralized control and low technical barriers made it possible for corporations to build systems incrementally. Adding servers and clients to the web can all be done in decentralized fashion. Remarkably, given the scale, the variety, and the geographic location all systems can interoperate using the web protocols.

In contrast object serving and managing architectures had so many implementation challenges that most successful implementations only happened within the same organization (in-house). As discussed, examples of object architectures are: the Distributed Component Object Model (DCOM), and the Common Object Request Broker Architecture (CORBA).

In a departure from CORBA and DCOM the core idea behind web services is to follow the architectural decisions that made the web so successful and extend them to software objects we want to interface programmatically. Being able to reach and use an application programming interface with the same ease as reaching a document in the World Wide Web.

Web services are independent, stateless, and atomic. These properties much like the properties of http allow decentralized development and high scalability.

There are a tremendous number of software applications that are accessible through a web browser. Everything from online banking, ordering take-out, to radio station music streams. Unfortunately, these applications are not accessible programmatically and though there is some level of standardization in the industry, consuming a company's API requires a bilateral coordination that is prohibitive in terms of cost and negates scalability. Web services establish the minimum set of rules for interoperability.

Web services communicate using the Simple Object Access Protocol (SOAP) instead of mime messages [W3C-SOAP, 2003]. SOAP is a lightweight, extensible, XML based protocol for a decentralized, distributed environment.

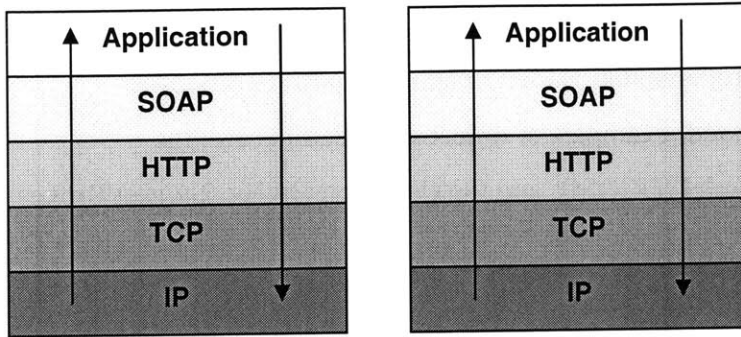


Figure 7-6 SOAP

SOAP can be used over the TCP/IP internet infrastructure. However, with the ubiquitous presence of web servers it is advantageous to use the protocol over HTTP. Looking at the image above, the message is routed and delivered using TCP/IP. Once at the destination host, the message is picked up by the web server, parsed, and passed to the application.

In more detail, SOAP runs over HTTP POST. In this sense, it is no different than any other web request which uses the POST method. For example, filling out and submitting a web form. The SOAP difference is additional information in the HTTP header sent to the server.

Alternatives to SOAP are HTTP POST and HTTP GET. HTTP GET has a number of disadvantages. The biggest is that all parameters passed to the server need to be appended to the URL. It is easy to discover web applications relying on HTTP GET while browsing the web. They are the ones with the really long URLs, with all kinds of parameters in the string. Due to the HTTP standard and browsers restrictions, the URL string length and content is limited. These restrictions limit both the size and type of data passed.

HTTP POST designed to pass value pairs to the HTTP server does not have length limitations. However, value pairs are not enough to pass complex data structures. Schemes can be designed to pass complex data and the necessary semantics for receiving host data structure deconstruction. The problem in devising schemes is the lack of standardization. Every designer would be solving the same problem over and over. What

is worse is interoperability would be extremely difficult due to the variability of solutions.

In contrast SOAP was designed for the purpose of exchanging semantically rich XML information. The standard specifies the protocol for remote method calls. The initial soap protocol is short and minimalist. The simplicity lowers the barriers of implementation. However, enterprises will still need additional services like security, routing, and transaction support. The SOAP protocol is extensible and services can be added as needed [Box, 2000].

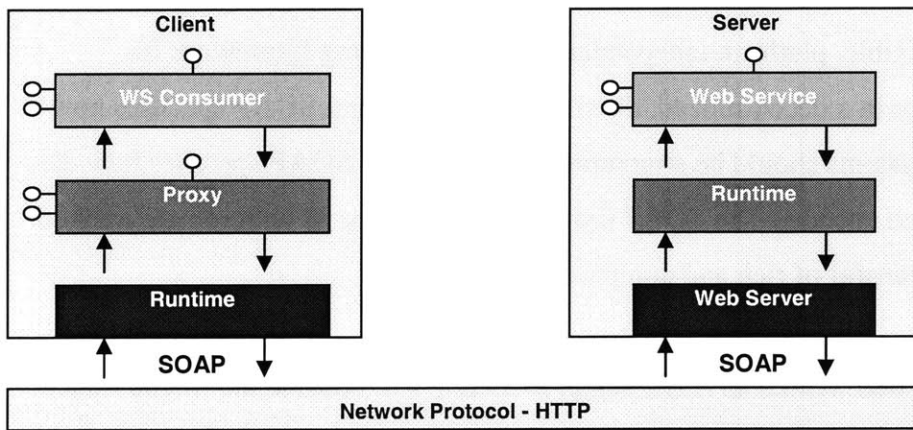


Figure 7-7 Web Service

To program against an available web service the client gets a web service description - more on the web service descriptions in later sections - and creates a skeleton of the service locally, called a proxy or stub. The programmer can then write against the web service using the local proxy. At execution time when the proxy is instantiated and the web service consumer client interfaces the stub, the data structure is serialized into an XML document by the runtime classes and sent to the server using SOAP over HTTP POST. The host's web server receives the SOAP message over HTTP POST. The xml message is deserialized and the data structure reconstructed using the runtime classes. The data is then sent to the web services methods.

To follow is discussion on the SOAP specification.

7.3.4 SOAP

Formally, from the SOAP 1.2 specification [W3C-SOAP, 2003]:

SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. SOAP uses XML technologies to define an extensible messaging framework, which provides a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.

SOAP provides a flexible, platform independent, XML messaging framework for information exchange in a decentralized, distributed environment. SOAP specifies how messages between systems should be structured and processed. SOAP specifies how data must be serialized and encoded. The SOAP protocol found a nice fit with remote method invocation and the transfer of rich and complex data types.

The SOAP standard was written as XML began to mature and become the lingua franca of interoperability of the software world. SOAP proposed a standardized way to convey messages and remote procedure calls between systems. A big positive of SOAP was its simplicity. The protocol was designed to be simple and extensible. The protocol did not include common components of distributed system such as: security, routing, transactions, and coordination. The resulting handshake between systems was greatly simplified and the learning curve flattened. Yet, the framework allows the missing components to be added incrementally as layered extensions.

SOAP is not transport specific. SOAP messages can be used over any transport protocol like SMTP for example. SOAP is flexible enough to allow the definition of new protocol bindings.

Although SOAP was initially envisioned for remote method access and later targeted to remote method access, the protocol is not tied to Remote Procedure Calls (RPC). The underlying model best fits a messaging system where SOAP specifies a model for

processing individual one-way messages. The composition, number of messages, and requirement of a response are all flexible. Stated differently, SOAP defines a message-processing model but not itself define any application semantics.

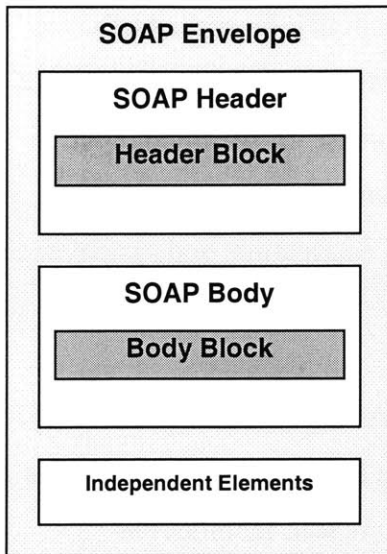


Figure 7-8 SOAP

The SOAP message is formed of three parts: the envelope, the header, and the body. The envelope is the top most or root element in the XML document that represents the SOAP message. The header is a generic container where additions can be made to SOAP in a decentralized way. The header allows you to place a payload header in the top of your SOAP message. The additional information is used to better describe the payload in the SOAP body. The body is mandatory information for the message receiver. In RPC, the body of a SOAP envelope is where the information for the remote method call is placed.

7.3.5 WSDL

SOAP provides a communication protocol to call remote methods and pass complex data types. XML Schemas describe the data structures but it does not describe the end points. The Web Service Description Language (WSDL) is an XML-based document that

specifies message end points using an extensible grammar that includes XML schema definitions (XSD) [W3C-WSDL, 2003]. Informally, WSDL describes the specifics on how to connect to the web service and its methods.

The Web Service Description language (WSDL) document, as the name implies, describes the web service to the consuming client. WSDL much like Interface Definition Language (IDL) in COM and CORBA defines the exposed interface. WSDL is a contract between the service and consumer promising a structured response on specified input. In contrast to CORBA and COM, WSDL was designed to be platform independent. The WSDL XML document can be consumed from any platform, application, and system and used to create a proxy of the service.

Technically, WSDL is not necessary to consume a web service. SOAP messages are self describing and specify data types and the values they contain. However, WSDL provides a standardized description that facilitates knowledge transfer. WSDL includes the supported communication protocols, location, encoding, methods exposed, data types involved. Once a WSDL document exists, a big plus is being able to write automated proxy generating tools to easily consume a web service without having to think about SOAP. None of which would be possible with out a standard.

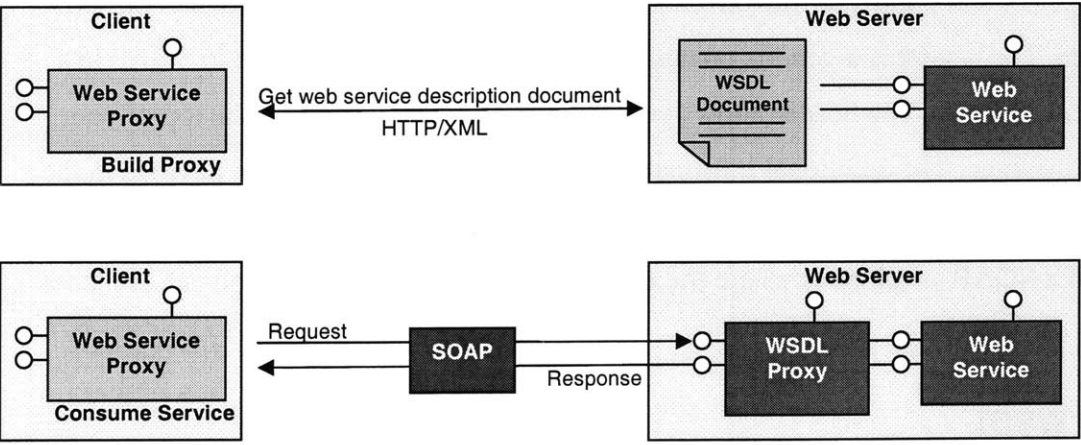


Figure 7-9 WSDL

The WSDL document can be divided into two sections containing six elements. The first group, abstract definitions also called the Service Interface Definition, has the Type, Messages, Operation, and PortTypes elements. The abstract section defines SOAP messages, in XML, in platform neutral manner. The second group, concrete definition also called the Service Implementation Definition, has the Bindings, Port, and Services elements. The concrete definitions specify how to create an instance of the service. The abstract and concrete sections are linked by the binding element.

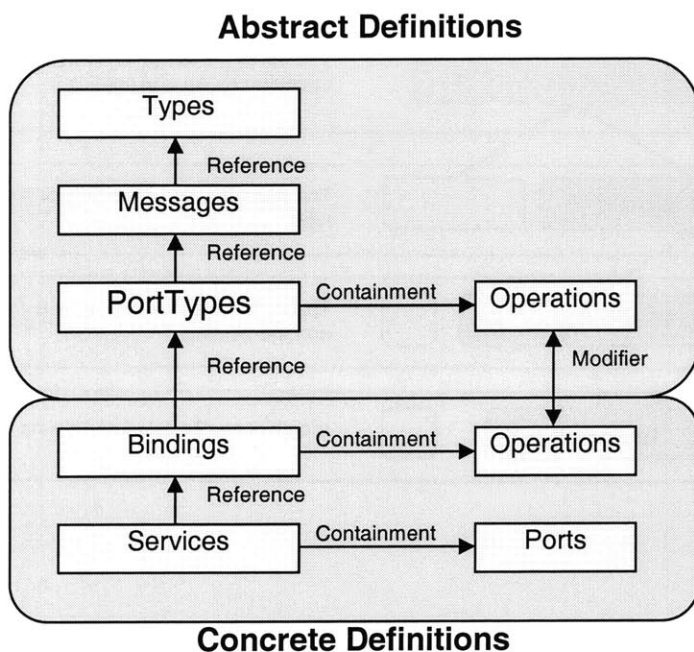


Figure 7-10 WSDL Description

Port types are analogous to application programming interfaces (API). A port type is a group of methods, called operations in WSDL speak, belonging to a service. To consume a method the client sends an input message containing the data for the interface. The reply comes in the form of an output message. Data items in the message are called message parts. The common protocols are SOAP, HTTP GET, and HTTP POST. The service protocols are specified in the bindings section. The hook(s) applications can grab onto is the service port(s). Ports list the network address and the protocol used with the port.

To follow is a diagram depicting a service that returns the server time, called Time Service.

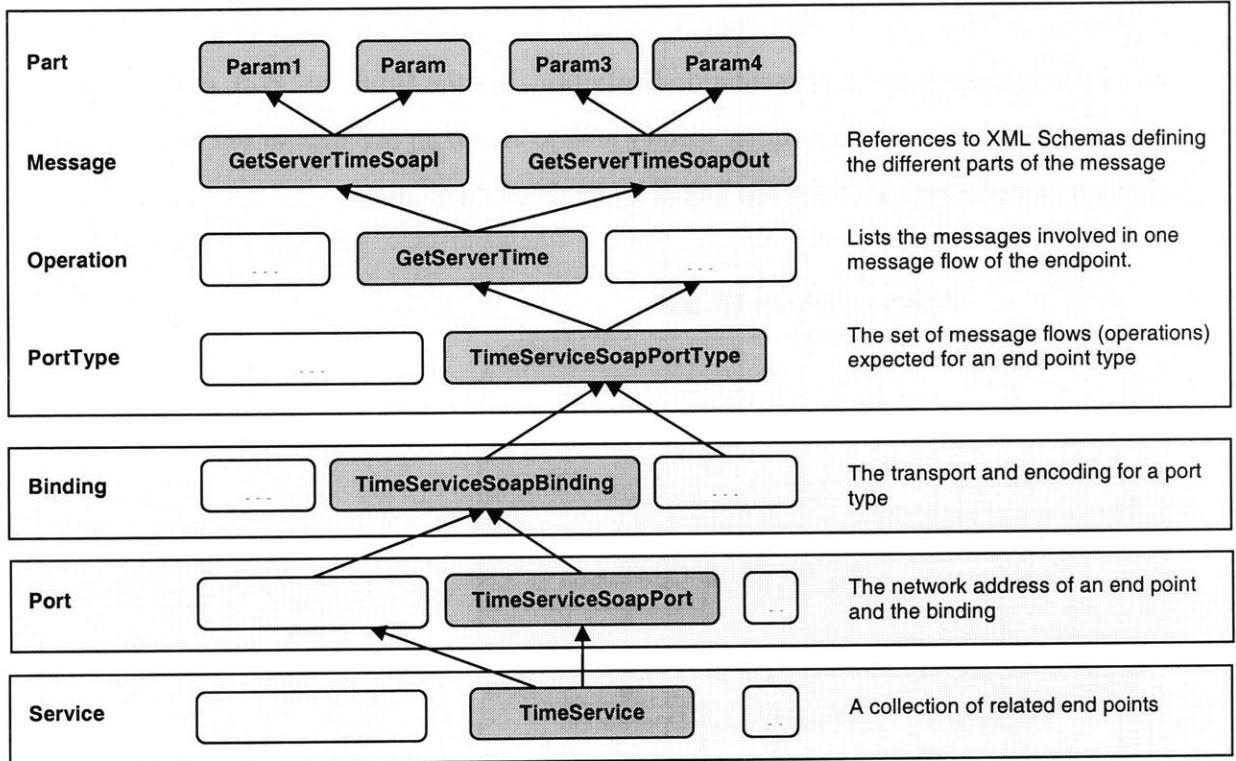


Figure 7-11 WSDL Sample

The schema for the time service

```
<s:schema targetNamespace="http://hostname/">
  <s:element name="GetServerTime">
    <s:complexType />
  </s:element>
  <s:element name="GetServerTimeResponse">
    <s:complexType>
      <s:sequence>
        <s:element name="GetServerTimeResult" type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="string" nillable="true" type="s:string" />
</s:schema>
```

The messages for the time service

```

<message name="GetServerTimeSoapIn">
  <part name="parameters" element="s0:GetServerTime" />
</message>
<message name="GetServerTimeSoapOut">
  <part name="parameters" element="s0:GetServerTimeResponse" />
</message>

```

The port types for the time service

```

<portType name="TimeServiceSoap">
  <operation name="GetServerTime">
    <input message="s0:GetServerTimeSoapIn" />
    <output message="s0:GetServerTimeSoapOut" />
  </operation>
</portType>

```

The bindings for the time service

```

<binding name="TimeServiceSoap" type="s0:TimeServiceSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <operation name="GetServerTime">
    <soap:operation soapAction="http://tempuri.org/GetServerTime"
      style="document" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>

```

The service definition for the time service

```

<service name="TimeService">
  <port name="TimeServiceSoap" binding="s0:TimeServiceSoap">
    <soap:address location="http://hostname/time/timeService.asmx" />
  </port>
</service>

```

Chapter 8 Discovery Services

The Universal Description Discovery and Integration (UDDI) protocol is becoming increasingly important as the adoption of web services scales up [UDDI, 2003]. “UDDI creates a standard interoperable platform that enables companies and applications to quickly, easily, and dynamically find and use Web services over the Internet. UDDI also allows operational registries to be maintained for different purposes in different contexts.” As defined by OASIS in www.uddi.org.

As more interfaces move to a web services model there is an opportunity for domains to standardize interfaces to be implemented by service providers. For example, universities can publish a lecture interface for the listing of lecture notes, lecture materials, and lecture discussions. The lecture interface can then be registered with UDDI. Then every university offering lecture notes resources can implement the interface and register with UDDI using the publishing services. Content creators can then search UDDI for universities that implement the lecture interfaces. The search is done programmatically using the inquiry application programming interface (API). Operations on UDDI are done programmatically and built into the application logic to remove the burden on the end user. Interfaces to UDDI are web services themselves and interacting with the interface is done with XML documents.

Organizations operating public UDDI registries replicate content among the operators. For this reason it is only necessary to register with one UDDI operator as the interfaces are propagated to all UDDI registries. Some specialized domains implement specialized registrars to facilitate registration or add value to the default interface.

There are a number of UDDI servers available for implementation within an organization. Large organizations have hundreds of departments with software components. UDDI servers may take place of in house solutions to register and reuse components. The standard can help large organizations with a diversity of solutions to

standardize component registration and discovery. To follow are application examples of UDDI.

Consider a scenario where you need mathematics tutoring services and would like to find the services in your city. You could search UDDI for a list of the services in your geographic region. Once you have selected a service you could then schedule a session. In this sense UDDI is providing a matching service. Likewise once you find a service provider you can explore other services by searching on the provider's name.

UDDI can also offer service location resolution for address changes. Service consumers have no service interruption since the protocol can transparently obtain the new URL, contained in UDDI, as part of the same service invocation.

Where as most web services are data pull models, coupled with UDDI, web services can be used in a push model. Typically, web service consumers execute and pull data from the service provider. Using UDDI, the service provider can publish an interface to be implemented by all the service consumers. Once the interface has been implemented; the service provider can push data to each implementation for a twist on data delivery. The variation of the model accommodates subscription models. An application of the subscription could be made to distribute educational content.

Many content creation companies, educational institutions, and governments have become interested in the Massachusetts Institute of Technology (MIT) open courseware (OCW) offering. However, at the time of writing there was no programmatic delivery or publishing. The previously described subscription scenario would fit OCW very well. OCW could publish an interface to UDDI and all interested organizations could implement the interface. MIT could then dynamically poll UDDI for all the implementers and push OCW content to them. In this model MIT does not have to manage subscriptions; the management is done by UDDI.

Organizations that see the need can also become an additional node on the UDDI fabric. UDDI information is propagated through all nodes through replication. Thus, uploading information need only be done in one place and it propagates to all existing UDDI nodes. In this sense UDDI is like DNS being logically centralized and physically distributed. In UDDI's case the propagated information linked to an organization name is software access contracts instead of domain names and internet protocol numbers.

To follow is a discussion on the data model and extensibility of UDDI.

8.1 Data Model

The core data structures of UDDI are: business entity, business service, binding template, and something called tModels [UDDI, 2003]. The business entity has a business description, unique handle, category, and a pointer to further information.

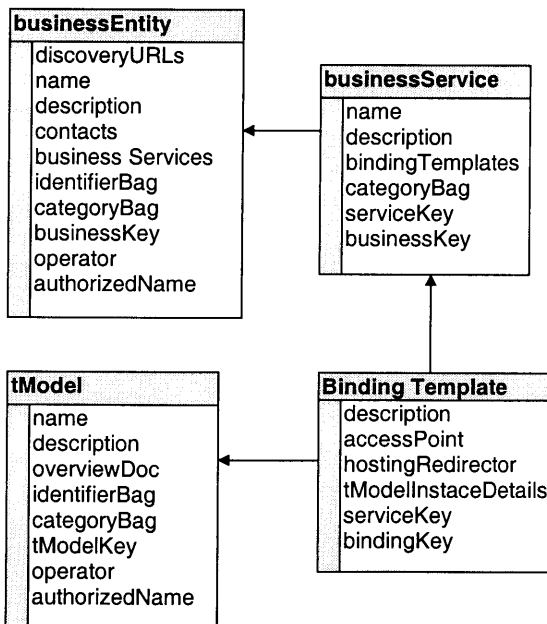


Figure 8-1 UDDI Basic Data Model

Organizations joining UDDI start by creating a business entity, see below for a sample. The business entity entry is differentiated by a unique key and can have a number of

description documents referenced by discovery URLs. Identifier and category bags are used to classify the entity and are optional.

Each business entity is linked to 0-to-N business services. For example, the popular online retailer Amazon can create a business entity and then create and link book services, electronics services, toys services, and services for numerous services under the online brand. As software niches become commoditized the organization of services, the ease with which new services are integrated, and the ability to derive new services becomes part of the strategy for software providers. UDDI was designed to be very generic and business services are not limited to web services. The service can be anything like a simple HTML page or a fax number. Business services, like business entities, can also be classified using an identifier and category bags.

Business services are mapped to binding templates for the technical implementation. The binding template most important elements are the access point and the redirector. The access point is the address of the service location. If the access point is not given a redirector must be provided. The redirector points to another binding template. The additional layer of indirection is useful to relocate services and to permit other services to have different metadata yet link to the same binding template.

The technical model (tModel) is a very flexible entity. The tModel is a generic pointer to metadata outside UDDI. As such, the key attribute, the address, is contained in the overview document attribute. The address can point to web service metadata, a text file, an AutoCAD file, or anything that is addressable.

In the case of web services, tModels point to a document describing the service interface. The document does not need to be a web service description language (WSDL) document but it must contain the metadata describing the service. The document describes the service but has no information about the endpoint. The endpoint, access point, information is contained in the binding template.

TModels are also used to extend UDDI with metadata. The UDDI designers used a commonly used technique is database design to make extensible tables. The technique involves creating a table with a variable for new attributes and another for the attribute values. To avoid attribute uniqueness the attributes are mapped to a tModel key as shown in the figure below. In this sense the tModel serves a similar role to namespaces where definitions and declarations are cast in a predefined context.

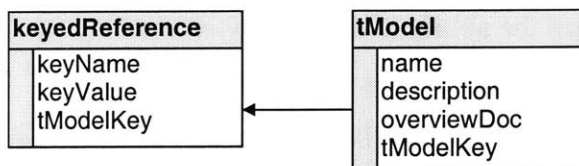


Figure 8-2 tModels

Each binding template can link to many tModels. Likewise, each tModel can link to many binding templates. The tModel instance information (tModelInstanceInfo) table is a mapping for the many-to-many table links. A sample including keyed references is shown below.

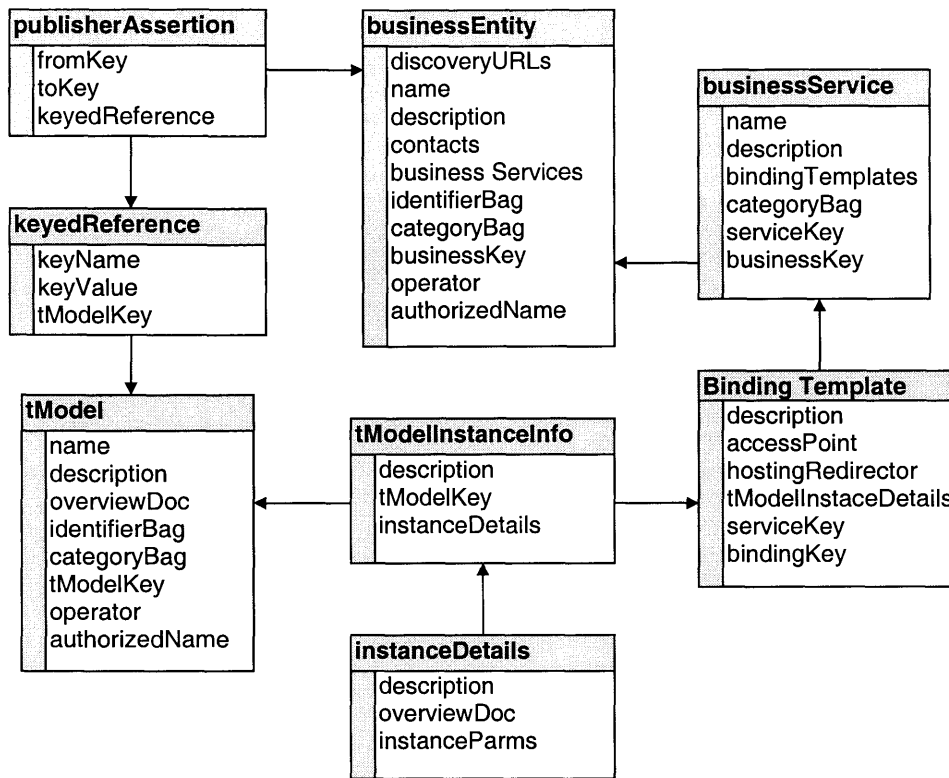


Figure 8-3 UDDI Data Model

As the UDDI server scales there is a need to relate business entities. The entities might be members of the same organizations that have entered UDDI separately or independent entities that have a logical relation. To relate two business entities the owner of each entity must create a publisher assertion. A bilateral assertion is necessary to avoid undesired business entity relationships. The publisher assertion table contains three fields: from key, to key, and keyed reference. The first two are pointers to the two entities. The third points to value pair defined in the context of a predefined UDDI relationship tModel.

Below is a comprehensive UDDI data model diagram using the uniform modeling language (UML) format.

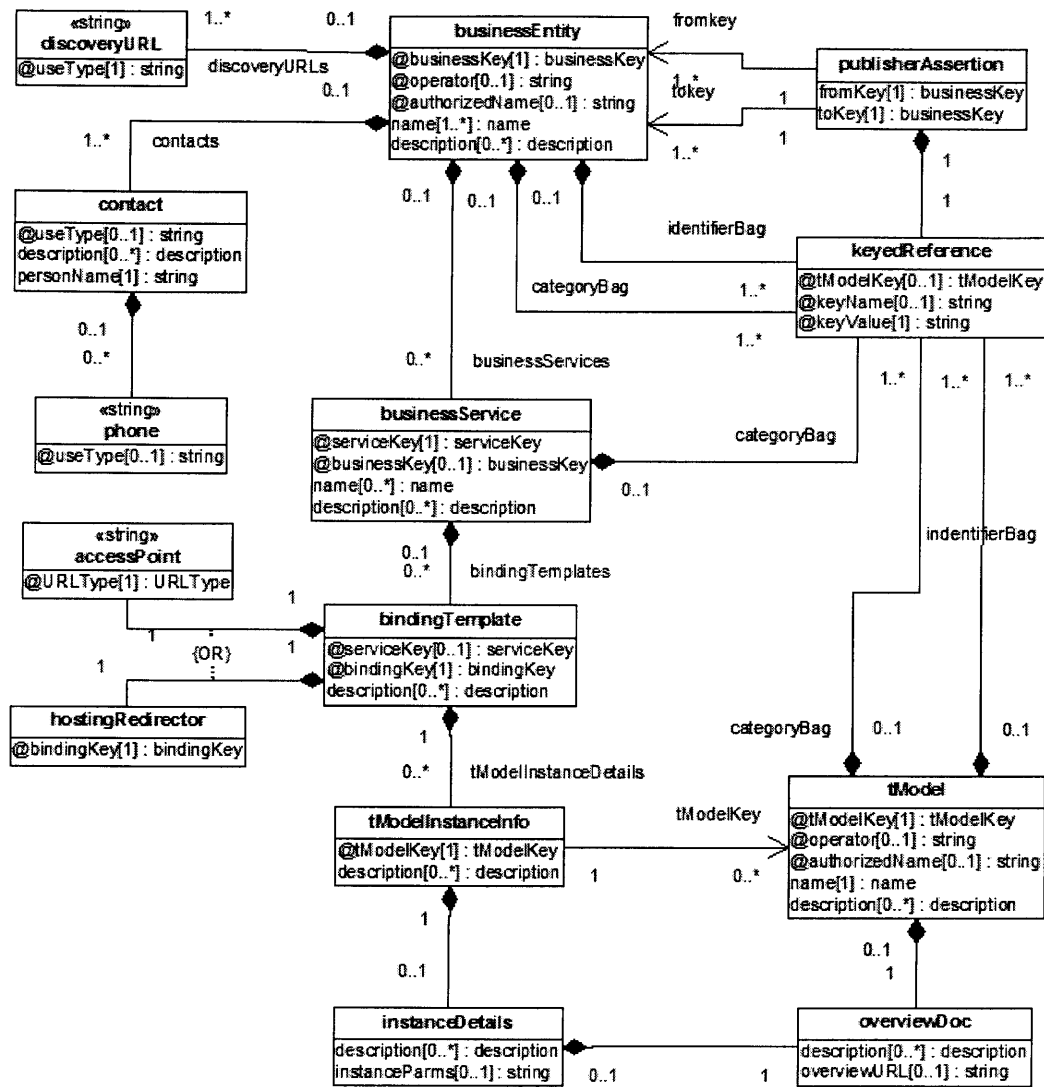


Figure 8-4 UDDI Extended Sample (Source msdn.microsoft.com)

In the sample below MIT has setup a UDDI server and a registered a business entity with business services. The business entity is an academic portal entry as a handle for academic portal services. Note that the discovery URL points to local UDDI server in the MIT domain.

The listed business service is for course documents. The business service has two binding templates. The first exposes course lectures notes and the second course assignments. The binding templates point to files that act as the web service interface. The proxy built by the service consumers then post data to the address given in the access point tag. The

tModel in turn points to the metadata address. The metadata holds the information needed to create the proxy. Thus, the binding template tells the service consumer where to send the data and the tModel how to format the data to be sent.

```

<!-- tModel for the Course Lecture Notes Service -->
<tModelDetail>
  <tModel
    tModelKey="uuid:55555555-5555-5555-5555-555555555555"
    operator="MIT"
    authorizedName="Abel Sanchez">
    <name>Course Lecture Notes Service</name>
    <description>
      This interface is to be implemented by services that return course
      lecture notes.
    </description>
    <overviewDoc>
      <description>WSDL document</description>
      <overviewURL>
        http://portals.mit.edu/AcademicPortal/CourseLectureNotes.asmx?WSDL
      </overviewURL>
    </overviewDoc>
    <categoryBag>
      <keyedReference
        tModelKey="uuid:01010101-0101-0101-0101-010101010101"
        keyName="Specification for a web service described in WSDL"
        keyValue="wsdlSpec"/>
      </categoryBag>
    </tModel>
  </tModelDetail>

```

```

<!-- tModel for the Course Assignments Service -->
<tModelDetail>
  <tModel
    tModelKey="uuid:66666666-6666-6666-6666-666666666666"
    operator="MIT"
    authorizedName="Abel Sanchez">
    <name>Assignments Service</name>
    <description>
      This interface is to be implemented by services that return course
      Assignments.
    </description>
    <overviewDoc>
      <description>WSDL document</description>
      <overviewURL>
        http://portals.mit.edu/AcademicPortal/Assignments.asmx?WSDL
      </overviewURL>
    </overviewDoc>
    <categoryBag>
      <keyedReference
        tModelKey="uuid:01010101-0101-0101-0101-010101010101"
        keyName="Specification for a web service described in WSDL"
        keyValue="wsdlSpec"/>
      </categoryBag>
    </tModel>
  </tModelDetail>

```

```

<!-- Business Entity entry -->
<businessEntity
  businessKey="00000000-0000-0000-0000-000000000000"
  operator="Massachusetts Institute of Technology"
  authorizedName=" Abel Sanchez : 86">
  <discoveryURLs>
    <discoveryURL useType="Home Page">
      http://portals.mit.edu/AcademicPortal/training/
    </discoveryURL>
    <discoveryURL useType="businessEntity">
      http://uddi.mit.edu/discovery?businessKey=00000000-0000-0000-0000-
      000000000000
    </discoveryURL>
  </discoveryURLs>
  <name>Academic Portal</name>
  <description>Academic Portal - An intance of the Academic Portal type in the
  Portal Factory</description>

```

```

<!-- The Course Content service -->
  <businessService
    serviceKey="11111111-1111-1111-1111-111111111111"
    businessKey="00000000-0000-0000-0000-000000000000">
    <name>Course Content</name>
    <description>Course documents</description>

    <bindingTemplates>
<!-- first binding: Course Lecture Notes Service -->
      <bindingTemplate
        serviceKey="11111111-1111-1111-1111-111111111111"
        bindingKey="22222222-2222-2222-2222-222222222222">
          <description>Course Lecture Notes Service</description>
<!-- The service's endpoint URL -->
          <accessPoint URLType="http">
            http://portals.mit.edu/AcademicPortal/CourseLectureNotes.aspx
          </accessPoint>
          <tModelInstanceDetails>
<!-- reference to the tModel describing the Lecture Notes interface -->
            <tModelInstanceInfo
              tModelKey="uuid:55555555-5555-5555-5555-555555555555">
                <description>Course Lecture Notes tModel</description>
                <instanceDetails>
                  <description>SDK document</description>
                  <overviewDoc>
                    <description/>
                    <overviewURL/>
                  </overviewDoc>
                  <instanceParms>
                    http://portals.mit.edu/AcademicPortal/SDK
                  </instanceParms>
                </instanceDetails>
              </tModelInstanceInfo>
            </tModelInstanceDetails>
          </bindingTemplate>

<!-- second binding: Course Assignments Service -->
      <bindingTemplate
        serviceKey="11111111-1111-1111-1111-111111111111"
        bindingKey="33333333-3333-3333-3333-333333333333">
          <description>Course Assignments Service</description>
<!-- The service's endpoint URL -->
          <accessPoint URLType="http">
            http://portals.mit.edu/AcademicPortal/CourseAssignments.aspx
          </accessPoint>
          <tModelInstanceDetails>
<!-- reference to the tModel describing the Course Assignments interface -->
            <tModelInstanceInfo
              tModelKey="uuid:66666666-6666-6666-6666-666666666666">
                <description>Course Assignments tModel</description>
                <instanceDetails>
                  <description>SDK document</description>
                  <overviewDoc>
                    <description/>
                    <overviewURL/>
                  </overviewDoc>
                  <instanceParms>
                    http://portals.mit.edu/AcademicPortal/SDK
                  </instanceParms>
                </instanceDetails>
              </tModelInstanceInfo>
            </tModelInstanceDetails>
          </bindingTemplate>

    </bindingTemplates>
  </businessService>
</businessEntity>

```

The naming of the data structures are colored by the era of conception, the internet boom. As used in this work, business entities are the provider of the service(s). The provider is the contact point for service operation. The business services are the web services associated with a provider. The web service endpoint is the binding template. The endpoint holds the URL for the web service address. Lastly, the tModel is metadata statement with a URL address to the web service description language document.

Chapter 9 Product Platforms

Platform research has been leveraged in many fields for gain [Meyer,1997; Baldwin, 2002]. This section discusses product platforms, product families, and derivate products. It addresses the design and manufacturing of information products. It investigates how information technology can be used to support the creation of new products, create the manufacturing platforms, and modularize the product design. It proposes an architecture for information products and it addresses the strategic and technical implications of the architecture.

Towards this goal, this chapter presents insights from platform research. In parallel will be an effort to apply platform research to software information architectures. In this work, software information products are software defined products used to manipulate information. The term information product is used interchangeably with that of software information product. The application of the architecture goes beyond information products since it is generic. However, the focus of this work is on products that consume and manipulate data. This chapter extends the work of Mark H. Meyer's, ""The Power of Product Platforms"" [Meyer, 1997], to the academic grid platform.

The information product is defined as a collection of web services. For example, a weather analysis product maybe comprised of a temperature, wind metrics, pressure, humidity, and rain gage web service modules. The chapter on the portal factory expands this topic.

The targeted consumers of the information products are external as well internal consumers. The information products definition of sale varies according to domain. For example in research and academe sales are taken as product adoptions since frequently the products are public domain. In this case, the use/implementation of an information product by an individual(s) constitutes a sale.

9.1 Architecture

The identification of commonalities in product construction and the inception of shared abstractions layers is the architecture of a product. Variations in the used commonalities will produce different architectures and different functionality. Architectures can also be developed to maximize a sought dimension. For example, constructing a product favoring high performance will result in an architecture where all design decisions are made favoring performance. Likewise, different architectures will result in different product functionality, cost, and quality. An architecture focuses the product developer on a pre-selected design path. Effective architectures gain adoption by product developers and are a basis for productivity. At the same time, the design concept of the architecture and the focus it provides, necessarily constrains the variety of product versions that can be offered.

Important to effective architectures and design is modularization [Meyer,1997; Baldwin, 2002]. Breaking down a product into modules is a principle of design that divides a mechanical system or structure into standardized elements. The modules can then be reused by other systems. Once the interface has been standardized, the modules can also be replaced by newer modules that perform the same task in a perhaps lower cost, higher performance, or more robust manner. Interchangeable modules can also be paired in new ways. Making modules interchangeable with standardized interfaces has the added benefit of being able to pair modules together in new and different ways to create new products [Ulrich, 2000].

Products are composed by many subsystems. For example, an automobile has numerous subsystems like the steering system, the cooling system, and the breaking system. Each of which can be further broken down into components. The granularity of the components, the interfaces, and targeted reuse become competitive dimensions for the product. A grouping of subsystems and components can be grouped and used as base to build new products. The set of subsystems and interfaces that form a common structure from which a stream of derivate products can be efficiently developed and produced is a product platform [Tabrizi, 1997].

The overall design concept, the sequence of design decisions, the process of dividing the product construction into modules, and creation of abstractions from functional layers is presented as the product architecture.

9.2 Black and Decker - Architecture Renewal Example

One of the flagship examples of architecture renewal is the Black and Decker (B&D) Company [Meyer & Lehnerd, 1997]. Back in the nineteen seventies Black and Decker's product architecture efforts had many problems. The architecture had little commonality between products groups and even between them. Each product was designed and redesigned one at a time. The product design did not leverage the design of existing products. Similarly, products were not designed and modularized so modules could be reused. A standout example of the product design approach was that B&D had over 100+ different motors, no common design, and different production lines. As the company sought to improve its inefficiencies in product creation it underwent an architecture renewal effort.

B&D designed a new architecture of all tools – drill, sander, saws, hedge trimmers, etc. One of the key components identified for commonality was a motor that could be used by several tools. The new architecture featured a new universal motor that could be used for all tools. The motor had a power range of 65 watts to 650 watts. A similar approach was taken with all key subsystems. The results were striking.

Under the new architecture B&D saw a 50% reduction in costs to produce products. The convergence to fewer reusable modules resulted in higher quality components and more durable products. The products were also lighter.

B&D experienced no restraints on product variety. The opposite effect occurred, common subsystems enabled variety. B&D product variety actually increased over time. B&D

advantages drove much of the competition out of business. B&D market share went from 20% to 80%.

9.3 Product Platform

The design concepts comprising a product's architecture are physically implemented as "platforms" for the product and for the process to produce it [Meyer & Tertzakian, 1997].

The platform is the grouping of product or process designs that are used as a base to derive products. The platform can be thought of as a framework that churns out new products. The components and design of the framework are the architecture while the physical implementation of the architecture is the product platform.

Platform "effectiveness" can be measured in terms of successful derivative products or market attractiveness. The first is the measure of the stream of new products derived from the same platform. The second is the measure of the product attractiveness to the chosen markets.

For information products, the basic structural dimension of the repository is the web service. The web service defines the atom of functionality to be stored, retrieved, and manipulated.

The product platform of information products is a repository of web services. The repository becomes the foundation from which families of information products will derive. Each web service is a subsystem module that can be used to compose information products. The richness of the pool of web services reflects the potential number of products that can be made. The modularity of components permits focusing resources of key modules that will be shared by a large number of products. The level of complexity, granularity, and the structure of the repository are design constraints. The more ways the repository can be sliced and diced, the greater the potential to create new products.

The definition of products will be a listing of the paired modules. The loose coupling of web services is an ideal property to create flexible products that can even be defined on

the fly. The product platform is comprised of three stages: acquisition, refinement, and storage/retrieval.

The acquisition stage is done using the web services description language (WSDL). Computing resources are exposed using the Hypertext Transfer Protocol (HTTP) and described using WSDL. Consumers of computing resources can hook onto the resources and create stubs. As the platform comes together there will be a pool of web services, representing computing resources, available. The loose coupling not only provides flexibility but also creates a way to easily perform platform renewal through module updates, removals, and new additions.

The refinement can vary from pairings of web services to wrapping web services in modules for presentation – graphic user interface – delivery. Certain modules will be consumed server side and not have a presentation element while others will need a presentation layer.

The platform web services will need an interface to store and retrieve modules. The Universal Description Discovery and Integration (UDDI) protocol creates a standard layer to quickly, easily, and dynamically find and use web services over the internet. It contains an industry supported standard for service description and discovery using XML and HTTP. Cross platform programming is supported using the messaging Simple Object Access Protocol (SOAP).

9.4 Process Platform

The process platform as the name indicates consists of the processes used to manufacture a firm's products. The technologies, the assembly, the staging of manufacturing, the testing, the physical housing of the infrastructure, the integration of outsourced modules are all potential parts of a process platform. Depending on the industry the process platform may be in a high rate of continual renewal. For example, computing devices

parts, casing, and software change so rapidly that OEMs strategy needs to plan for a continual renewal of process platforms.

Process platform investment is about more than high product throughput, it is an additional opportunity for innovation to facilitate greater product variety. Process platforms can be designed and built to enable product variety. The advantage, or lack of there of, comes from the flexibility built into the production process. Such allowances expand the production possibilities. In the absence of flexibility, the variety of products is constrained. Thus, care should be taken to not construct a process platform that not only has high volume and capacity but also one that is flexible and will welcome product variety.

For information products, at a high level, the process platform consists of: acquisition, refinement, storage/retrieval, distribution, and presentation. Acquisition of the web services, the first step of product construction, gathers the base structural units upon which the platform will be build and products based on. Issues to address are quality, reliability, robustness, scalability, security, control, exclusivity, and timeliness. Some of these issues like robustness are technical, while others like exclusivity are determined by the policy, license, or contract. Even when all the computing resources are self-owned verification of module robustness, security, and scalability is needed.

Once the web services have been secured and tested the next step is to refine the resources. Not all web services will be built to needed specifications, many will need to be extended or wrapped for a better fit into the product design. Other will have to be paired to get the need functionality. Most web services will need to be integrated into a common software foundation for the platform. The software foundation is the computational context addressing things like identity, authorization, authentication, culture, and language. Refining not only adds value adapting web services for consumption in the platform but also structuring the pool of resources in a flexible and logical manner to facilitate product variety.

Storage and retrieval represent a transition between the product platform and the distribution and presentation. The platform structural elements, web services, need a strategy for storage and retrieval. An organization scheme is needed plus an interface for searching plus a registry process for new modules that are available to the platform. Interfaces for web services pool can be best leveraged if the interfaces are standardized. The software industry identified the need to access computational resources in a standardized way to leverage the efforts being replicated in many software houses. Led by IBM and Microsoft the Universal Description Discovery and Integration (UDDI) specification was proposed to define and discover web services. The specification is based on internet standards. It is platform and implementation neutral. And it has gained buy in from a majority of the industry leaders. UDDI is a natural choice to create a web service registry to be exposed for the platform consumption. UDDI can help to quickly and dynamically discover the sought after module.

The form in which a product is delivered to users represents the distribution. The distribution channels and types can spider into several different products and different business models. To take an example, a newspaper can be delivered to a mobile phone in a reduced format charging per packet much like the NTT DoCoMo business model. The same newspaper can be distributed through a web site, to be consumed by a web browser, on a monthly subscription basis. The distribution of the information product also considers the timing and the frequency.

The last stage in the process is the presentation. The presentation of information products refers to the graphical user interface delivered to the end user. Quality, usability, and transparency as well as aesthetics are all dimensions for differentiating products in the market place. A powerful, robust, and scalable process with poor presentation will negatively affect the attractiveness of the product. In domains where competitors' information content and process is similar, the presentation effectiveness will become a competitive edge.

9.5 Product Families

An effective platform gives way to an array of new products based on the same platform at a fraction of the cost. Product families are the streams of related products. In our previous example of architecture renewal, families refer to products that share a number of key modules and subsystems. Similarly, a product family is defined as a set of products that share a common technology and address a related set of market expectations [Meyer & Tertzakian, 1997]. The commonality shared between different products permits the company to devote more resources to the common elements. Higher investment in the common modules leads to more testing, higher robustness, effectiveness in manufacturing, distribution, and service.

The product platform, the repository for information products, the process platform, and the refinery are the infrastructure to derive product families. As discussed in previous sections the repository serves to create the web services pool, extend the services, integrate the services, and create a registry of modules consumption of the platform. The process platform is presented as five stages: acquisition, refinement, storage/retrieval, distribution, and presentation. Using the product and process platform are the infrastructure to modularize and integrate internal and external web services for information products and derive product families.

9.6 Product Platform Evolution

If platforms do not evolve they will eventually become obsolete. This section discusses transitions states in platform evolution. The transition events are: new platform generation; new platform; and platform renewal.

The initial platform architecture consists of common subsystems and interfaces for an array of products.

A new platform generation takes place when the interfaces between modules remain constant but the modules and subsystems change. Internally the modules might be doing

something very different but the interface to other systems continues to have the functionality. Other modules consuming the interface would continue to function properly. The modules would change but since the interfaces did not change the platform would continue to function into the new generation.

Platform extensions are a new generation where the number and type of subsystems remain constant but where the subsystems and interfaces are enhanced.

A new platform occurs when a new architecture, new modules, and interfaces are introduced. Modules and interfaces may be reused from prior platforms and paired with the new components in the new architecture. The new platform is not simply an extension but a completely new approach, a new concept, and a new design.

9.7 Technological Leverage

This work will discuss two types of leverage: technological leverage and market leverage. Technological leverage is the extent to which investments in the basic platforms serve as a foundation for efficiently developing derivative products. This section introduces the basic measures for leverage; later sections cover metrics in detail.

One of the first measures is that the cost of the derivative product needs to be lower than the cost of the platform development plus the research and development. The measure is a fundamental first check to answer the question “is the platform making any difference?” The measure is a viability test of the platform. If a platform can not pass this simplest of tests then the platform has failed to produce any savings and should be discarded or redesigned. Having passed the test, the cost of derivative products should be significantly lower than the platform creation cost. The goal of platforms is to reduce the cost of derivative products and in this pursuit the lower the cost of derivatives the more promise for the platform.

Another measure is platform efficiency defined as the engineering cost of underlying platforms divided by the average engineering cost of product derivatives. Given the ratio, values approaching one, or under one, are poor efficiencies. Values close to one would mean the price of derivative products are costing just as much as the cost to create platform. Conversely, the larger the efficiency number the more positive and efficient the platform. High efficiency numbers means the cost of derivative products is significantly lower than the cost of the platform. As discussed previously, there needs to be a periodic investment in the platform to maintain its viability. For this reason, values should not go to infinity as there needs to be continual investment into the platform. The investment should be reflected through cost of derivatives. Values that are abnormally high may indicate a platform that is outdated and will become obsolete.

The third measure to be discussed in the context of technological leverage is cycle time. Products are only useful if they are available when needed. A product that has great efficiency but unavailable in time can not be used. Cycle time is the time required to create a derivative product divided by the time it took to create a platform. Sample values of derivative products can be in the months while the platform maybe in years, varying accordingly by industry.

For information products the ability to rapidly and efficiently engineer new products is based on the in product platform. The number of web services in the platform will have a direct impact on the products that can be made. A good stock of web services is necessary to enable product variety. Other dimensions are the way the web services are packaged, the structure of the module registry, and the forms and levels of access the module pool.

9.8 Market Leverage

The market leverage is the return provided on the platform investment. The return maybe measured differently depending on the organization. For example, in academe and research the measure maybe the number of product adoptions by academic groups. More

often the return will be measured in terms of sales. The accrued sales for a set of derivative products divided by the total engineering costs of those products is defined as the platform effectiveness [Meyer & Tertzakian, 1997].

The discussions of the last sections have led to leverage. The leverage that a platform can give when creating new products. Evolving product families are based of product and process platforms. Leverage is gained when robust efficient platforms are constructed that produce successful product families in short cycles times efficiently.

9.9 Distinctions from Physical Products

The time to take products to market is much shorter when compared to physical products [Sundgren, 1995]. There is a tremendous pressure to get products to market in the fastest time possible. Especially in the interest of speed the importance of a platform can not be overstated. Although the time to build the platform may take longer than to build a single product, once the platform is in place products will be derived quickly. An efficient platform will permit the derivation of products at a much faster cycle time than when constructed independently [Wheelwright, 1992].

Once an efficient, effective, low cycle time platform is built the construction of new products is very fast. In many cases the new products can be created on the fly on demand. Likewise, the incremental cost is very small.

On some cases the creation of platform interfaces that can also be leveraged by external developers and systems can lead to market domination. When a platform clearly exposes and documents the interfaces there is an opportunity to become the market standard. How much market share, developer friendliness, and developer tools are some of the dimensions that win developers [Meyer, 1998].

From a physical product perspective the platform can become a channel of distribution. As external developers leverage the platform, the number of products offered expands

with out any additional cost to the platform maker. The more products developed on a platform, the larger the number of users and visibility of the platform. Several products with varying degrees of success have attempted to win developers over to a platform focusing on software abstractions like data, presentation, or language. For example the Oracle Corporation, at the time of writing the world's second-largest software company, established itself as a relational database management system (RDBMS) platform. RDBMS platforms have become so common place that product manufacturers leverage them routinely for the data layer of the software product.

The modularity of information products permits mixing and matching to create new products. Products originally created for web site distribution can be repackaged for mobile phone distribution. Similarly, the content from several sources can be combined to create a new product or the content from one source distilled to provide a different offering.

One more way in which information products differ from physical products is in the frequency of purchase. Information has a shelf life after which it must be renewed or replaced. Periodic purchases of information products will be a common model for consumers.

9.10 Barriers for competition

Exclusivity of the data interface or data would limit competitors. Even though exclusivity of data is the strongest position, in the case where data can be equally obtained by all, securing the lowest cost channel would be an advantage over the competition.

As the module pool grows and grows the size and the management of the repository will become a barrier to competitors. New competitors will not have the stock of modules and will need to manage and optimize their repositories as they grow.

Developing a platform creates barriers to entry for the competition. One of the first is product variety and cycle time. As discussed in previous sections a well designed product architecture allows the rapid development of products. Companies creating products one at a time will be at disadvantage with a longer product cycle and less variety.

When competing with other platforms, the technology used in the platform can be another barrier to entry. Technologies to refine, categorize, and structure the module pool with create value. Similarly, ease of use, flexibility, and extensibility will facilitate content renewal and product variety at lower costs.

9.11 Platform Conclusions

The design approach in general is to design/consume small modular pieces with the goal of creating an underlying platform where new modules can be snapped on at a low cost. The plugin framework is the platform that will be leveraged to create product families and in some cases used as a market standard, a shared framework.

Well designed platform architectures can rapidly construct families of derivative products at a low cost. The software platform is something to be carefully managed to provide flexibility through modularity and standardization. In some cases, the interfaces themselves may become a market standard enabling the company to become a channel of distribution. In any case the flexibility and customization possibilities are an added benefit.

Key to the platform architecture is the basic unit of the module pool. In this work the basic information unit was the information web service. The basic unit was coarser than on a text feed but it was also more modular and the grouping and handle more obvious. The interface to the module pool was done though the Universal Discovery Description Integration (UDDI) registry for web services.

The market advantages, as discussed, are many and can be used as a business model. Using a platform approach products have a timelier introduction to market. The product family will be richer and more flexible. The scope in the market will be broader. Finally, competitors with out an equivalent robust product line will face a barriers to entry.

Chapter 10 Portal Factory

Grid computing has many advantages as has been discussed in prior sections. However, for information products there are a number of shortcomings. The literature is missing a home or container for services. Designers need a way to define applications (portal types) quickly and inexpensively. The presentation meta-layer to the pool of computational resources is unspecified. This chapter presents a stateless version of the grid's abstract framework for communication.

For information products the connection to computational resources is short lived and stateless connections are sufficient. Removing the stateful requirement for connections simplifies the architecture and makes it more scaleable. The resulting application is very much like the web. Web servers responding to web services requests can scale very much in the same manner web servers scale today. The difference is quite significant as the Hypertext Transfer Protocol (HTTP) has scaled globally

10.1 Introduction

The portal platform is built upon a pool of web services. The web service forms the basic architectural unit for portal construction. Everything from infrastructure components like user registration to custom components like a weather module are constructed and integrated as a web service. There is no distinction in the construction process between a custom module that displays the top ten movies of the week and an infrastructure module like registration. Both modules can be loaded and leveraged dynamically by the portal factory.

The imposed architectural mandate is that every component must truly encapsulate functionality, execution, and logic. The component does not have a priori knowledge of when, how, and in what context it will be used. Therefore, the logic, functionality, and execution can not be dependent on computing resources that are not contained within the component itself. The designer of the component is forced to design a component no with

dependencies. The restrictions of the programming model enable a long sought after goal, reuse of computing resources. The restrictions are no more than a stricter adherence to the object oriented programming principles.

Once web services are constructed they are registered with the Universal Description Discovery and Integration (UDDI) Server. Computing resources constructed locally or remotely expose their interface, their usage contract, using UDDI. The UDDI server as discussed in previous sections is a platform that enables organizations to quickly, easily, and dynamically find and use web services over the Internet. The UDDI protocol enables the registration, publication, and implementation of the web service. The UDDI protocol is open standard, XML based, interoperable, and accessible over HTTP. The UDDI contract is a web service contract.

Web service implementations and development environments have been implemented in most computing platforms. The open source group has Apache and Axis, Windows has the .Net Framework, and there are implementations by IBM, Sun, Oracle, and Hewlett Packard. One of the big draws to the development paradigm is that much like the HTTP architecture the architecture of web services is open, simple, and the protocol is human readable XML documents. The technical barriers are low compared to component technologies of the past, like the Common Object Request Broker Architecture (CORBA) and the Component Object Model (COM). The core infrastructure needed is a HTTP server to listen for XML documents and an XML parser. Considering web servers and XML parsers have become ubiquitous in today's computing infrastructures; the cost of leveraging components across operating systems and computing platforms has been lowered and simplified much in the same manner that HTTP made linking to documents much simpler than the days before the World Wide Web (WWW).

10.2 The Portal Infrastructure

The portal infrastructure implementation is based on the Common Language Infrastructure (CLI) ECMA standard. The programming language used is C#, also part

the ECMA standard. For more information on the ECMA CLI and C# standards see <http://www.ecma.ch/ecma1/STAND/ecma-335.htm> and <http://www.ecma.ch/ecma1/STAND/ecma-334.htm>. On the Windows platform, the .Net development platform from Microsoft is based on ECMA standard. The Mono Project sponsored by Ximian Inc. is an open source Linux/Unix version of the Microsoft .Net development platform. Like Microsoft .Net Mono is a complete development platform including a Common Language Infrastructure (CLI), virtual machine, just-in-time (JIT) compiler, a garbage collection runtime, C#, and a class library. There is also Rotor, a shared source implementation of the same ECMA standards. Both Mono and Rotor offer full source code for their implementations.

The same computing architecture approach can be taken on other computing platforms. The reference to ECMA is mentioned for completeness and context for the reader. The architectural discussion of the portal factory is made generically.

The portal factory infrastructure is a framework in the classic sense composed of a collection of patterns. The context of the framework and patterns is collaborative portal applications. Examples are:

- Problem identification
- Informing and knowledge exchange
- Events
- Evaluating alternatives
- Choice
- Implementing
- Review/feedback
- Coordination
- Socialization
- Motivation
- Negotiation

In spite of the context, the granularity of the platform is such that modules, like a weather component, are written independently and can be leveraged by any application in need of such a component.

The portal infrastructure data model is composed of portals, component instances, component definitions, users, and roles. The portal infrastructure also contains a rendering component, a module loader, a portal loader, the portal context, satellite assemblies. In the sections to follow the infrastructure components and data model are discussed.

10.3 Users

The management of registration, authentication, and authorization has become increasingly important as organizations try to integrate the constantly expanding number of heterogeneous systems that make up the enterprise. The complexity has grown with distributed systems and the internet highlighting the need for security, delegation, and federation.

There are a number of solutions and protocols on the market for the management of user identities. Some of the major offerings are the Lightweight Directory Access Protocol (LDAP) which has been adopted in enterprise products by Microsoft, IBM, Hewlett Packard, Sun, and an open source flavor called OpenLDAP. Each major platform also has a server to manage identity. Examples are Microsoft's Active Directory and the open source Kerberos Server. Microsoft, with a web service solution, is making a big push with the "Passport" service. Users need only register once and they can login to every application, service, or web site using Passport without having to reregister. Sun is backing a similar open source service called the "Liberty Alliance". Not to be outdone America Online (AOL) has announced "Magic Carpet" with the same goals. All three services seek to provide registration, authentication, and authorization. For hosts, the advantage of signing on with an identity manager is automatic access for every user

subscribed with the service – Passport had more than 200 million users at the time of writing. Users also get the benefit of only remembering one username and password.

With the variety of software applications, platforms, protocols, and operating system management of identity has become increasingly complex. The portal infrastructure uses a web service solution. Given the model’s required abstraction and encapsulation, the registration service can also be used for authentication on other systems. A simplified data model is shown in the figure below.

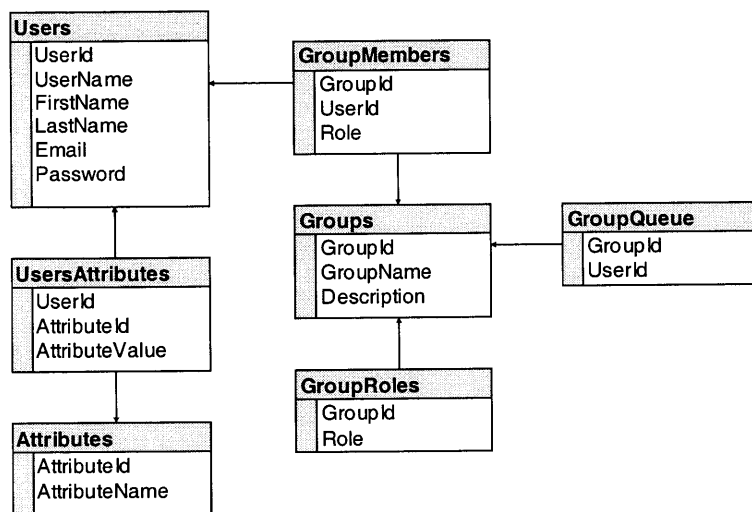


Figure 10-1 Users Data Model

The data model is presented in an entity relationship (ER) diagram for ease of discussion. However, the reader should keep in mind that the model is not exclusive to a Relational Database Management Systems (RDBMS). The ER model can just as well be implemented in XML using the Schema Definition Language (XSD).

Applications need to be able to create groups, assign roles, and extend the registration attributes. In the given data model groups are created in the “Groups” table and the group’s roles in the “GroupRoles” table. Group members are added by role with a separate entry for every role. In a course portal a faculty member may have a grader role, a lecture author role, and an announcement author role.

No matter how many registration fields a user table has new applications always have a need for new ones. For example, a portal for structural engineers may need a field for design manual that would have been missed by the most comprehensive user registration tables. A way to deal with an unknown number of fields at design time is to create a table of attributes to be defined on the fly as new portals are created. The attributes can then be mapped to users with a data relationship.

A common extension to the given model is a table of actions in which each role is mapped to a number of actions. The read lecture, edit lecture, author lecture, and delete lecture can be mapped to the faculty role in course portal. One of the advantages of this approach is lower granularity of authorization. For example, to allow all users with the student role to have the faculty action of “authoring announcements” the administrator can selectively add the author announcement action to the student role. If the logic was solely dependant on roles, the student role could not be given partial actions of the faculty role. The users with the student role could be given the faculty role but it would have to be done on a one-by-one basis and what is worse they would receive the full actions of the faculty role. To achieve the partial authorization, the logic of the affected components would have to be modified to allow both the faculty role and the student role to author announcements.

There are lots of interfaces that can be written to manipulate the data model. Below is sampling of interfaces (for the WSDL interface see the appendix).

- AddUserToGroupWithGroupId
- AddUserWithUserIdToGroupWithGroupId
- ChangeUserPassword
- CreateAccount
- CreateGroupRole
- CreateUserGroupWithGroupName
- DeleteUserFromGroup
- GetRolesInUserGroup
- GetUsersInGroup

- GetUserAttributes
- GetUsersAttributes
- GetUserAttributesByUserId
- GetUserGroupFromGroupId
- GetUserIdFromUsername
- GetUsernameFromUserId
- GetUserPicture
- GetUserRoleInGroup
- GetUsersFromGroupWithThisRole
- IsUserInGroup
- LogIn
- LogOut
- ListGroups
- ListGroupsUserIsIn
- LogInWithoutSiteId
- SaveAttributeValue
- SaveUserPicture
- UpdateGroupByGroupId
- UpdateUserRoleInGroup

10.4 Rendering

The number of target panes supported by the portal infrastructure has no limit. However, the implementation has standardized on 5 panes: header pane, footer pane, left pane, right pane, and content pane as shown on the image below.

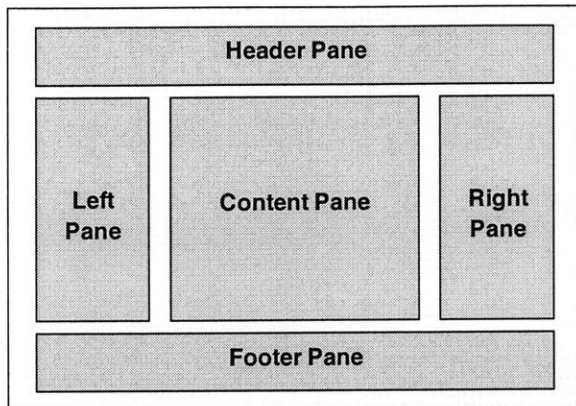


Figure 10-2 Rendering Panes

The addressable panes can always be extended by the user. The addressable panes can also be decreased, changed, deleted, or completely recreated.

At this point it is worth having some discussion on rendering components.

A rendering component executes server side, provides a programmable object model, and renders markup text to a web browser. The component is a fundamental building block of the web application portion of the architecture.

The component object provides the basic functionality for participating in the portal page framework. In particular, it provides the functionality that allows a component to be placed within the component collection as a page is rendered. Another useful layer of abstraction is to uniformly handle rendering HTML content. The underlying base class can provide support for styles through properties such as font, dimensions, and color.

Components are reusable application logic that involves a modularized solution to a problem context. Components are namespaces that contain classes with methods and properties that can be reused. One of the advantages of components is they increase the rendering speed of dynamic web pages. Because components are compiled into machine code they execute faster than pages that are interpreted at rendering time.

Components promote robustness. A component can be shipped as a black box that performs a function. Consumers of the black box can leverage the component and need not worry about the inner workings.

Components can also be used to divide web application functionality into a number of smaller, self-contained chunks of code. Using these methods can help web applications become more scalable and easier to manage.

Below is the pseudo code for a class of html images, the number of properties have been abbreviated for convenience. All components are written as stand alone modules and know how to render themselves.

```
public class WebImage
{
    private String src;
    private String alt;

    public String Src{get{return src;} set{src=value;}}
    public String Alt{get{return alt;} set{alt=value;}}

    public String render()
    {
        //render HTML
    }
}
```

Much like the logic and HTML generation have been packaged into the “WebImage” class above, bigger blocks like a calendar or a contacts list can also be rolled into a component. The resulting module maybe have a number of classes and supporting methods but the rendering logic can treat the module like any other object that knows how to render itself. The objectification of modules enables very flexible rendering options.

Using reflection, the framework can load components written by remote contributors and dynamically populate pages in the rendering cycle. The next section discusses reflection and its role in rendering portal pages.

10.5 Reflection

Recent programming languages like Java and C# have been built to be self-describable. The description information is contained in what is called metadata, data about the data. Metadata describes structures, enumerations, classes, method's contracts, referenced types, as well as defined types within a component or application.

Metadata is used to enforce type-safety by comparing the executing code versus the self-description, an important check for robustness. Metadata is also used to discover and query components dynamically, called reflection.

Reflection is the ability to discover information and query the metadata about a type at runtime. By reflecting over an application or component it is possible to find what types, methods, and interfaces it has. Furthermore, applications can be written to dynamically load and instantiate components using reflection. For example, you can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object. You can then invoke the type's methods or access its fields and properties.

Using reflection the portal infrastructure has developed a rendering cycle that loads components dynamically, instantiates them, and targets a pane(s). The computation performing the rendering does not need to know the components a priori at compile time. This means that what it loads, where it loads it from, how much it loads, when it loads, and the graphical user interface target are variables determined at runtime. The flexibility is such that the loaded components could be determined based on user input at runtime.

A less apparent benefit is the capability of loading federated components. The software boundaries and imposed framework treats components at arms length imposing a strict separation on the logic, execution, and functionality of the component. The separation negates the necessity to centrally control the development of components. This property of the architecture is very important. With it, components can be constructed by anyone and still be leveraged by the portal factory framework. This means developer groups and

individuals anywhere may contribute components to the portal infrastructure. The image bellow depicts a scenario where components have been constructed by many different groups, loaded dynamically at run time, and rendered targeting graphical interface panes.

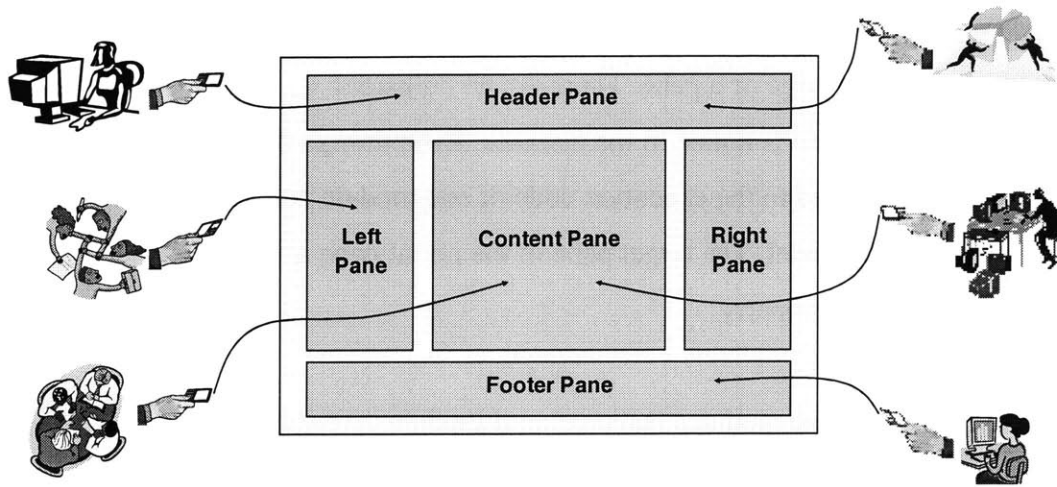


Figure 10-3 Late Binding Using Reflection

10.6 Portal Data Model

The portal factory data model is composed of modules, module definitions, pages, and portals as shown below.

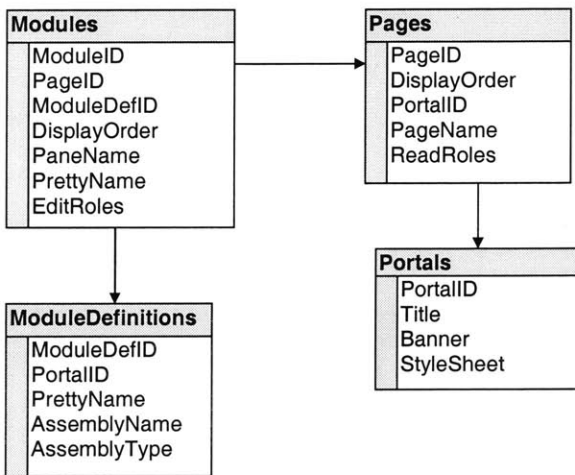


Figure 10-4 Portal Data Model

Modules are defined by specifying the path, the name of the type to instantiate, and the portal the modules are to be listed under. It is that modules definitions are unique with in a portal instance. Meaning there can be modules that only appear in some portals.

Each instance of a module is unique and linked to a page. There can be many instances of a module even with in one page or a pane. There are no collisions since each instance is unique and its associated data is linked to the instance. Even though there are many instances they are all mapped to the execution code of one module and the mappings handled through the data model. The target pane in the portal page is dynamic and handled through a module property.

The number of pages just like modules instances and module definitions is dynamic. Each page is linked to a portal and is unique in the portal framework. Pages are the containers for modules. The panes are the graphical interface targets for the modules with in each page.

In practice there is often the need to have the same module appear on many pages or appear on a page more than once. An example maybe a banner image or a menu that need to be shown on all pages. The data model can be extended as with a mapping table to map modules to pages.

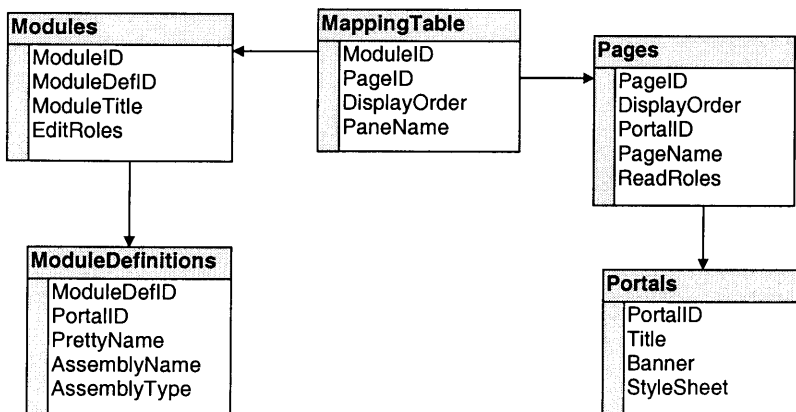


Figure 10-5 Model Pages Mapping

In the extended data model shown above, a module can appear in as many pages as needed. The same module can also exist in different panes in each page, multiple times on a page, and multiple times on each pane. Below is a sample set of data for the mapping table.

Modules			
ModuleID	ModDefID	Title	EditRoles
010	100	Menu	Author
011	101	Links	Author
012	102	Html	Author
013	101	Links	Author
014	102	Html	Author

MappingTable				
ModuleID	PageID	PaneName	DisplayOrder	
010	000	Content	00	Page 000
011	000	Content	01	
012	000	Content	02	
010	001	Content	00	Page 001
013	001	Content	01	
014	001	Content	02	
010	002	Left	00	Page 002
010	002	Content	00	
010	002	Right	00	
010	002	Right	01	
010	002	Right	01	

Pages				
PageID	DisplayOrder	PortalID	PageName	ReadRoles
000	00	1000	Home	Reader
001	01	1000	About	Reader
002	02	1000	Empty	Reader

Figure 10-6 Mapping Sample

10.7 Language and Culture

Distributed on the World Wide Web (WWW) software, small and large, has a global reach. On the web, the barrier to global distribution is lowered to post access to a HTTP server. However, software localization to the myriad of possible cultures on the world requires design time consideration. The image below shows a small sample of existing cultures.

Cultures and Languages		
Afrikaans	Chinese - Hong Kong SAR	Finnish - Finland
Afrikaans - South Africa	Chinese - Macau SAR	French
Albanian	Chinese - China	French - Belgium
Albanian - Albania	Chinese (Simplified)	French - Canada
Arabic	Chinese - Singapore	French - France
Arabic - Algeria	Chinese - Taiwan	French - Luxembourg
Arabic - Bahrain	Chinese (Traditional)	French - Monaco
Arabic - Egypt	Dhivehi	French - Switzerland
Arabic - Iraq	Dhivehi - Maldives	Galician
Arabic - Jordan	Dutch	Galician - Galician
Arabic - Kuwait	Dutch - Belgium	Georgian
Arabic - Lebanon	Dutch - The Netherlands	Georgian - Georgia

Arabic - Libya	English	German
Arabic - Morocco	English - Australia	German - Austria
Arabic - Oman	English - Belize	German - Germany
Arabic - Qatar	English - Canada	German - Liechtenstein
Arabic - Saudi Arabia	English - Caribbean	German - Luxembourg
Arabic - Syria	English - Ireland	German - Switzerland
Arabic - Tunisia	English - Jamaica	Greek
Arabic - United Arab Emirates	English - New Zealand	Greek - Greece
Arabic - Yemen	English - Philippines	Gujarati
Armenian	English - South Africa	Gujarati - India
Armenian - Armenia	English - Trinidad and Tobago	Hebrew
Azeri	English - United Kingdom	Hebrew - Israel
Azeri (Cyrillic) - Azerbaijan	English - United States	Hindi
Azeri (Latin) - Azerbaijan	English - Zimbabwe	Hindi - India

Figure 10-7 Cultures and Languages

Localization goes beyond language. A truly global application needs to be culture-neutral and language-neutral. Though language is the cornerstone of localization, culture needs to also address the writing system, calendar type, date/time formatting, sorting rules, currency and numbering conventions. The most flexible solutions create a culture abstraction

Modern computing frameworks like Java and .Net have built culture abstractions to facilitate localization. Although these abstractions are helpful there is still graphical user interface text and user input to consider.

It is important to have perspective on culture and the multiple layers that come into play as an application executes. At the base is the operating system with a culture specific code page. Calls to the operating system will return code page specific values. Likewise the relational database system and the virtual machine will have code pages that are independent from the operating system and as such may differ. Database code pages are particularly important since lookups, queries, and record sets will be ordered according to the culture rules. The diagram below illustrates the layers of a sample web service.

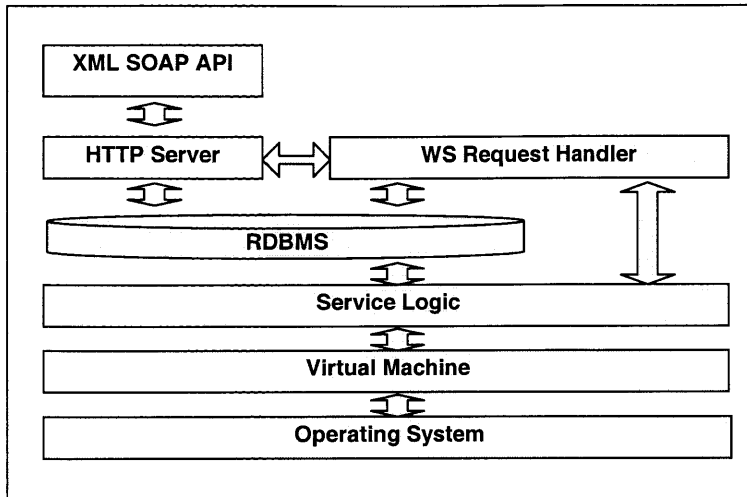


Figure 10-8 Culture Layers

In this work the culture is delegated to the virtual machine underlying the executing code. The database code page is set to the local culture in each case. And the user interface language is dynamically loaded conditional on the user profile. A data sample follows:

```

<languageFile>
  <langStr>
    <varName>contactLang</varName>
    <en-US>Contact</en-US>
    <es-MX>Contacto</es-MX>
    <ja-JP>□ □ </ja-JP>
  </langStr>
  <langStr>
    <varName>editLang</varName>
    <en-US>Edit</en-US>
    <es-MX>Corregir</es-MX>
    <ja-JP>□ □ □ □ □ </ja-JP>
  </langStr>
  <langStr>
    <varName>nameLang</varName>
    <en-US>Name</en-US>
    <es-MX>Nombre</es-MX>
    <ja-JP>□ □ </ja-JP>
  </langStr>
</languageFile>

```

Using data structures like the sample above the user interface language can be set on the fly giving each user the language of choice. Translation to new languages can be done with a simple grid text editor like the one below.

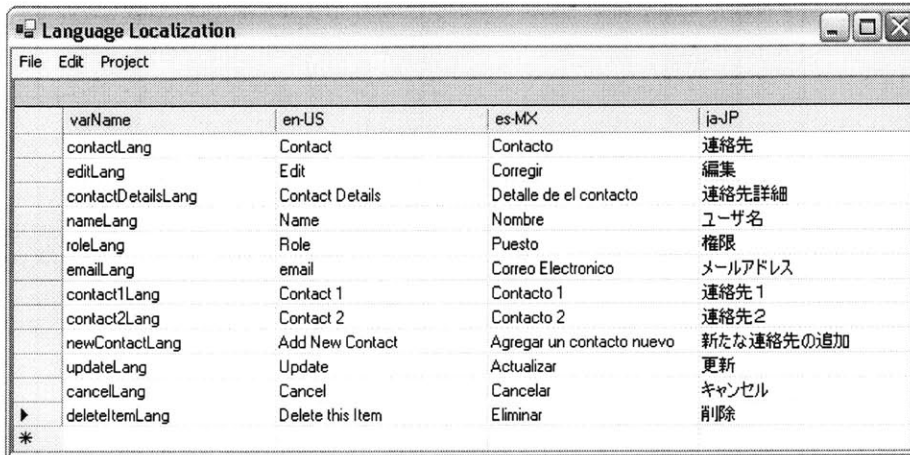


Figure 10-9 Language Editor

The translation is only for the user interface text and text that exists at the time of the portal construction. The reader should note that applications have evolving text that is contributed by several users in collaborative applications. Translation of user contributed text is outside the bounds of this discussion.

10.8 Portal Architecture

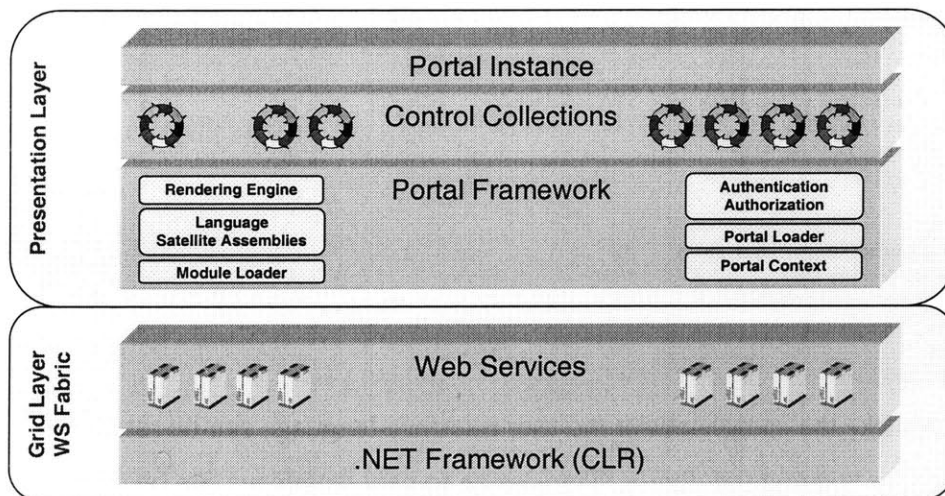


Figure 10-10 High Level Portal Architecture

The common language runtime (CLR) lies at the base of the application. In an analogous manner to Java the CLR acts a virtual machine. As mentioned previously, the underlying

platform and operating system are variable. Successful tests were made on Apache/Axis and Ximian/Mono.

The web services layer holds the pool of services available to assemble information products. A service oriented architecture virtual fabric of web services. The fabric is constructed on the services that solve holistic information product's problems. The glue between threads is the overarching applications and roles. The difference to traditional glue is that it behaves more like Velcro. A completely new fabric can be constructed by rearranging the threads. See the image bellow for an illustration.

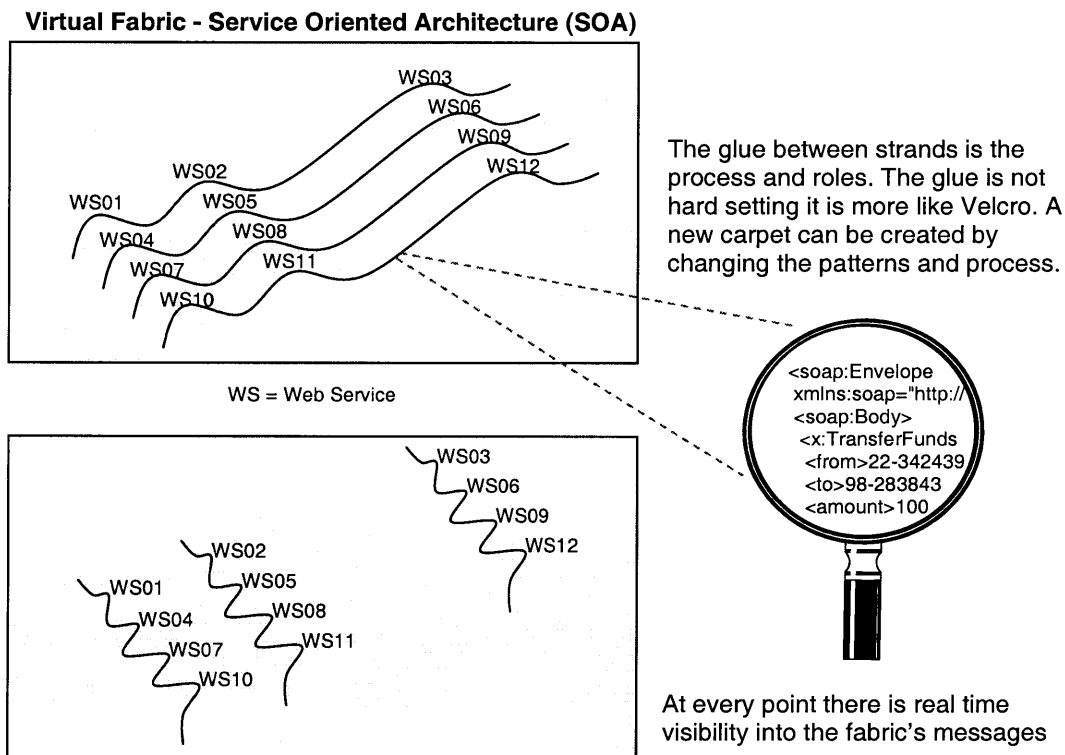


Figure 10-11 Virtual Fabric

The portal framework provides the computational context for the portal. The portal composition is completely dynamic. That is, the portal does not exist anywhere until a request is made for it. Each time a user requests a portal the portal is composed on the fly and personalized for the user. It is important to note that not only does the portal not exist before each request but that it goes out of existence once the request has been processed

and a response has been created. So every request is completely independent and stateless. The portal context is the glue of the application. The context of the portal composition is the language/culture, security, and the rendering engine loader.

Based on the user profile, the user interface language and culture is selected on each portal request. On anonymous requests the portal returns the default language and culture. The portal can load any language and culture as long as the definitions have been made in the data model and the language strings translated.

A control, also called component or module in this work, is a unit of software that knows how to generate a user interface for itself. Controls are well encapsulated and developed independently from the portal. The autonomy of the control is advantageous in a distributed development environment where programmers can create controls remotely and deliver the unit as a dynamically linked library to be accessed by the rendering engine. Controls also function as a front end to web services. Each control is a client to one or more web services grouping the required functionality for a portal element. For example, the bookmarks control is a client to the bookmarks web service. When the bookmarks control is instantiated it goes out and retrieves the bookmarks for the portal from the bookmarks web service.

To preserve design simplicity the module designer is encouraged to construct simple modules and manage complex needs by grouping modules. Much in the same way as service oriented architectures address complex needs by grouping web services and maintaining each web service as simple as possible. The advantages are maintainability, modularity, and reuse.

The modules registered with each portal page are dynamically loaded on each request. A portal request is really a request for a portal virtual page. Each portal page is composed of a header, footer, style sheet, and a module collection. When the portal request arrives at the server, the loader looks up the modules registered on the portal page. The next step is to get a handle on the modules and load them into the page. Each module in turn contacts

web services and its own data model to bind the data corresponding to this module instance. The module instances themselves need unique identifiers to differentiate a module instance from instances on other pages and from multiple instances on the same page.

The rendering engine loops over the module collection, targets panes on the request page, and asks each module to render itself. Once the rendering engine has a handle on the module collection it targets the defined panes on the requested page.

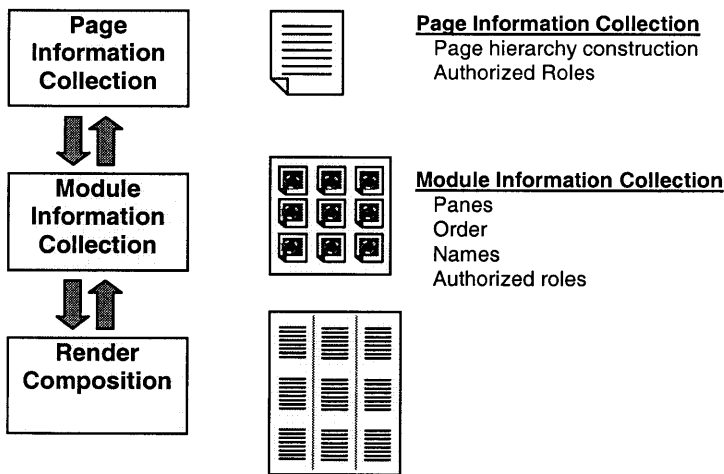


Figure 10-12 Rendering

The steps outlined are a high level description of the instantiation of a portal. The diagrams below present two distinct phases of the portal construction. In the first stage, once the portal framework context has been computed, the handshakes to each web service fire, the services are instantiated, and handles propagated. The second stage illustrates the connection between web services and page rendered.

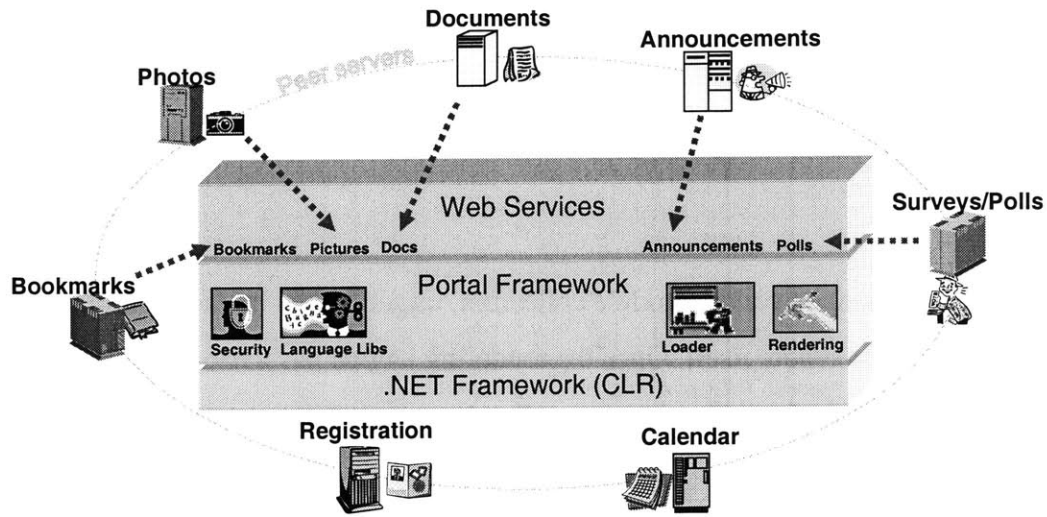


Figure 10-13 Portal Composition

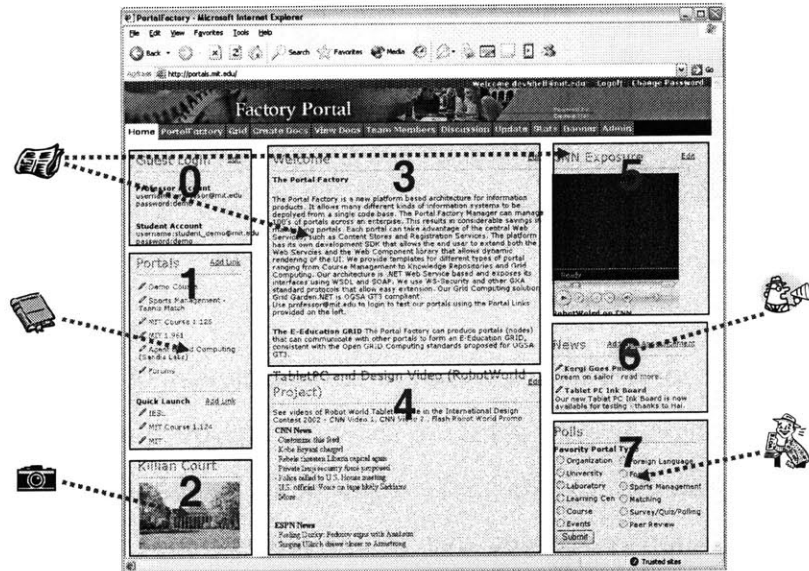


Figure 10-14 Portal Rendering

10.9 Going Multi-Portal

The dynamic architecture of the portal has proven beneficial for the reuse of computational resources. However, the speed with which new applications could be

constructed led to the problem of maintenance. Quickly there were many portals custom built to support a number of collaborative efforts. Shortly after the individual portal constructions the code sets began to diverge. Keeping track of the changes, patching, and leveraging new code became difficult as the number of independent portals increased. So, while it was possible to quickly compose applications there was no planned structure for portal evolution. A multi-portal abstraction was sought.

The multi-portal abstraction needed to run multiple portals off the same code base. Since all portals execute of the same code, any updates or additions would be global and instantaneous. Each instance needed to be independent and have variable paging, modules, format, security, and culture. The portal context needed to be independent for each portal. For example, user's permission and roles in one portal needed to be striped as users moved from one portal to the next. The same issues needed to be addressed with the rest of the portal context.

The portal data models were revised and iterated to arrive at the multi-portal solution. See the portal data models in previous sections to see the end result.

10.10 Portal Types

As more and more portals were built, portal types started to emerge. Early samples were course portals, team portals, and discussion forum portals. A new type abstraction was added to represent portal types. Types were referenced each time a new portal instance was created. Custom portals could still be created but every time a new pattern was spotted a new portal type could be defined.

Portal types enabled portal creation automation. Portals could now be created on demand. The difference was very noticeable. There was a ten fold increase in the number of portals. The portal automation cycle is illustrated below.

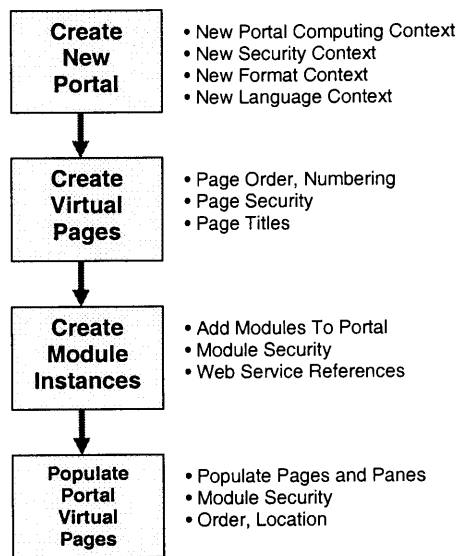


Figure 10-15 Portal Automation Cycle

Hence, a portal type definition involves the web service references, module registration, page registration, module registration, and security setup. The diagram below emphasizes the distributed nature and loose coupling of the portal definition. Using XML messaging obfuscates the operating system hosting the web service. The web service abstraction directs the designer to compose applications thinking at the service level and focusing on message paths as the definition of the application. Much like in the portal definitions, changing the chaining and services produces a different portal path.

The architecture is even more flexible than a portal factory. The derived information products need not be a portal. The application can be anything the user desires based on the services pool. For example, the application can be a front end for high performance computing, a television programming aggregation site, or a real time link to laboratory equipment. The focus on collaborative applications has been chosen because it is at the heart of the grid problem definition.

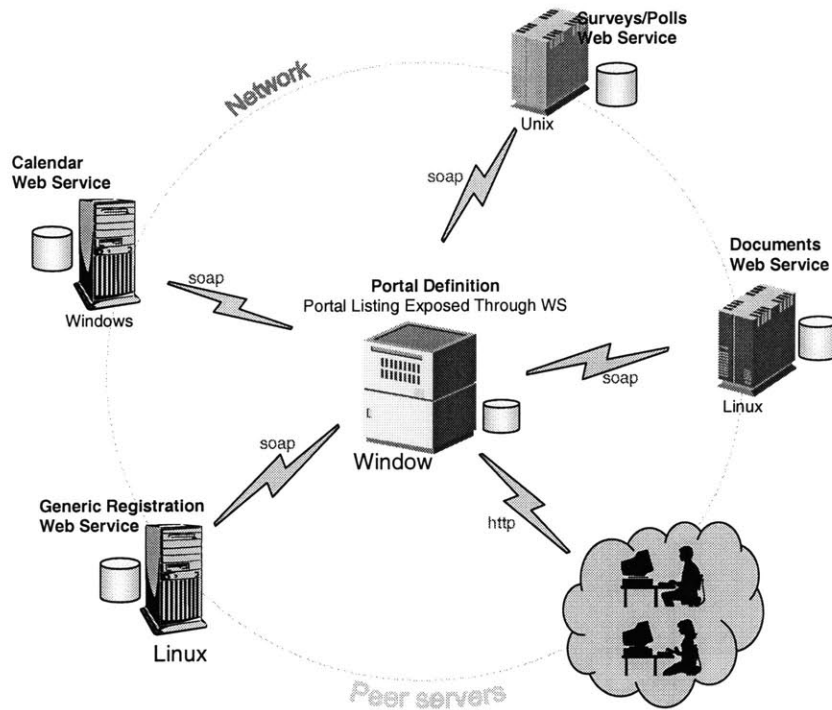


Figure 10-16 Peer Servers in a Services Grid

10.11 Portal Managers

As the number of portals increase at a rapid pace soon there is a management problem. Locating a portal, inter-portal navigation, sharing relationships between portals, and logical grouping and categorization are issues that need to be addressed. The portal managers are defined by portal type.

The sharing relationships between portals can be very complex. The use cases and workflow for sharing resources between portals can be rich and varied. For example, the access to portal resources is too broad and too permissive. The permissions need to be more granular to share portal components/modules. Each module needs an approval cycle for each member/portal request. The member/portal receiving the permission also needs another approval cycle for the delegation of permissions with in its own portal.

Portal managers lie at the root of the portal type hierarchy. The manager is a portal itself and portal creation, portal deletion, and portal management is performed through the root portal. The portal manager also holds modules for portal type statistics.

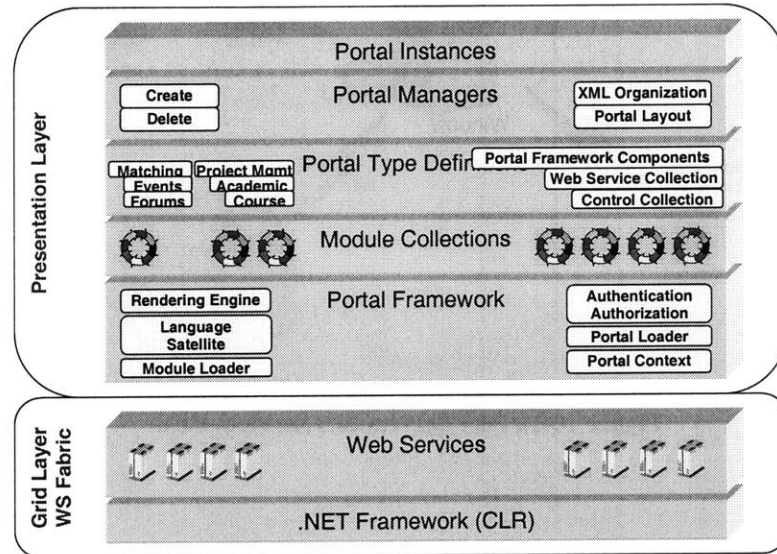


Figure 10-17 Portal Managers

10.12 Portal Factory

The architecture as introduced in this section is a platform to create platforms. Each new portal type being a new application platform that can be extended, derived, and edited. For example, the “events” portal type creates events types which in turns are instantiated for each event instance. A seminar and a conference would be event types. Each new conference would be a new portal and a new instance of the conference event type. At the time of writing there were eleven portal types, listed below.

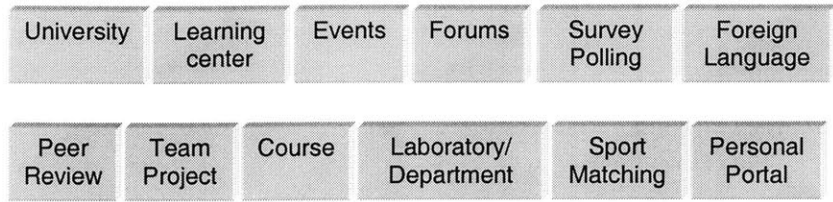


Figure 10-18 Portal Types

Sample of portal instances

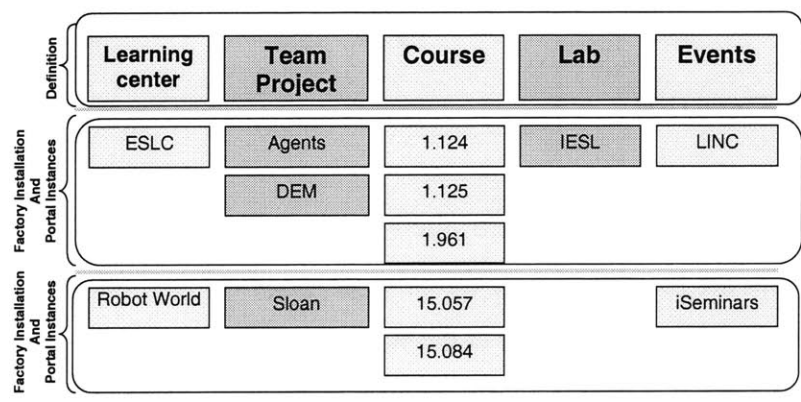


Figure 10-19 Portal Instances

The portal architecture can be extended from one installation to a grid of multi-portal installations. The messaging infrastructure is loosely coupled, distributed, and based on XML. Spanning multiple installations is a well supported by the portal infrastructure.

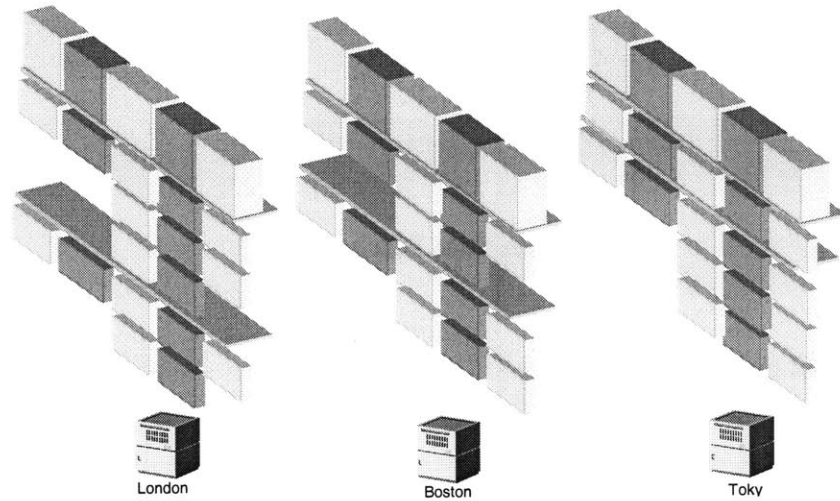


Figure 10-20 Multi-Portal Grid

The diagram above illustrates a scenario where three multi-portal installation systems and supporting services are located in three different geographic locations. The three independent installations can integrate at the service level.

The portal is built on a service oriented architecture (SOA) and all portal components are services. Any system can remotely bind to remote services like registration, documents, and web logs. Portal implementations can converge on one registration, authentication, and authorization service or they can delegate credentials. Likewise, several implementations can agree to share documents and use the same service. On the flip side one installation can decide to leverage services from through out the world, see below.

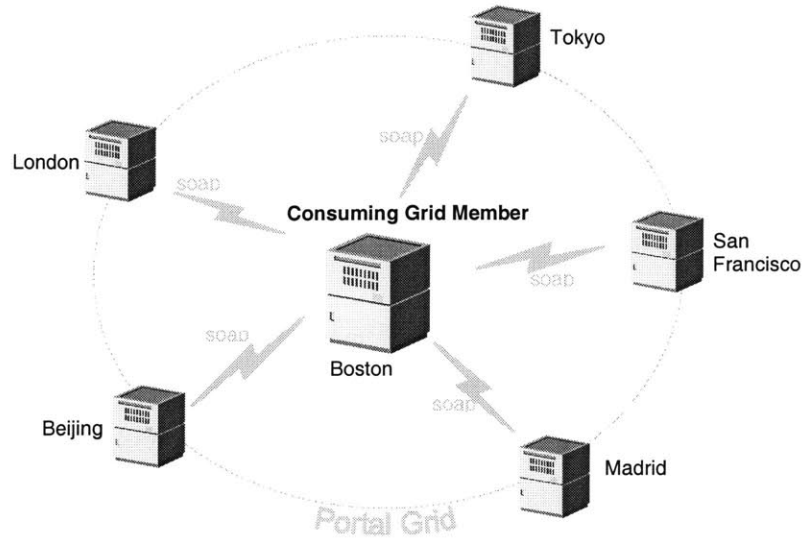


Figure 10-21 Referencing the Global Grid

At rendering time the modules would go out and bind the remote web service. The web service response is computed and targeted to a portal page pane.

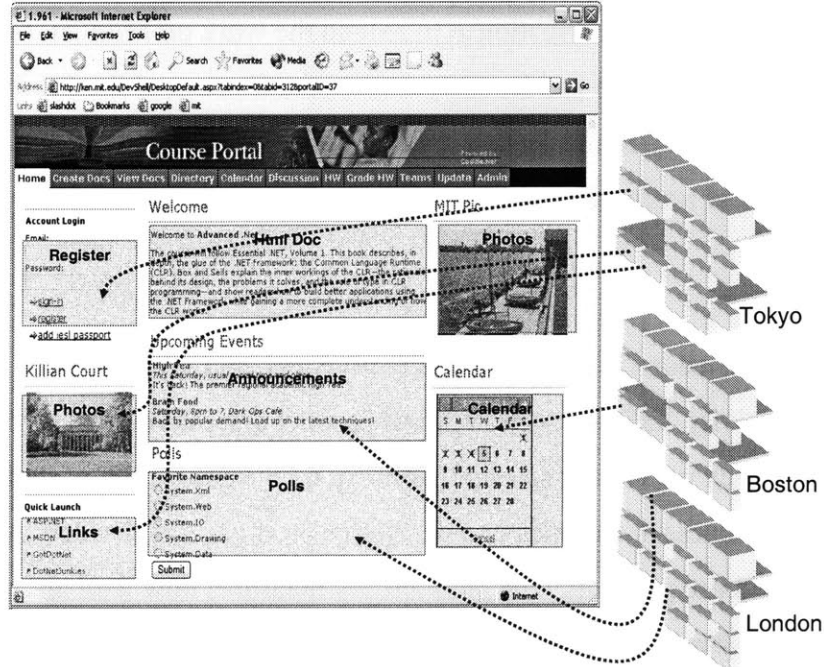


Figure 10-22 Sourcing the Global Grid

Chapter 11 Metrics

This section proposes metrics to assess the research and development of information product platforms.

Product platform literature, has studied how product series can be efficiently constructed on successive generations of underlying product architectures commonly referred to as product platforms. (Sanderson and Uzumeri, 1991; Wheelwright and Clark, 1992; Meyer and Utterback, 1993). Measurements of an organizations ability to efficiently create new products of a product platform are important in validating the existence of platforms.

Numerous metrics approaches have been proposed to measure research and development in the literature. However, a survey by 248 R&D executives by the Industrial Research Institute found that measuring and improving R&D productivity and effectiveness remained the biggest problem [Industrial Research Institute, 1993] [Rockwell & Particelli, 1982]. The focus of this section is to extend the work on Mark H. Meyer's, "Metrics for Managing Research and Development in the Context of the Product Family" [Meyer & Tertzakian, 1997], to the academic grid platform. The aim is to better understand the technical and economic effectiveness of research and development based on product platforms.

The measurement dimensions are efficiency, effectiveness, and research and development productivity. Efficiency is a ratio of product derivatives cost to that of the platform. The efficiency puts a monetary relationship on the generation of derivative products. The effectiveness is the sales of a derivative products line divided by the total engineering costs of these products. The productivity measures how well an organization translates technological advantage into financial gain through products. Overarching all measures are the resource inputs to research and development: the engineering budget and the project time.

The implementations and measurements were done in an academic setting. In this setting, product adoptions were taken as sales since the considered products are public domain. In this case, the use of an information product by an individual(s) constitutes a sale.

Products sales are a better measure of commercial outcomes than profits. The way in which profits are calculated varies significantly for from product to product and from firm to firm. [Patterson, 1983]

The application analysis sought to identify [Meyer & Tertzakian, 1997]:

- The initial new platform development efforts
- Extensions to the existing platforms
- The creation of the whole new platforms or product architectures to replace those in existence
- The specific products associated with each platform generation

The application analysis sought to measure [Meyer & Tertzakian, 1997]:

- Individual products within the platform version of a product family
- An aggregated level for the product family as a whole for successive platform versions
- Comparatively across different platform versions for the different product families

The raison d'être for measurement is to help quantify platform development; to help understand effective/efficient products across product lines. To determine, in a field of competing platforms, the ones that can continue to regenerate and derive products flexibly and at a low cost.

The parameters are sales(S), costs(C), and an adjustment factor(D) for inflation in very long product cycles. There are also a number of indices:

NP = Number of portal types in a product family *excluding* the base portal type version or architecture.

NV = Number of portal type extensions, or new types created off the base type architecture, *excluding* the base portal type.

NF = Number of follow-on portal types in a platform, or type extensions, excluding the base portal type.

NTD = The total number of follow-on portal types in a family excluding the base portal type.

TD = The last time period number prior to entering the academic term.

TC = The last time period for which implementations were recorded in the academic term.

p = portal type index; p = base, 1, 2, 3, ..., NP

v = portal type version index; v = base, 1, 2, 3, ..., NV

f = follow-on product index; f = base, 1, 2, 3, ..., NF

t = time period index; t = 1, 2, 3, ..., TD or TC

The formulas below represent the aggregated product implementation (S) and cost (C) for individual follow-on products within the specific platform version within the same product family. The quantities aggregate the implementations and costs of a product over time.

$$S_{p,v,f} = \sum_{t=1}^{T_c} (S_{p,v,f})_t$$

$$C_{p,v,f} = \sum_{t=1}^{T_D} (C_{p,v,f})_t$$

Implementations and costs are recorded at regular time intervals, t , through the academic term. The academic term can be exchanged for any product cycle. Implementations are added from inception until the product is discontinued. Research costs are added from inception.

“To accumulate costs at the platform version level, the summations are extended. For example, the accumulated inflation adjusted sales for an entire platform, p , including all its platform extensions, would be given by” [Meyer & Tertzakian, 1997]

$$S'_p = \sum_{v=base}^{N_v} \sum_{f=base}^{N_f} S'_{p,v,f}$$

Where S' is

$$S'_{p,v,f} = \sum_{t=1}^{T_c} (S_{p,v,f})_t D_t$$

The corrected expenses for the platform are calculated using the implementation variable, C.

11.1 Efficiency

The efficiency of developing an individual follow-on product relative to its base platform is [Meyer & Tertzakian, 1997]

$$\text{Platform Efficiency} = \frac{R \& D \text{ Costs for Derivative product}}{R \& D \text{ Costs for Platform Version}}$$

At the individual follow-on product level, the mathematical formulation for E , the efficiency of developing an individual follow-on product relative to its base platform, is given by [Meyer & Tertzakian, 1997]

$$E_{p,v,f} = \frac{C'_{p,v,f}}{C'_{p,v,base}}$$

Determines the cost of the product as fraction of the amount spent on the base platform architecture [Meyer & Tertzakian, 1997]

$$\bar{E}_{p,v} = \frac{\frac{1}{N_f} \sum_{f=1}^{N_f} C'_{p,v,f}}{C'_{p,v,base}}$$

Is the average cost.

Differential comparisons across platforms are important to gage efficiency between platforms built by different groups.

A cumulative efficiency for a product family is given by:

$$\bar{E} = \frac{\frac{1}{N_{TF}} \sum_{f=base}^{N_f} \sum_{v=base}^{N_v} \sum_{f=1}^{N_f} C'_{p,v,f}}{\sum_{p=base}^{N_p} \sum_{v=base}^{N_v} C'_{p,v,f}}$$

Platform efficiency should improve with each revision and generation. However, there must be a continual investment in the platform to incorporate the latest technologies and processes. Absence of investment in the platform leads to platform death [Foster, 1985].

Platform efficiency can be used as an indicator of platform demise. An increase in E over successive follow-on products may indicate weakness in the underlying product architecture. The S-curve phenomenon where, after a certain point, little additional product performance is achieved from incremental investment. [Foster, 1986]

Cycle time is another measure of efficiency. Architects must strive for platforms that not only are efficient but also have short cycle times. Cycle time can be measured as:

$$E_{p,v,base}^{CycleTime} = \frac{T_{p,v,f}}{T_{p,v,base}}$$

11.2 Effectiveness

Efficiency answers the questions, “can the product be made in a cost effective manner?”
And, “can the product be made in a timely manner?”

Effectiveness measures whether the market is interested in the product. The return provided on the product development investment. That is, is the product being used, implemented, or sold.

Platform effectiveness questions [Meyer & Tertzakian, 1997]:

- What have been the returns realized on the firm's R&D investments as seen in comparing sales from products to the costs of developing them?
- How has a particular product family's sales to development costs effectiveness trended over time through successive platform versions? Have these returns improved or declined?
- Have certain development groups proven to be more effective in creating leverage from their respective platform development efforts than other groups?

Platform effectiveness:

$$L_{p,v,f} = \frac{S'_{p,v,f}}{C'_{p,v,f}}$$

Aggregated platform effectiveness:

$$L_{p,v} = \frac{\sum_{f=base}^{N_f} S'_{p,v,f}}{\sum_{f=base}^{N_f} C'_{p,v,f}}$$

The effectiveness is the accrued sales for a set of derivative products divided by the total engineering costs of those products.

Across all platforms with in a product family:

$$L_p = \frac{\sum_{v=base}^{N_v} \sum_{f=base}^{N_f} S'_{p,v,f}}{\sum_{v=base}^{N_v} \sum_{f=base}^{N_f} C'_{p,v,f}}$$

11.3 Declining platform effectiveness.

As platforms cease to yield low cost products, leveraging the platform subsystems, the life of the platform can be questioned. One of the first flags is declining adoptions. On the flip side platform investment may increase as the platform becomes less efficient. If this is the case, the efficiency and effectiveness become inversely related. Possible reasons are organizational process or the emergence of new technologies [Utterback, 1993].

Unless a continuous investment is made platforms become outdated and obsolete. However, those are not the only reasons for platform demise. New technologies that are orthogonal to the platform architecture may make a redesign imperative. Changes in the organization, such as mergers, may disrupt the smooth flow of a platform. There are more that can be found in the literature [Meyer & Tertzakian, 1997].

11.4 Observations

More work is needed to incorporate better measures in the following areas:

- The technology buzz that accompanies some technologies brings forth a large number of adopters. Less attractive technologies that are better designed and are more efficient experience slow growth. Over many years

the differences are reflected in the metrics. However, platforms do not always survive that long. Over the short cycles of software developments they are hard to detect.

- In academe product timing is very important. Introducing a product at the end of an academic term is doom. Time pressures and end of term commitments absorb the academe audience.
- A strong market brings more research funds to academe. With more research funds come more resources and more opportunities to experiment. Platforms introducing products into a strong market can more easily achieve high efficiencies and effectiveness. However, in a weak period a more efficient/effective platform maybe introduced with poorer results.

A Final point on platform investment, “The failure to develop new platform architectures on a continuous basis subjects the firm to substantial market risk. Research has found that engineering managers generally understand this. Business managers often do not, leading to inappropriate expectations regarding project time and cost (ie. new platforms are different than platform enhancements, and both are different than specific product developments)” [Meyer & Tertzakian, 1997].

Chapter 12 Future Work and Contributions

12.1 Future Work

The most direct paths of future work are transaction support layers for the XML architecture and workflow management systems. Other venues are mining the vast collected data of the command system.

12.1.1 XML Transactions

Since work began on the portal grid, software architects have been hard at work writing new protocols for the web services federation. There are proposals for:

Web Service Specifications	
Routing	Secure Conversation
Referral	Security Policy
Inspection	Policy
Security	Policy Attachment
Attachments	Policy Assertions
Coordination	Addressing
Transaction	Reliable Messaging
Trust	

Figure 12-1 Web Service Specifications [IBM-WS, 2003]

Many architects have worked on the web service specification group. As the heterogeneity of grid applications spreads and the complexity of distributed systems increases many of the specification's insights will be valuable.

The current work being done in federated architecture specifications will likely be the most relevant to the portal grid work in the short run. In the long run, transactions coupled with workflow will have a big impact in the flexibility and scale of the implementations.

12.1.2 Workflow

Today's information systems need to manage the flow of work through the organization. Complex business processes highlight the need for concepts, techniques, and tools to support the management of workflow. The problem area is called workflow management; applications written to support the definition, execution, registration and control of *processes*. More formally, the Workflow Management Coalition [WfMC, 2003] defines workflow management systems as a system that completely defines, manages, and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic.

Once workflow is built into an application it is expensive to generalize and leverage in other applications. The dependencies of the application environment force developers to create custom solutions for each application. The commonality of the problem is lost and efficiency is lost. The generic functionality of workflow should be pushed out of the applications into a framework much like data was pushed out of applications into relational database management systems (RDBMS).

The portal grid goes a long way to isolate services and make it easy to build new applications based on services building blocks. However, more needs to be done to approach the goal of efficiently, practically, and affordably building workflows.

Several models, like the finite state machine, have been proposed to model and analyze workflow processes. A suggested approach is Petri nets. Petri nets are a well known process modeling technique with a firm mathematical foundation. Petri nets have been used to model processes in protocols, hardware, and flexible manufacturing systems.

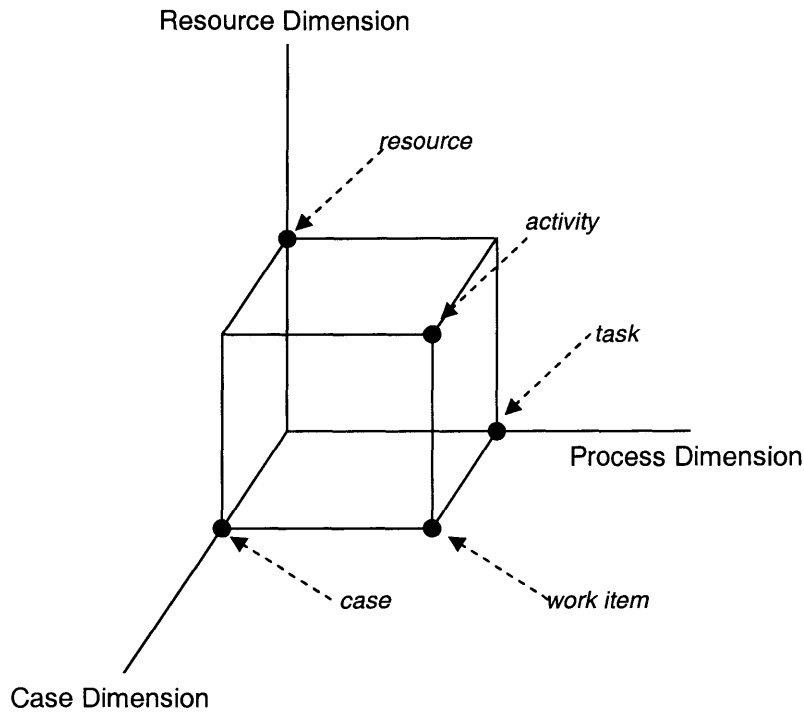
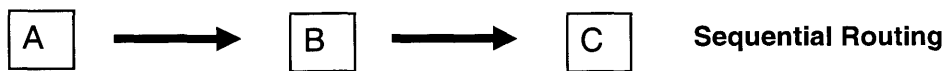


Figure 12-2 Workflow Dimensions

In the diagram above each dot represents either a work item (case + task) or an activity (case + task + resource). The figure shows that workflow management is the glue between the cases, the tasks, and the organization.

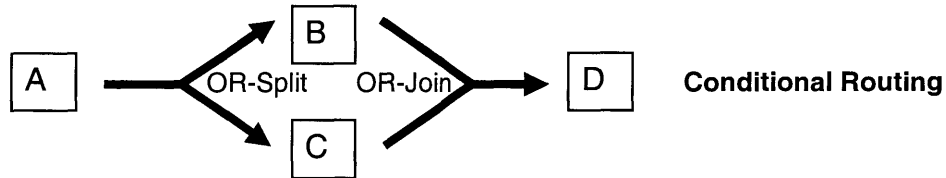
The Petri nets solution focuses on the case and process dimension. For these dimensions, workflow to support cases, routing is the core issue. A workflow process definition specifies how cases are routed along the tasks execution order. The WfMC has identified four types of routing constructs:



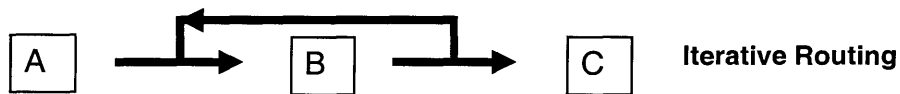
Sequential routing, tasks are executed sequentially if the execution of one task is followed by the next.



In parallel routing tasks are executed at the same time.



In conditional routing a choice is made between tasks and only one is executed.



A tasks that is repeated a number of times is an example of iterative routing.

To model workflow using Petri nets tasks are modeled as transitions, conditions as places, and cases as tokens.

To reduce complexity it is advantageous to separate the workflow process model from the resources model. Not only is the complexity reduced but the separation facilitates modularity, code reuse, and workflow process updates with out touching the organizational model.

Workflow parameters like the address in a registration form are supported through the high-level Petri with color extension. The variables are not part of the workflow process definition but are associated with the task and used for execution.

The discussion on work is meant as a brief introduction to the future work on the portal grid and workflow management systems.

12.1.3 Data Mining

A tremendous amount of data exists for the Course Management and Delivery (COMMAND) System. Data logs and operations have been recorded for the five years the system has been in operation. The course databases have been archived at the end of each academic term for future analysis.

The data captured includes HTTP logs, course materials, and discussions forums. HTTP logs include operating systems, browser, IP, and the myriad of CGI variables that accompany each HTTP request. Course materials are lecture notes, syllabi, readings, assignments, and videos. The forums include discussion threads and every entry made by students and faculty.

All operations were recorded in a data store. Every create action, read action, update action, and delete action was recorded individually in each course database. Every operation has a date and time stamp. The time and data patterns are thus preserved for future analysis. The data stores will be available for analysis as part of this work. The data will be in a masked form to protect student and faculty privacy.

After 5 years of operation the system has in store 476 course instances. There are over 200,000 documents, 16,199 users, and 61,627 submitted files. There are also 84 team spaces for which data has been captured in an similar fashion.

12.2 Contributions

12.2.1 Grid Extension and Implementation

This work has introduces a presentation layer to a stateless grid implementation. The presentation layer acts as a run time portal definition, dynamically binding services. The late binding permits on the fly manipulation of the portal components and just in time

portal creation. The presentation layer draws on the pool of computational resources stored in UDDI.

12.2.2 Information Product Platform

Leveraging lessons learned from product platforms, an information product platform was architected. The realized goals were:

- Lower cost for information products
- A modular platform for collaborative work
- A framework to analyze information product platforms
- Measures to gage the efficiency and effectiveness of the platform.

12.2.3 Service Oriented Architecture

This work presents an implementation of the proposed service oriented portal grid architecture. The portal makes the case that it is much easier to manage change in the composition of many small web services who share few well known abstractions than it is to manage change in varying abstractions exposed by the same set of services. In the former the difficulty lies in composing messaging routes. In the latter the barrier lies in propagating the knowledge to consume each abstraction.

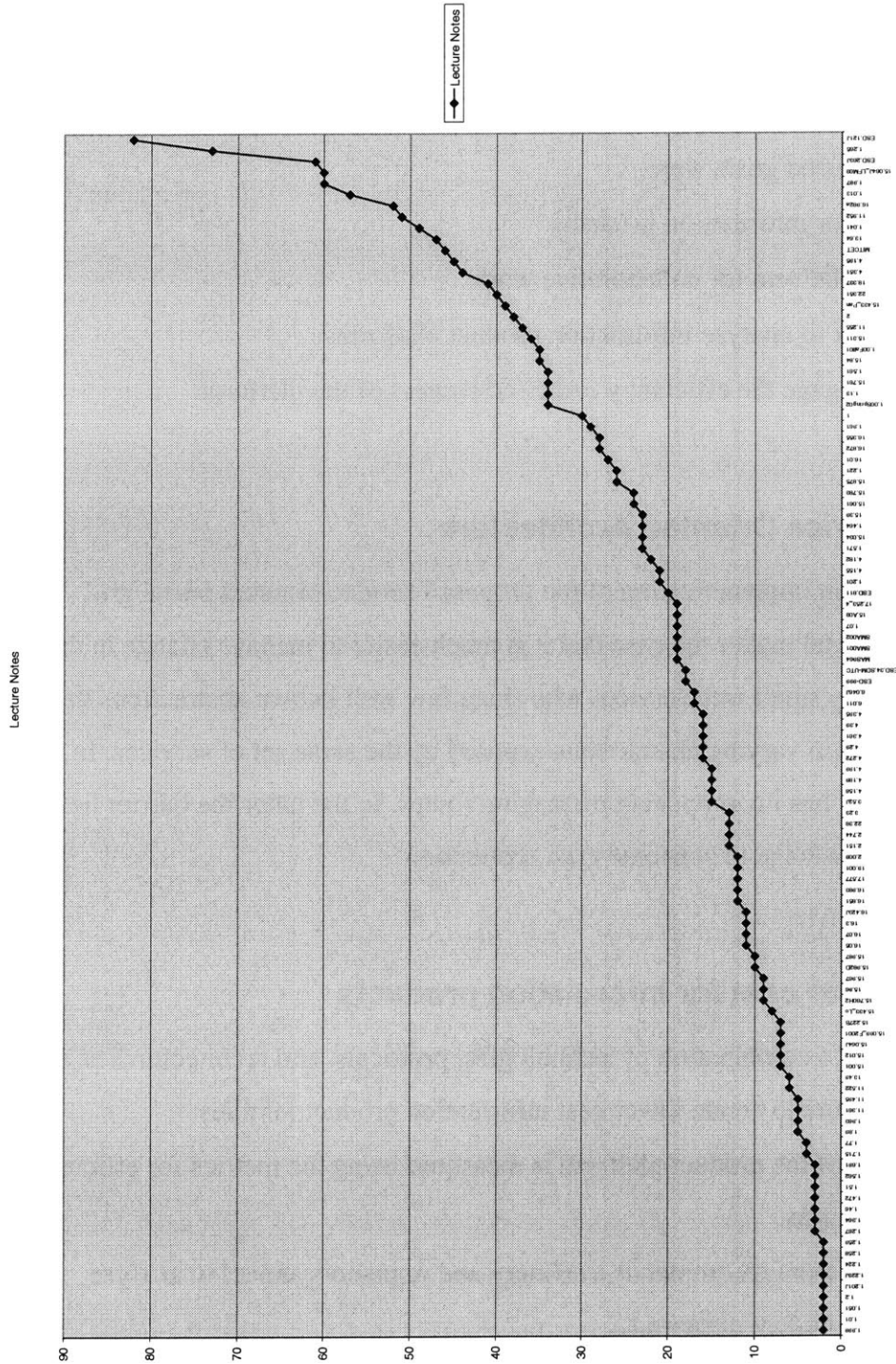
12.2.4 Lower cost for information products

This work presented a combination of technologies, protocols, and architectures as a reference infrastructure to create lower cost information product families.

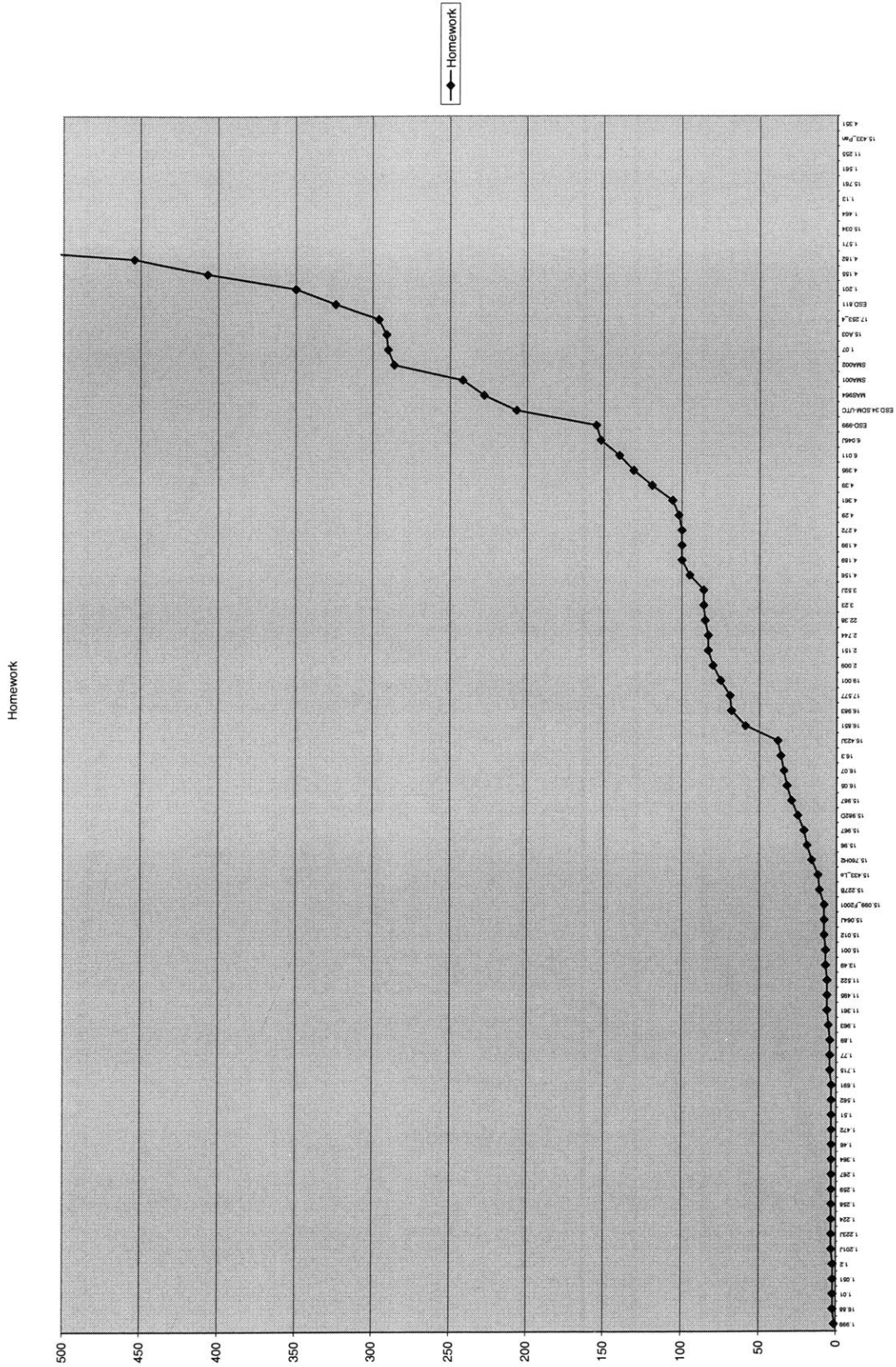
- The success of the product platform is measured using the metrics for efficiency and effectiveness.
- The infrastructure recommends a refinery and repository model to analyze information product platforms.

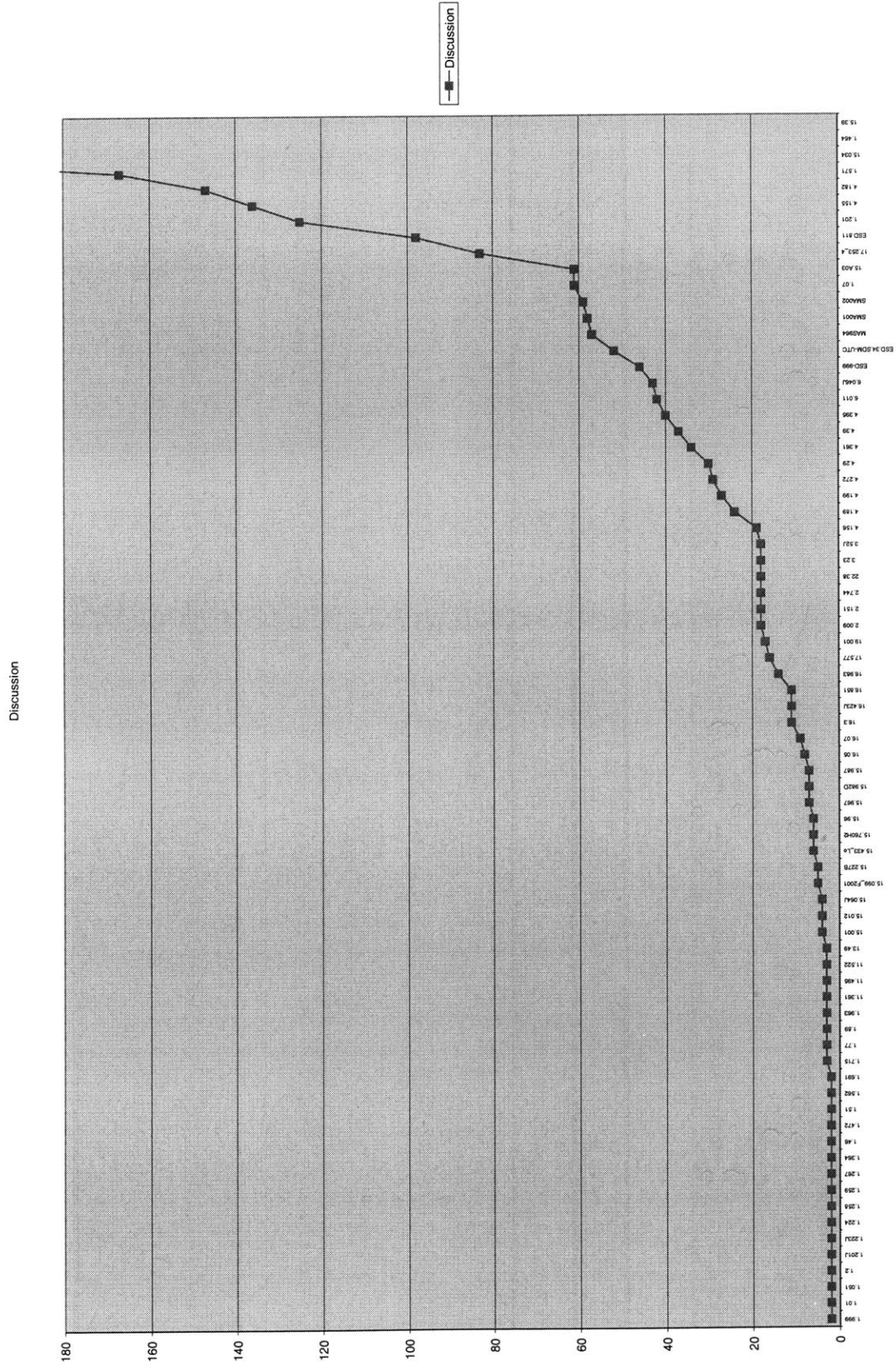
Appendix 1 Command Data

A.1.1 Document Counts per Category – System Wide

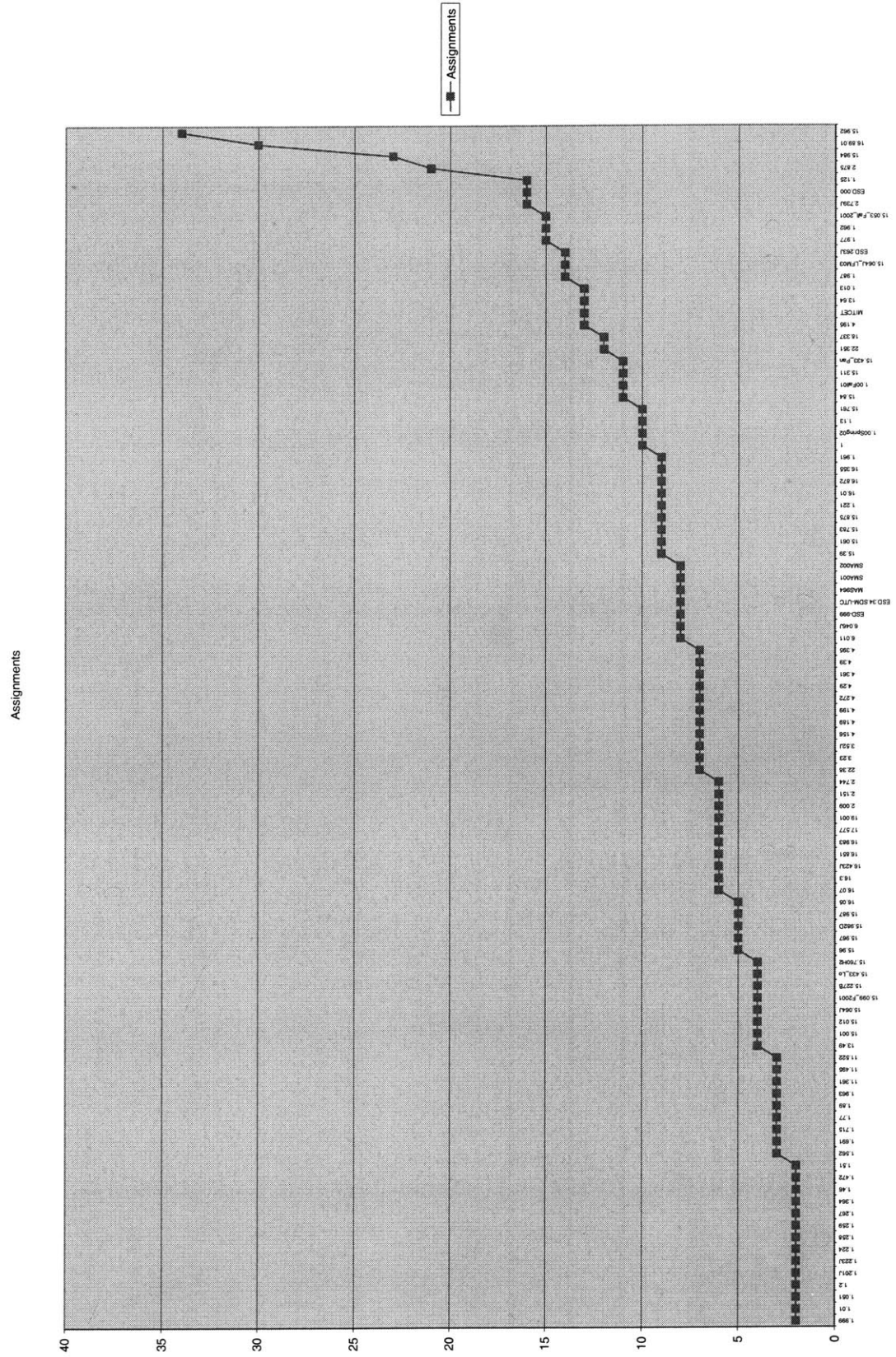


Appendix Figure 1 Lecture Notes

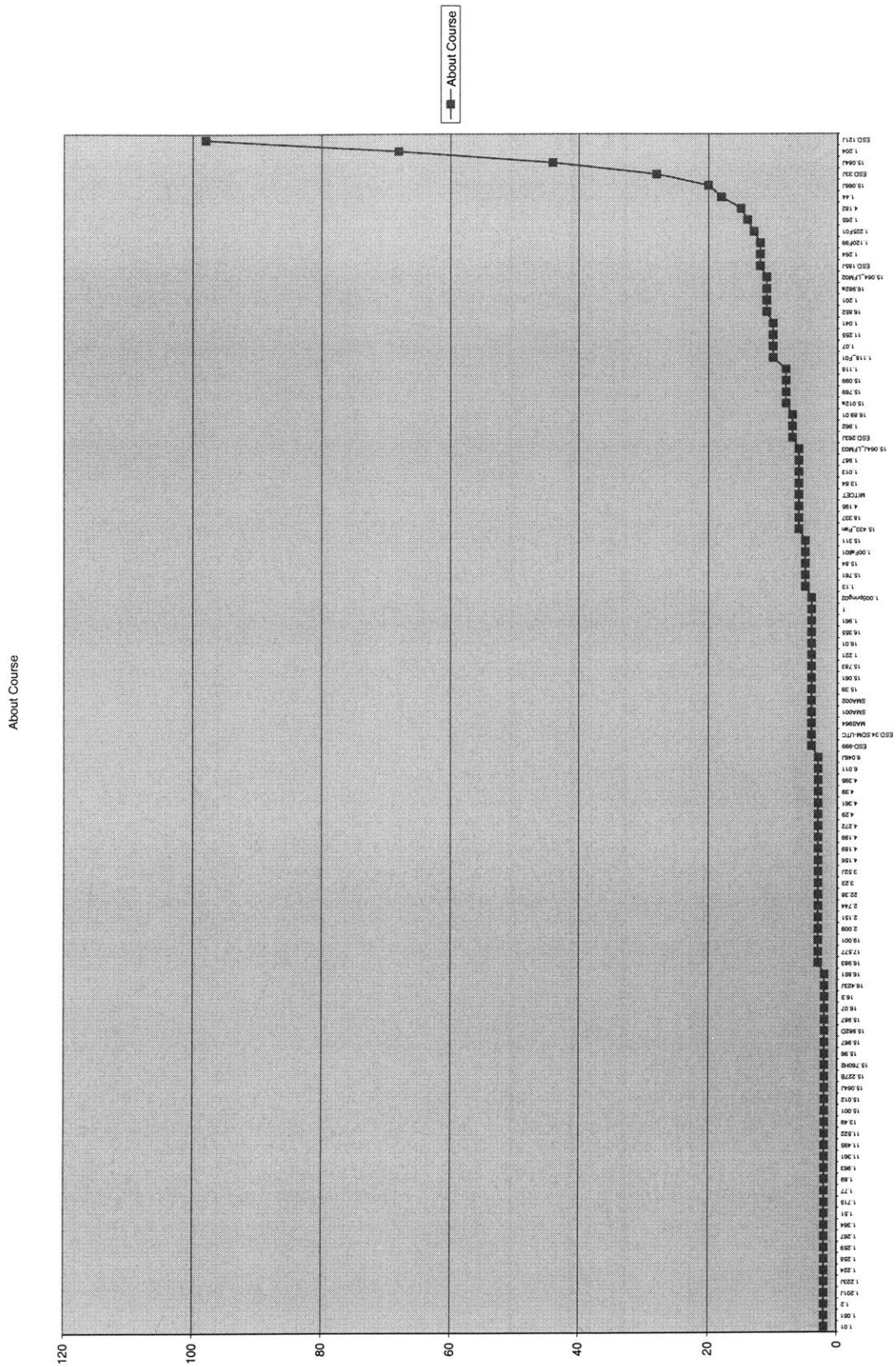




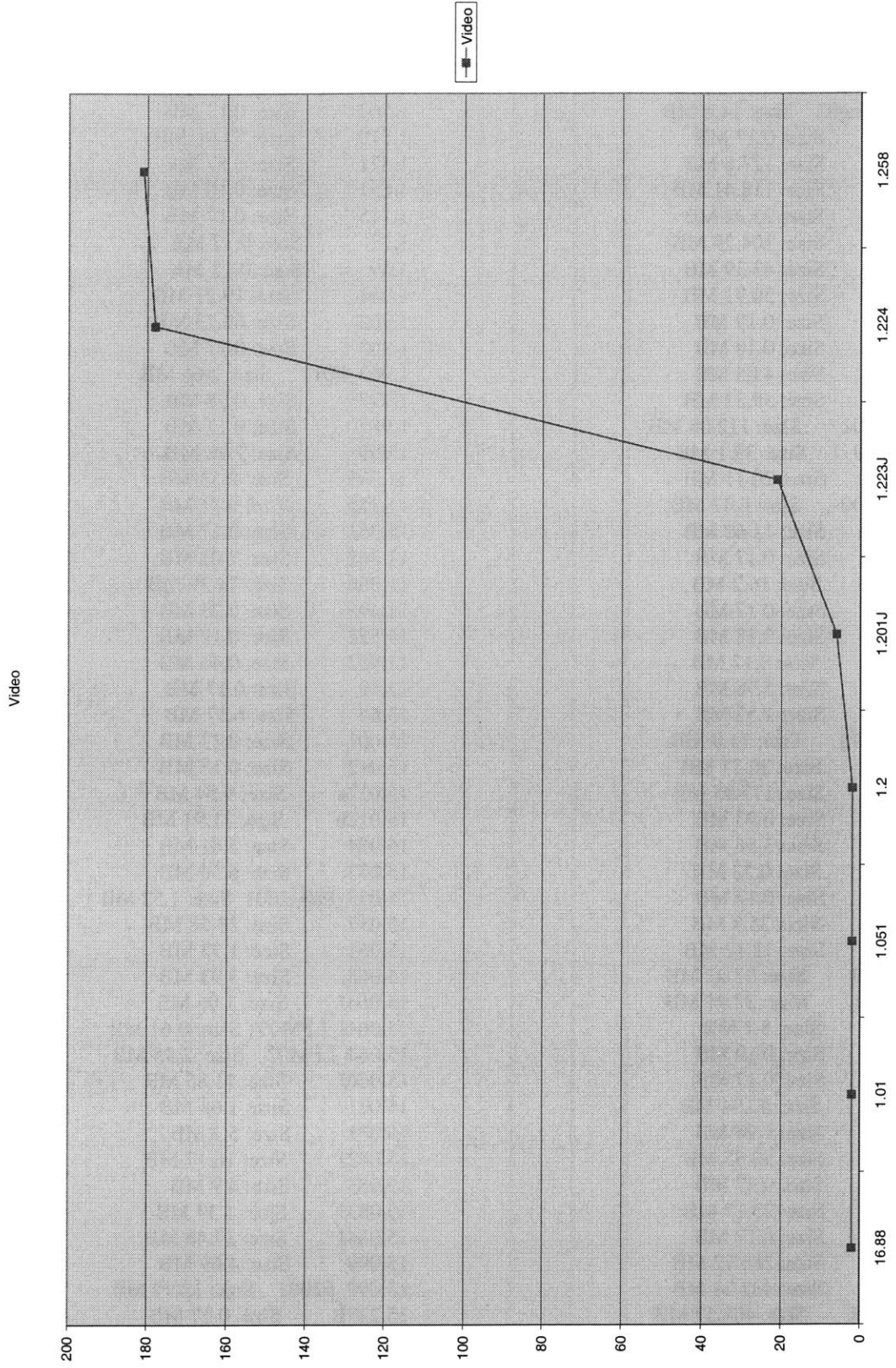
Appendix Figure 3 Discussions



Appendix Figure 5 Assignments



Appendix Figure 6 About Course

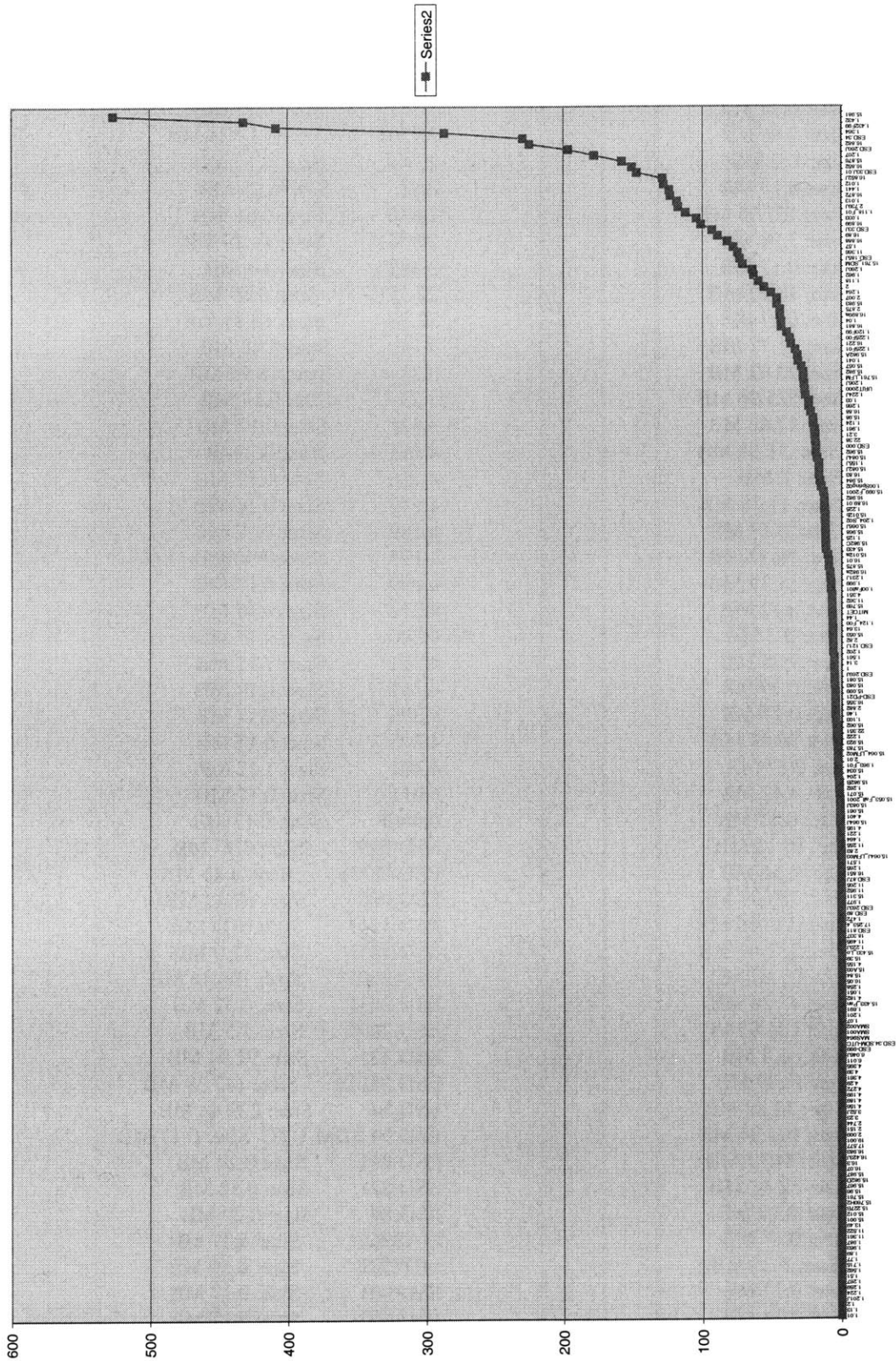


Appendix Figure 8 Video

A.1.2 Directory Sizes

1.00	Size: 5.44 MB	1.51	Size: 0.17 MB
1.00Fall01	Size: 7.43 MB	1.561	Size: 5.51 MB
1.00Spring02	Size: 14.6 MB	1.562	Size: 0.17 MB
1.010	Size: 0.17 MB	1.570	Size: 77.68 MB
1.012	Size: 127.8 MB	1.571	Size: 0.57 MB
1.013	Size: 118.41 MB	1.691	Size: 0.18 MB
1.030	Size: 23.32 MB	1.715	Size: 0.17 MB
1.033	Size: 104.29 MB	1.77	Size: 0.17 MB
1.040	Size: 43.39 MB	1.89	Size: 0.17 MB
1.041	Size: 30.91 MB	1.961	Size: 19.21 MB
1.051	Size: 0.19 MB	1.962	Size: 62.73 MB
1.070	Size: 0.18 MB	1.963	Size: 0.17 MB
1.103	Size: 4.05 MB	1.963_F01	Size: 2.66 MB
1.118	Size: 59.33 MB	1.977	Size: 0.38 MB
1.118_F01	Size: 112.08 MB	1.987	Size: 0.17 MB
1.120F99	Size: 39.1 MB	1.999	Size: 7.48 MB
1.124	Size: 20.11 MB	11.205	Size: 0.52 MB
1.124_F00	Size: 6.47 MB	11.255	Size: 0.71 MB
1.125	Size: 11.62 MB	11.361	Size: 0.17 MB
1.130	Size: 0.17 MB	11.362	Size: 7.02 MB
1.155J	Size: 16.2 MB	11.366	Size: 74.39 MB
1.200	Size: 0.17 MB	11.495	Size: 0.23 MB
1.201	Size: 0.18 MB	11.522	Size: 0.17 MB
1.201J	Size: 0.17 MB	11.952	Size: 0.46 MB
1.202	Size: 5.98 MB	13.49	Size: 0.17 MB
1.204	Size: 2.52 MB	13.64	Size: 6.37 MB
1.204_S02	Size: 11.9 MB	15.001	Size: 0.17 MB
1.206J	Size: 26.77 MB	15.012	Size: 0.17 MB
1.207	Size: 177.95 MB	15.012a	Size: 9.94 MB
1.221	Size: 0.93 MB	15.012b	Size: 11.91 MB
1.222	Size: 3.56 MB	15.034	Size: 2.61 MB
1.223J	Size: 0.22 MB	15.053	Size: 6.36 MB
1.224	Size: 0.17 MB	15.053_Fall_2001	Size: 1.52 MB
1.224J	Size: 25.8 MB	15.057	Size: 28.58 MB
1.225	Size: 12.12 MB	15.061	Size: 1.33 MB
1.225F00	Size: 37.02 MB	15.062	Size: 3.93 MB
1.225F01	Size: 32.91 MB	15.064J	Size: 1.06 MB
1.231J	Size: 8.1 MB	15.064J_LFM03	Size: 0.61 MB
1.258	Size: 0.19 MB	15.064_LFM02	Size: 2.95 MB
1.259	Size: 0.17 MB	15.066J	Size: 11.85 MB
1.260J	Size: 63.94 MB	15.071	Size: 1.68 MB
1.262	Size: 1.94 MB	15.081	Size: 5.2 MB
1.264	Size: 50.55 MB	15.082J	Size: 16.17 MB
1.265	Size: 0.57 MB	15.083	Size: 4.9 MB
1.266	Size: 23.17 MB	15.083J	Size: 1.39 MB
1.267	Size: 0.17 MB	15.084J	Size: 17.48 MB
1.364	Size: 286.72 MB	15.099	Size: 4.69 MB
1.432	Size: 432.34 MB	15.099_F2001	Size: 12.99 MB
1.432F99	Size: 408.52 MB	15.227B	Size: 0.17 MB
1.44	Size: 6.47 MB	15.311	Size: 0.45 MB
1.441	Size: 123.9 MB	15.390	Size: 0.21 MB
1.46	Size: 4.32 MB	15.433	Size: 11.25 MB
1.464	Size: 0.92 MB	15.433_Lo	Size: 0.21 MB
1.472	Size: 0.26 MB	15.433_Pan	Size: 0.18 MB

15.760H2	Size: 0.17 MB	2.000	Size: 55.54 MB
15.761	Size: 0.17 MB	2.007	Size: 46.4 MB
15.761_LFM	Size: 27.16 MB	2.009	Size: 0.17 MB
15.761_SDM	Size: 70.74 MB	2.010	Size: 2.73 MB
15.769	Size: 6.78 MB	2.151	Size: 0.17 MB
15.783	Size: 2.97 MB	2.739J	Size: 117.28 MB
15.840	Size: 0.2 MB	2.744	Size: 0.17 MB
15.875	Size: 9.15 MB	2.82	Size: 6.25 MB
15.876	Size: 157.95 MB	2.830	Size: 0.61 MB
15.923	Size: 3.39 MB	2.875	Size: 44.29 MB
15.960	Size: 0.17 MB	2.882	Size: 4.4 MB
15.962	Size: 18.27 MB	22.351	Size: 3.69 MB
15.967	Size: 0.17 MB	22.38	Size: 18.84 MB
15.968	Size: 11.77 MB	3.14	Size: 5.48 MB
15.980	Size: 20.42 MB	3.21	Size: 18.99 MB
15.981	Size: 525.96 MB	3.23	Size: 0.17 MB
15.982	Size: 27.42 MB	3.52J	Size: 0.17 MB
15.982A	Size: 31.53 MB	4.155	Size: 0.2 MB
15.982B	Size: 2 MB	4.156	Size: 0.17 MB
15.982C	Size: 11.38 MB	4.182	Size: 0.18 MB
15.982D	Size: 0.17 MB	4.189	Size: 0.17 MB
15.983	Size: 46.37 MB	4.195	Size: 0.94 MB
15.984	Size: 15.39 MB	4.199	Size: 0.17 MB
15.987	Size: 0.17 MB	4.272	Size: 0.17 MB
15.A03	Size: 0.2 MB	4.290	Size: 0.17 MB
16.010	Size: 9.25 MB	4.351	Size: 7.11 MB
16.050	Size: 0.19 MB	4.361	Size: 0.17 MB
16.070	Size: 0.17 MB	4.390	Size: 0.17 MB
16.221	Size: 36.07 MB	4.395	Size: 0.17 MB
16.30	Size: 0.17 MB	4.401	Size: 1.12 MB
16.355	Size: 4.42 MB	6.011	Size: 0.17 MB
16.423J	Size: 0.17 MB	6.046J	Size: 0.17 MB
16.83	Size: 16.12 MB	ESD-999	Size: 0.17 MB
16.851	Size: 0.56 MB	ESD-PD21	Size: 4.46 MB
16.852	Size: 150.94 MB	ESD.000	Size: 18.42 MB
16.852J	Size: 128.89 MB	ESD.121J	Size: 6.03 MB
16.872	Size: 122.99 MB	ESD.185J	Size: 72.9 MB
16.880	Size: 22.02 MB	ESD.260J	Size: 196.46 MB
16.881	Size: 43.29 MB	ESD.263J	Size: 0.32 MB
16.882	Size: 224.8 MB	ESD.269J	Size: 5.3 MB
16.888	Size: 81.8 MB	ESD.33J	Size: 92.91 MB
16.89	Size: 88.89 MB	ESD.33J.01	Size: 147.39 MB
16.89.01	Size: 12.29 MB	ESD.34	Size: 229.41 MB
16.899	Size: 101.04 MB	ESD.34.SDM-UTC	Size: 0.17 MB
16.899a	Size: 44.11 MB	ESD.811	Size: 0.24 MB
16.982	Size: 12.44 MB	ESD.87J	Size: 0.52 MB
16.982a	Size: 8.23 MB	ESD.89	Size: 0.29 MB
16.983	Size: 0.17 MB	MAS964	Size: 0.17 MB
17.253_4	Size: 0.25 MB	MITCET	Size: 6.59 MB
17.577	Size: 0.17 MB	SMA001	Size: 0.17 MB
18.337	Size: 0.24 MB	SMA002	Size: 0.17 MB
19.001	Size: 0.17 MB	URIT2000	Size: 26.18 MB



Appendix Figure 9 Directory Sizes

A.1.3 File Types

A.1.3.1 File Type Count per Course

----- COURSE 1.00 ----- txt, 161 zip, 306 html, 823 tar, 1 java, 928 ZIP, 1 jpg, 24 class, 2 fig, 33 java~, 1 TOTAL FILE NUMBER: 2280 ----- COURSE 1.00Fall01 ----- txt, 114 zip, 342 html, 695 java, 820 rtf, 4 ZIP, 29 jpg, 21 class, 13 java~, 10 TOTAL FILE NUMBER: 2048 ----- COURSE 1.00Spring02 ----- xls, 1 txt, 35 zip, 368 exe, 1 html, 1690 dat, 49 java, 3273 rtf, 3 ZIP, 25 jpg, 21 class, 41 java~, 29 rar, 1 TOTAL FILE NUMBER: 5537 ----- COURSE 1.010 ----- html, 11	jpg, 21 TOTAL FILE NUMBER: 32 ----- COURSE 1.012 ----- zip, 4 html, 21 htm, 19 dwg, 10 jpg, 74 JPG, 77 TOTAL FILE NUMBER: 205 ----- COURSE 1.013 ----- ppt, 6 txt, 4 html, 71 jpg, 66 JPG, 12 tif, 8 TOTAL FILE NUMBER: 167 ----- COURSE 1.030 ----- ppt, 3 pdf, 44 html, 12 htm, 1 jpg, 21 TOTAL FILE NUMBER: 81 ----- COURSE 1.033 ----- ppt, 2 pdf, 27 ps, 25 html, 12 htm, 2 jpg, 21 TOTAL FILE NUMBER: 89 ----- COURSE 1.040 ----- ppt, 16 xls, 15
--	--

zip, 10
html, 90
rtf, 2
jpg, 23

TOTAL FILE NUMBER: 156

COURSE 1.041

ppt, 7
xls, 3
html, 33
htm, 1
PPT, 2
jpg, 21
DOC, 1

TOTAL FILE NUMBER: 68

COURSE 1.051

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.070

html, 11
dat, 1
jpg, 21

TOTAL FILE NUMBER: 33

COURSE 1.103

xls, 14
pdf, 9
zip, 16
html, 12
dat, 1
jpg, 21
DOC, 5

TOTAL FILE NUMBER: 78

COURSE 1.118

ppt, 17
xls, 3
txt, 1
pdf, 3
ps, 2
zip, 10
exe, 2
html, 86

htm, 18
js, 1
ZIP, 1
jpg, 24
DOC, 2
class, 2

TOTAL FILE NUMBER: 172

COURSE 1.118_F01

ppt, 24
xls, 5
txt, 7
pdf, 18
ps, 2
zip, 22
exe, 2
html, 62
htm, 16
jpg, 21
gz, 2
vsd, 3

TOTAL FILE NUMBER: 184

COURSE 1.120F99

ppt, 14
xls, 1
pdf, 2
ps, 2
zip, 8
html, 108
htm, 1
tar, 1
dwg, 1
ZIP, 2
jpg, 26

TOTAL FILE NUMBER: 166

COURSE 1.124

ppt, 4
pdf, 4
ps, 2
exe, 1
html, 69
htm, 34
jpg, 22
C, 1

TOTAL FILE NUMBER: 137

COURSE 1.124_F00

ppt, 3
pdf, 7
ps, 1
html, 79
htm, 5
jpg, 21

TOTAL FILE NUMBER: 116

COURSE 1.125

ppt, 6
pdf, 8
zip, 11
html, 65
htm, 4
java, 6
jpg, 21

TOTAL FILE NUMBER: 121

COURSE 1.130

html, 12
jpg, 21

TOTAL FILE NUMBER: 33

COURSE 1.155J

xls, 9
pdf, 65
zip, 1
html, 105
htm, 1
PDF, 5
jpg, 21

TOTAL FILE NUMBER: 207

COURSE 1.200

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.201

html, 11
htm, 2
jpg, 21

TOTAL FILE NUMBER: 34

COURSE 1.201J

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.202

ppt, 1
pdf, 3
html, 17
dat, 2
jpg, 21
asc, 5
sav, 5
mcd, 1

TOTAL FILE NUMBER: 55

COURSE 1.204

txt, 7
pdf, 37
zip, 2
html, 45
dat, 2
PDF, 22
jpg, 21
c, 20
C, 27
h, 4
tif, 5

TOTAL FILE NUMBER: 192

COURSE 1.204_S02

ppt, 15
xls, 11
txt, 33
pdf, 27
ps, 1
zip, 24
html, 95
tar, 4
java, 19
rtf, 1
PPT, 1
jpg, 22
c, 12
gz, 2
cc, 11
cpp, 19
C, 15
h, 1

rar, 3

TOTAL FILE NUMBER: 316

COURSE 1.206J

xls, 99
pdf, 26
html, 66
PDF, 5
jpg, 21
XLS, 1

TOTAL FILE NUMBER: 218

COURSE 1.207

ppt, 16
pdf, 29
ps, 14
html, 39
tar, 25
jpg, 22
tex, 15
dvi, 2
prn, 3
wpd, 1
cgi, 3

TOTAL FILE NUMBER: 169

COURSE 1.221

html, 16
jpg, 21

TOTAL FILE NUMBER: 37

COURSE 1.222

ppt, 3
xls, 1
txt, 1
pdf, 7
zip, 1
html, 11
jpg, 21

TOTAL FILE NUMBER: 45

COURSE 1.223J

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.224

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.224J

ppt, 9
xls, 99
txt, 1
pdf, 20
zip, 17
html, 110
mod, 70
PPT, 2
ZIP, 2
jpg, 21
XLS, 5
dvi, 4
MOD, 3
csv, 1

TOTAL FILE NUMBER: 364

COURSE 1.225

ppt, 2
xls, 16
pdf, 32
ps, 1
html, 26
jpg, 21
XLS, 2
prn, 1
cgi, 1

TOTAL FILE NUMBER: 102

COURSE 1.225F00

xls, 47
pdf, 34
html, 46
PDF, 3
jpg, 21
m, 2

TOTAL FILE NUMBER: 153

COURSE 1.225F01

ppt, 19
xls, 45

pdf, 9
html, 38
jpg, 21
XLS, 1

TOTAL FILE NUMBER: 133

COURSE 1.231J

xls, 3
pdf, 91
html, 13
PDF, 6
jpg, 21

TOTAL FILE NUMBER: 134

COURSE 1.258

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.259

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.260J

ppt, 13
xls, 390
html, 366
htm, 3
jpg, 21
XLS, 12

TOTAL FILE NUMBER: 805

COURSE 1.262

ppt, 20
xls, 7
html, 29
jpg, 21

TOTAL FILE NUMBER: 77

COURSE 1.264

ppt, 28
mdb, 2
pdf, 2

zip, 2
html, 19
htm, 5
js, 1
PDF, 2
jpg, 24
MDB, 1
vsd, 1
xml, 6
css, 2

TOTAL FILE NUMBER: 95

COURSE 1.265

ppt, 1
html, 11
htm, 4
jpg, 21

TOTAL FILE NUMBER: 37

COURSE 1.266

ppt, 2
pdf, 32
html, 11
htm, 8
PDF, 9
jpg, 21

TOTAL FILE NUMBER: 83

COURSE 1.267

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.364

pdf, 6
html, 11
PDF, 1
jpg, 21

TOTAL FILE NUMBER: 39

COURSE 1.432

ppt, 74
xls, 38
zip, 183
exe, 1
html, 955

rtf, 105
PDF, 2
mdl, 154
ZIP, 42
vdf, 6
jpg, 21
DOC, 9
mpp, 1
RTF, 3
MDL, 2

TOTAL FILE NUMBER: 1596

COURSE 1.432F99

ppt, 66
xls, 8
txt, 1
pdf, 1
zip, 110
exe, 1
html, 362
rtf, 2
mdl, 22
ZIP, 13
vdf, 2
jpg, 25
DOC, 1
mpp, 4

TOTAL FILE NUMBER: 618

COURSE 1.44

ppt, 3
xls, 2
pdf, 2
html, 300
rtf, 1
jpg, 21

TOTAL FILE NUMBER: 329

COURSE 1.441

ppt, 8
xls, 376
pdf, 1
html, 998
htm, 31
rtf, 2
jpg, 26
DOC, 16

TOTAL FILE NUMBER: 1458

COURSE 1.46

ppt, 2
pdf, 1
html, 11
jpg, 21

TOTAL FILE NUMBER: 35

COURSE 1.464

ppt, 1
xls, 1
html, 11
htm, 18
PDF, 1
jpg, 21

TOTAL FILE NUMBER: 53

COURSE 1.472

xls, 2
html, 11
jpg, 21

TOTAL FILE NUMBER: 34

COURSE 1.51

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.561

pdf, 20
html, 14
jpg, 21

TOTAL FILE NUMBER: 55

COURSE 1.562

html, 11
htm, 1
jpg, 21

TOTAL FILE NUMBER: 33

COURSE 1.570

xls, 2
pdf, 10
ps, 8

zip, 1
html, 12
htm, 3
jpg, 21
m, 5

TOTAL FILE NUMBER: 62

COURSE 1.571

pdf, 6
html, 12
jpg, 21

TOTAL FILE NUMBER: 39

COURSE 1.691

html, 11
jpg, 21
tex, 2

TOTAL FILE NUMBER: 34

COURSE 1.715

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.77

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.89

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.961

ppt, 1
txt, 23
pdf, 10
ps, 2
zip, 6
html, 48
tar, 1
PPT, 1
jpg, 21

c, 31
gz, 4
TXT, 3
C, 1
sit, 1

TOTAL FILE NUMBER: 153

COURSE 1.962

ppt, 9
pdf, 5
html, 36
htm, 1
jpg, 21

TOTAL FILE NUMBER: 72

COURSE 1.963

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 1.963_F01

ppt, 5
xls, 2
txt, 1
pdf, 2
html, 11
jpg, 21

TOTAL FILE NUMBER: 42

COURSE 1.977

html, 12
htm, 2
jpg, 21

TOTAL FILE NUMBER: 35

COURSE 1.987

txt, 1
html, 11
jpg, 21

TOTAL FILE NUMBER: 33

COURSE 1.999

html, 14
jpg, 21

TOTAL FILE NUMBER: 35

COURSE 11.205

html, 12
htm, 12
jpg, 21

TOTAL FILE NUMBER: 45

COURSE 11.255

txt, 1
html, 38
jpg, 21

TOTAL FILE NUMBER: 60

COURSE 11.361

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 11.362

txt, 3
pdf, 2
html, 11
htm, 5
jpg, 36
dbf, 4

TOTAL FILE NUMBER: 61

COURSE 11.366

ppt, 1
txt, 3
pdf, 8
zip, 1
html, 11
htm, 1
jpg, 29

TOTAL FILE NUMBER: 54

COURSE 11.495

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 11.522

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 11.952

html, 75
jpg, 21

TOTAL FILE NUMBER: 96

COURSE 13.49

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 13.64

xls, 8
zip, 3
html, 28
jpg, 21
tif, 1

TOTAL FILE NUMBER: 61

COURSE 15.001

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 15.012

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 15.012a

ppt, 12
xls, 5
pdf, 1
html, 16
jpg, 21

TOTAL FILE NUMBER: 55

COURSE 15.012b

ppt, 11
xls, 2
html, 17
rtf, 1
jpg, 21

TOTAL FILE NUMBER: 52

COURSE 15.034

html, 11
jpg, 21
DOC, 5
XLS, 7

TOTAL FILE NUMBER: 44

COURSE 15.053

ppt, 20
xls, 3
pdf, 19
ps, 1
html, 14
PDF, 13
jpg, 21

TOTAL FILE NUMBER: 91

COURSE 15.053_Fall_2001

ppt, 2
html, 14
jpg, 21

TOTAL FILE NUMBER: 37

COURSE 15.057

ppt, 22
xls, 11
pdf, 35
html, 87
htm, 2
PDF, 10
jpg, 21

TOTAL FILE NUMBER: 188

COURSE 15.061

ppt, 1
xls, 2
html, 13
jpg, 21

TOTAL FILE NUMBER: 37

COURSE 15.062

ppt, 2
xls, 6
pdf, 23
zip, 1
html, 11
jpg, 21

TOTAL FILE NUMBER: 64

COURSE 15.064J

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 15.064J_LFM03

ppt, 1
html, 11
jpg, 21

TOTAL FILE NUMBER: 33

COURSE 15.064_LFM02

ppt, 1
txt, 2
pdf, 2
html, 11
jpg, 21

TOTAL FILE NUMBER: 37

COURSE 15.066J

ppt, 7
xls, 42
pdf, 18
zip, 1
html, 12
htm, 1
jpg, 21

TOTAL FILE NUMBER: 102

COURSE 15.071

ppt, 3
xls, 2
txt, 1

pdf, 34
html, 14
jpg, 21

TOTAL FILE NUMBER: 75

COURSE 15.081

txt, 3
ps, 44
html, 12
jpg, 21
m, 5
dvi, 2

TOTAL FILE NUMBER: 87

COURSE 15.082J

ppt, 2
pdf, 40
ps, 24
html, 13
PDF, 5
jpg, 21

TOTAL FILE NUMBER: 105

COURSE 15.083

pdf, 11
html, 67
htm, 4
PDF, 13
jpg, 21

TOTAL FILE NUMBER: 116

COURSE 15.083J

pdf, 39
html, 11
jpg, 21

TOTAL FILE NUMBER: 71

COURSE 15.084J

ppt, 12
pdf, 49
ps, 1
html, 12
jpg, 21

TOTAL FILE NUMBER: 95

COURSE 15.099

ppt, 10
pdf, 4
ps, 3
html, 11
jpg, 21

TOTAL FILE NUMBER: 49

COURSE 15.099_F2001

ppt, 7
pdf, 18
ps, 1
html, 12
jpg, 21

TOTAL FILE NUMBER: 59

COURSE 15.227B

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 15.311

html, 17
jpg, 21

TOTAL FILE NUMBER: 38

COURSE 15.390

html, 12
jpg, 21

TOTAL FILE NUMBER: 33

COURSE 15.433

xls, 5
pdf, 29
html, 29
jpg, 21

TOTAL FILE NUMBER: 84

COURSE 15.433_Lo

pdf, 1
html, 11
jpg, 21

TOTAL FILE NUMBER: 33

COURSE 15.433_Pan

html, 26
jpg, 21

TOTAL FILE NUMBER: 47

COURSE 15.760H2

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 15.761

html, 12
jpg, 21

TOTAL FILE NUMBER: 33

COURSE 15.761_LFM

ppt, 18
xls, 1
txt, 1
pdf, 6
exe, 1
html, 12
htm, 2
PDF, 1
jpg, 22

TOTAL FILE NUMBER: 64

COURSE 15.761_SDM

ppt, 36
xls, 18
mdb, 2
pdf, 4
exe, 1
html, 70
htm, 1
PDF, 1
jpg, 25
DOC, 1
XLS, 1
MDB, 2

TOTAL FILE NUMBER: 162

COURSE 15.769

ppt, 16
html, 32
jpg, 21

TOTAL FILE NUMBER: 69

COURSE 15.783

html, 13
jpg, 21

TOTAL FILE NUMBER: 34

COURSE 15.840

html, 14
jpg, 21

TOTAL FILE NUMBER: 35

COURSE 15.875

zip, 1
exe, 1
html, 14
mdl, 1
jpg, 21
vip, 4

TOTAL FILE NUMBER: 42

COURSE 15.876

ppt, 11
xls, 4
txt, 3
pdf, 12
zip, 41
html, 345
mdl, 71
ZIP, 6
jpg, 21
DOC, 2
vmf, 12
VIP, 4
mws, 20
sit, 2
MDL, 1
tif, 2

TOTAL FILE NUMBER: 557

COURSE 15.923

ppt, 11
pdf, 1

html, 14
htm, 1
vdf, 1
jpg, 21
vmf, 1
vgd, 1

TOTAL FILE NUMBER: 51

COURSE 15.960

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 15.962

ppt, 11
xls, 3
pdf, 5
html, 44
jpg, 21
DOC, 1

TOTAL FILE NUMBER: 85

COURSE 15.967

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 15.968

ppt, 14
pdf, 11
html, 29
htm, 4
jpg, 21

TOTAL FILE NUMBER: 79

COURSE 15.980

ppt, 11
xls, 1
pdf, 13
zip, 1
exe, 1
html, 194
htm, 1
PDF, 1
PPT, 1
mdl, 17

vdf, 1
jpg, 21
DOC, 1
vmf, 9
vip, 2
VIP, 4
mws, 4
MDL, 3
jar, 1

TOTAL FILE NUMBER: 287

COURSE 15.981

ppt, 43
txt, 1
pdf, 2
zip, 17
exe, 2
html, 327
htm, 3
PPT, 14
mdl, 19
jpg, 21
ram, 3
RAM, 54
vip, 6
cin, 14
vgd, 1

TOTAL FILE NUMBER: 527

COURSE 15.982

ppt, 11
pdf, 25
zip, 1
html, 289
PDF, 2
mdl, 38
vdf, 3
jpg, 21
vmf, 1

TOTAL FILE NUMBER: 391

COURSE 15.982A

ppt, 1
txt, 1
pdf, 1
zip, 4
exe, 1
html, 56
htm, 3
PDF, 12

jpg, 21
ram, 6

TOTAL FILE NUMBER: 106

COURSE 15.982B

ppt, 11
pdf, 1
zip, 4
exe, 1
html, 11
htm, 2
mdl, 2
ZIP, 5
jpg, 21
ram, 4

TOTAL FILE NUMBER: 62

COURSE 15.982C

ppt, 9
pdf, 1
zip, 1
exe, 1
html, 177
htm, 3
mdl, 6
jpg, 21
DOC, 14
ram, 4
MDL, 1

TOTAL FILE NUMBER: 238

COURSE 15.982D

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 15.983

xls, 4
pdf, 10
zip, 5
html, 535
htm, 15
PDF, 19
mdl, 31
jpg, 23
vmf, 5
MDL, 1

TOTAL FILE NUMBER: 648

COURSE 15.984

ppt, 9
xls, 1
txt, 3
pdf, 23
zip, 1
html, 821
rtf, 13
PDF, 1
jpg, 21
TXT, 2

TOTAL FILE NUMBER: 895

COURSE 15.987

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 15.A03

xls, 1
html, 12
jpg, 21

TOTAL FILE NUMBER: 34

COURSE 16.010

html, 13
jpg, 21
tif, 5

TOTAL FILE NUMBER: 39

COURSE 16.050

html, 12
jpg, 21

TOTAL FILE NUMBER: 33

COURSE 16.070

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 16.221

pdf, 22
zip, 2
html, 11
PDF, 1
jpg, 21

TOTAL FILE NUMBER: 57

COURSE 16.30

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 16.355

ppt, 3
pdf, 14
zip, 1
html, 92
jpg, 21

TOTAL FILE NUMBER: 131

COURSE 16.423J

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 16.83

ppt, 8
html, 13
PDF, 9
jpg, 22

TOTAL FILE NUMBER: 52

COURSE 16.851

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 16.852

ppt, 55
xls, 1
pdf, 10
html, 60
PDF, 1
PPT, 2

jpg, 23

TOTAL FILE NUMBER: 152

COURSE 16.852J

ppt, 75
xls, 15
pdf, 45
html, 134
PDF, 1
jpg, 21
DOC, 1
XLS, 1

TOTAL FILE NUMBER: 293

COURSE 16.872

ppt, 41
xls, 4
pdf, 9
zip, 5
exe, 1
html, 16
htm, 1
PPT, 1
jpg, 21
vsd, 9

TOTAL FILE NUMBER: 108

COURSE 16.880

ppt, 7
xls, 2
pdf, 12
zip, 1
html, 11
htm, 1
PDF, 2
jpg, 21

TOTAL FILE NUMBER: 57

COURSE 16.881

ppt, 22
xls, 3
txt, 1
pdf, 1
html, 16
rtf, 21
jpg, 21
mcd, 11

TOTAL FILE NUMBER: 96

COURSE 16.882

ppt, 48
xls, 7
pdf, 47
zip, 3
html, 403
rtf, 2
PDF, 34
jpg, 24
m, 1
vsd, 1

TOTAL FILE NUMBER: 570

COURSE 16.888

ppt, 48
xls, 4
txt, 1
pdf, 37
zip, 7
exe, 1
html, 29
htm, 1
avi, 1
PDF, 6
jpg, 21
DOC, 1
m, 5

TOTAL FILE NUMBER: 162

COURSE 16.89

ppt, 33
xls, 7
txt, 1
pdf, 6
zip, 3
html, 18
htm, 1
PDF, 5
mdl, 1
ZIP, 2
jpg, 21

TOTAL FILE NUMBER: 98

COURSE 16.89.01

ppt, 10
xls, 1
pdf, 2

html, 76
jpg, 21

TOTAL FILE NUMBER: 110

COURSE 16.899

ppt, 18
xls, 6
pdf, 45
html, 72
htm, 1
PDF, 1
jpg, 23

TOTAL FILE NUMBER: 166

COURSE 16.899a

ppt, 6
pdf, 11
html, 70
htm, 1
jpg, 21
MDL, 1

TOTAL FILE NUMBER: 110

COURSE 16.982

ppt, 4
txt, 1
pdf, 21
ps, 17
html, 211
rtf, 2
jpg, 21

TOTAL FILE NUMBER: 277

COURSE 16.982a

ppt, 2
xls, 1
pdf, 1
html, 76
jpg, 21

TOTAL FILE NUMBER: 101

COURSE 16.983

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 17.253_4

html, 11
htm, 2
jpg, 21

TOTAL FILE NUMBER: 34

COURSE 17.577

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 18.337

txt, 3
html, 40
jpg, 21
gz, 2
dvi, 1

TOTAL FILE NUMBER: 67

COURSE 19.001

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 2.000

ppt, 2
xls, 2
pdf, 5
zip, 1
html, 109
dat, 1
rtf, 1
PDF, 2
jpg, 128
JPG, 2
TIF, 2
wps, 1

TOTAL FILE NUMBER: 256

COURSE 2.007

pdf, 28
html, 13
jpg, 21

TOTAL FILE NUMBER: 62

COURSE 2.009

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 2.010

pdf, 29
html, 13
PDF, 1
jpg, 21

TOTAL FILE NUMBER: 64

COURSE 2.151

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 2.739J

ppt, 16
xls, 24
pdf, 1
zip, 5
html, 94
avi, 5
jpg, 21
efx, 4
tif, 2

TOTAL FILE NUMBER: 172

COURSE 2.744

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 2.82

ppt, 4
xls, 4
pdf, 1
html, 11
htm, 2
PPT, 1
jpg, 21
XLS, 1

TOTAL FILE NUMBER: 45

COURSE 2.830

pdf, 6
html, 11
htm, 1
jpg, 21

TOTAL FILE NUMBER: 39

COURSE 2.875

ppt, 13
xls, 6
exe, 1
html, 22
avi, 1
rtf, 1
jpg, 21
m, 7

TOTAL FILE NUMBER: 72

COURSE 2.882

ppt, 6
pdf, 3
html, 21
jpg, 21

TOTAL FILE NUMBER: 51

COURSE 22.351

xls, 2
txt, 1
zip, 1
exe, 1
html, 24
htm, 6
jpg, 21

TOTAL FILE NUMBER: 56

COURSE 22.38

html, 11
PDF, 12
jpg, 21

TOTAL FILE NUMBER: 44

COURSE 3.14

pdf, 14
html, 11
htm, 6
jpg, 21

TOTAL FILE NUMBER: 52

COURSE 3.21

pdf, 31
html, 11
jpg, 21

TOTAL FILE NUMBER: 63

COURSE 3.23

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 3.52J

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 4.155

html, 18
jpg, 21

TOTAL FILE NUMBER: 39

COURSE 4.156

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 4.182

html, 11
htm, 7
jpg, 21

TOTAL FILE NUMBER: 39

COURSE 4.189

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 4.195

ppt, 2
html, 26
jpg, 21
JPG, 6

TOTAL FILE NUMBER: 55

COURSE 4.199

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 4.272

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 4.290

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 4.351

html, 34
htm, 12
avi, 1
jpg, 23

TOTAL FILE NUMBER: 70

COURSE 4.361

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 4.390

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 4.395

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 4.401

pdf, 10
html, 11
htm, 5
jpg, 21

TOTAL FILE NUMBER: 47

COURSE 6.011

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE 6.046J

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE ESD-999

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE ESD-PD21

ppt, 13
txt, 2
html, 18
htm, 1
rtf, 2
jpg, 21

TOTAL FILE NUMBER: 57

COURSE ESD.000

xls, 14
mdb, 1
pdf, 21
html, 11
PDF, 83
jpg, 30

TOTAL FILE NUMBER: 160

COURSE ESD.121J

xls, 5
pdf, 1
html, 16
jpg, 21

TOTAL FILE NUMBER: 43

COURSE ESD.185J

ppt, 30
xls, 5
pdf, 2
html, 590
PDF, 1
PPT, 1
mdl, 10
jpg, 22
DOC, 2
mpp, 2

TOTAL FILE NUMBER: 665

COURSE ESD.260J

ppt, 19
xls, 804
pdf, 7
html, 821
htm, 3
PDF, 10
PPT, 4
jpg, 21
DOC, 2
XLS, 20

TOTAL FILE NUMBER: 1711

COURSE ESD.263J

ppt, 1
html, 11
jpg, 21

TOTAL FILE NUMBER: 33

COURSE ESD.269J

ppt, 1
xls, 10
pdf, 10
html, 21
PDF, 5

jpg, 21

TOTAL FILE NUMBER: 68

COURSE ESD.33J

ppt, 36
xls, 66
pdf, 9
zip, 6
exe, 1
html, 330
tar, 1
rtf, 1
PDF, 4
PPT, 1
ZIP, 2
jpg, 21
DOC, 1
XLS, 1
mpp, 10
JPG, 1
vsd, 1

TOTAL FILE NUMBER: 492

COURSE ESD.33J.01

ppt, 34
xls, 32
txt, 1
pdf, 34
zip, 1
exe, 1
html, 260
htm, 5
tar, 1
PDF, 9
PPT, 1
jpg, 21
mpp, 3
rdt, 40

TOTAL FILE NUMBER: 443

COURSE ESD.34

ppt, 76
xls, 6
txt, 2
pdf, 11
html, 241
PDF, 13
jpg, 21

TOTAL FILE NUMBER: 370

COURSE ESD.34.SDM-UTC

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE ESD.811

html, 11
htm, 20
jpg, 21

TOTAL FILE NUMBER: 52

COURSE ESD.87J

pdf, 1
html, 14
htm, 18
PDF, 3
jpg, 21

TOTAL FILE NUMBER: 57

COURSE ESD.89

pdf, 1
html, 11
htm, 11
PDF, 2
jpg, 21

TOTAL FILE NUMBER: 46

COURSE icons

TOTAL FILE NUMBER: 0

COURSE MAS964

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE MITCET

ppt, 2
pdf, 3
html, 32
jpg, 21

TOTAL FILE NUMBER: 58

COURSE SMA001

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE SMA002

html, 11
jpg, 21

TOTAL FILE NUMBER: 32

COURSE URIT2000

ppt, 5
xls, 5
pdf, 8
zip, 1
html, 12
htm, 4
dat, 1
PDF, 7
jpg, 21

TOTAL FILE NUMBER: 64

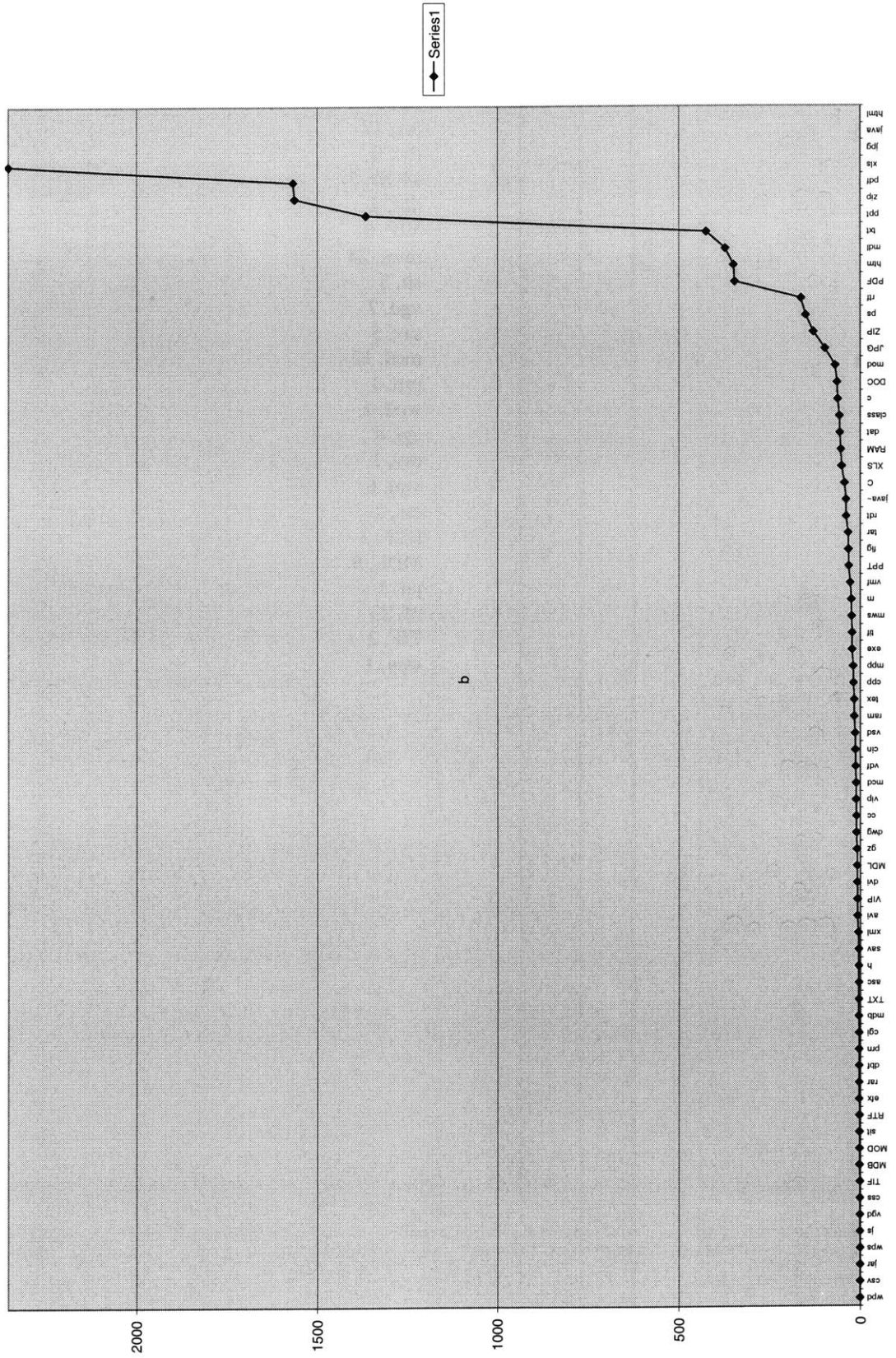
A.1.3.2 File Type Count Totals for the Entire System

ppt, 1366
xls, 2355
mdb, 5
txt, 425
pdf, 1567
ps, 151
zip, 1563
exe, 23
html, 16364

htm, 349
tar, 34
dwg, 11
avi, 8
dat, 57
js, 2
java, 5046
rtf, 164
mod, 70

PDF, 346
PPT, 32
mdl, 372
ZIP, 130
vdf, 13
jpg, 4735
c, 63
gz, 10
TXT, 5
DOC, 65
XLS, 52
m, 25
MDB, 3
ram, 17
RAM, 54
cc, 11
cpp, 19
vmf, 28
vip, 12
cin, 14
efx, 4
mpp, 20
rdt, 40
class, 58
fig, 33
java~, 40
JPG, 98
vsd, 15

asc, 5
C, 44
h, 5
rar, 4
tex, 17
dvi, 9
MOD, 3
dbf, 4
VIP, 8
mws, 24
sit, 3
vgd, 2
sav, 5
mcd, 12
prn, 4
wpd, 1
cgi, 4
csv, 1
xml, 6
css, 2
RTF, 3
MDL, 9
jar, 1
tif, 23
TIF, 2
wps, 1



Appendix Figure 10 File Type

Appendix 2 Designing for Communication: Layout, Structure, and Navigation.

This section presents guidelines for the design of site structure. It also goes on to discuss graphic and text design in the context of graphical user interfaces.

A.2.1 *Structural Guidelines*

Once the features in the application have been determined the next step is to construct the containing structure. The designer should think on structure from the conceptual stage and revisit often his vision of structure.

As the structure begins to take form it is useful to simulate the users experience to the design group and get feedback on shortcomings. Team meetings are a good opportunity to present potential site structures on whiteboards or overheads. In this setting, everyone can contribute and quick changes can be incorporated.

Presenting information in familiar forms helps the learner feel comfortable. Fortunately, there are many classic informational structures that work well. Among these are:

- linear or temporal (e.g., process)
- topical or conceptual (e.g., perspectives)
- hierarchic or categorical (e.g., analysis)
- logical (e.g., persuasion, discussion)
- pedagogical (e.g., tutorial)
- spatial (e.g., organization chart)

The forms of the structure should be varied and take advantage of web possibilities. For example, JavaScript can be used to recreate the Windows 95 desktop and plugins can create interactive structures. Also, the virtual reality modeling language (VRML) can be used to create a varied landscape and environments that can be manipulated.

A.2.2 *Explicate Structure*

A web site structure is itself a form of content. Therefore, the same care that is taken to explain content should be taken to explain structure. Explanations should not just be in the form of graphics but also in the form of strategically placed help through out the system.

The structure should be such that the learner knows where he/she is at all times. Transparent navigation will make the learner feel more secure in the web environment. It will also minimize or obliterate the time spent learning site structure.

Using common standard structural practices help the user by tapping into knowledge he already possesses. Examples are chapters, sections, introductions, headings, and other familiar forms.

Links to the home page are sometimes overlooked by designers. The learner must always be able to go home from any part of the site. This simple feature encourages the user to explore the site with out fear of getting lost.

A.2.3 *Emphasize Central Structure*

Most sites have complex file structures on the developer's side. It is the job of the designer to mask the file structure and present a central, one dimensional, continuous structure

Large jumps in the site structure should be hidden from the average user. Overlooking this point, can create discontinuities in the structure, through which, users unfamiliar with

the site structure can lose their way. Thus, large jumps should be confined to a site map or made inaccessible to new users.

Layering can be used to provide information to the user, on demand. JavaScript permits an additional layer of information to be written into a web page. Designers use this layer to write help on a “onMouseOver” function. The help text is displayed when the user places the mouse over the hidden layer

Finally one must remember that with bookmarks users can access a site at any place. This means that regardless of where a user enters the site he/she must get some sense as to the structure and where he/she is.

A.2.4 *Graphics Design*

Graphic Design considerations:

- organizing the visual field
- guiding the hand and eye within that field
- encouraging exploration by creating an appealing visual environment
 - ◇ unified (and thus comprehensible)
 - ◇ varied (and thus interesting)
 - ◇ balanced (and thus satisfying)

Variables Associated with Graphics:

- Location (relative to screen, other elements)
- Size (typesize, graphic size)
- Style (typeface or artistic style)
- Emphasis or visual weight (density, line weight)
- Texture

- Color
- Whitespace (amount, shape)

There are four principles of graphics design that will be discussed: contrast, repetition, alignment, and proximity.

The *contrast* of graphics can be a powerful tool to convey differences in a graphical user interface. Strong differences in size, style, weight, texture, and color send a clear message to the user saying “this are different”. However, in order to avoid clutter and overwhelming the user, the designer should take care to use strong elements sparingly, try to use them only to point out important features, and create a focal point. This way the user will quickly see the important points with out being distracted by the cosmetic aspects.

Repeating elements can imply unity. Emphasizing consistent elements such as navigational bars, headers, typefaces, colors, icons, locations creates a “feel” for a particular web site. It is the designer's job to create an environment that will be consistent through out the entire site. A web site has achieved this goals if the user can immediately know if he/she is still in the site by looking (not reading) at a page.

Alignment can create visual connections between elements. It aids the organization of a page by directing the eye and it's important both in text justification and the alignment of images. Mixed text alignment should be avoided as it misdirects the eye. Also, alignment at a page edge is undesirable.

Organization can be improved by *grouping* related elements. The natural tendency of a viewer is to look for relationships and the proximity of elements implies relationships. A good example, is a paragraph header, the closer it is to the body of text, the clearer the message that they belong together. In general, try not to use more than 3-5 eye-stoppers groups.

A.2.5 Text Design

Text design (choosing font, style, color, weight) in web environments is limited unless the text is an image. (Users control default font, size choices; HTML specifies only relative sizing, and even specified fonts may not be available). Given this limitations consideration will only be given to:

- Concordant type
(one type family, *little variation* in style, size, weight...)
- Conflicting type
(two different type families, *similar* style, size, weight...)
- Contrasting type
(two different type families, *distinct* style, size, weight...)

In most situations the only true font type variations are serif and sans serif.

Concordant layouts are very soothing and should be used to achieve a calm and formal effect. The opposite practice creates a disconcerting feeling on the user. However, contrasting type elements such as monotype vs. proportional can be used as a simple way to achieve contrast (outline differences) in design.

Web pages differ from software screens and print media in the following ways:

- Variable page size. The designer can not be sure if the user will view material at 600x400, 800x600, 1024x768, 1152x864, or 1280x1024.
- Primacy of the top four inches: people will not scroll. Thus, the top four inches must engage the user.

- Bandwidth limitations of users. In some cases limited bandwidth will eliminate many TCP/IP applications. For example, real time video is virtually impossible to do over 14.4 kbps connection.
- Client Browser setup. Users on occasion may disable Java, JavaScript, and Cookies. Other may choose unusually large or small fonts destroying an otherwise effective layout.
- Version of HTML the client browser will support. Currently only the last releases of the major browsers support dynamic HTML (DHTML) a feature of HTML 4.0. In contrast it is still possible to download browsers that are several generations old and that only support HTML 3.0 or HTML 2.0.

Thus, due to the many configurations available to the user a developer must do extensive testing on different:

- operating systems
- display sizes
- browsers
- bandwidths

A.2.6 *The Graphical User Interface*

The inessential parts of a graphical user interface (GUI) are the page layout, site structure, and site navigation. Using lessons learned from software design the discussion to follow will focus on: 1) the principle of user correctness, 2) the principle of least astonishment, and 3) the principle of user centeredness.

A.2.6.1 The Principle of User Correctness

When it comes to user interfaces, the common expression of retail sales applies: the user is always right. A design can meet every theoretical principle but if users are confused, misled, or disconcerted by the GUI, it is useless. A successful design should require no explanation.

An effective GUI is a difficult task. The designer should not test his own GUI. Designers are not typical users. This is especially true when one person does all the development. The problem lies in familiarity; the designer becomes so familiar with the project that a logical navigation and transparent content are no longer necessary.

A.2.7 *The Principle of Least Astonishment*

Some of the key elements in a predictable user interface are: simplicity, clarity, completeness, consistency, and robustness.

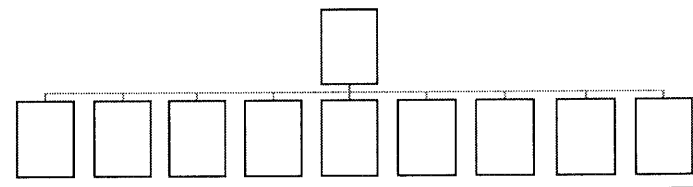
A.2.7.1 Navigational Bars

When designing navigational aids such as bars, preference should be given to simple structures. Some of the least complicated and familiar to users are the horizontal and vertical structures (see below).

Vertical

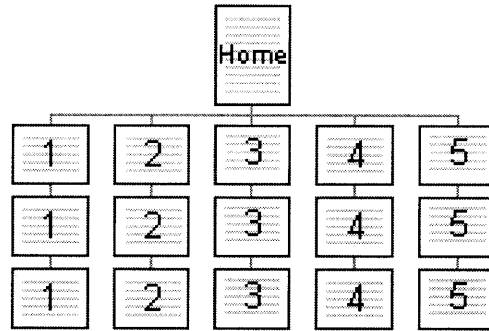


Horizontal



Appendix Figure 11 Navigational Structures

The key is making the structure apparent, easily comprehensible, and keeping the navigation decisions at any point, simple. For example, horizontal and vertical navigation styles could be combined as long as the end result was apparent to a casual user. In the example below, numbers are used to give the user a navigational reference.



Appendix Figure 12 Navigational Aides

Other ways of promoting simplicity are: reducing the number of controls, no more than five is a good principle. Group related controls, separate control groupings that have different functional types, and use contrast to highlight the important controls

A.2.7.2 Clarity

The purpose and use of controls should be immediately clear. Important buttons should dominate the page. The designer should take advantage of web standards such as arrows, text navigation bars, house icon for home button, and rollover scripts. Another important aspect is to make use of the users natural tendencies. For example, people in western cultures search a page left to right and top to bottom. Thus, a good place for an essential piece of information is in the top left hand corner.

A.2.7.3 Completeness

Control buttons for essential features, should always be present. The designer must identify the key functions and incorporate them to every view the user will see. It is also a good idea to provide a link home from every page. In many occasions a user will enter

the site at a random point (as a result of a search or a bookmark) and with out a link to the home page the user can become lost or confused as to its location.

Consideration should also be given to how a page will print. If the content of the pages will be printed often, the designer should take into consideration a color scheme that will be readable in gray scale. A better solution is to have an additional version (different format) of the information for printing purposes only. Some of the common current formats are portable document format and postscript.

In order to create a complete interface the designer must take into account the clients browser. Many browsers do not support the latest versions of HTML and thus may display nothing or incomplete information. One strategy used to avoid this problem is to discover the clients browser when the HTTP server is propositioned. Once the browser is determined the designer can conduct the client to the appropriate web page. Another option is to present a text only welcome page with several browser options.

A.2.7.4 Consistency

The user should be able to know if he is in the web site by looking at the page (not reading it). Similarities can be supported using appearance, placement, and operation. It follows that all aspects of the site should appear and function consistently. Successful implementation will align the designer's conceptual organization with the users perceived site structure.

A.2.7.5 Robustness

Users will commit errors and perform unexpected tasks. Therefore, the designer must anticipate user actions, prevent accidents, and give helpful feedback when errors occur.

A.2.7.6 User-Centeredness

Place the user at the center of your creation.

- Control: give the user control, or the appearance of it
- Responsivity: respond immediately to user actions, and keep him/her informed

- ◇ Keep download times short
- ◇ specify height, width of graphics
- ◇ use fast servers
- Forgiveness: (related to Robustness) assume users will make mistakes, and provide a means to correct them.

Appendix 3 Technology and Education

According to David Perkins the goals of education are very simple: retention, understanding, and active use of the knowledge and skills. The problem with these goals, though simple, is that they are deceptively difficult to attain. This is particularly true when it comes to conceptual themes and abstractions. Students often leave classes confused or with the wrong idea.

What does information technology offer to help solve this problem? How can it help constructivism? Are both concepts working together more than the sum of their parts?

Constructivism is the view that learners are not passive but constantly try to understand the information they are receiving, try to apply it, and make conclusion about it.

The facets of a learning environment can be grouped into five categories: information banks, symbol pads, construction kits, phenomenaria, and task managers.

A.3.1 *Information Banks*

The classic information banks are the professor and texts provided in class. In addition to these there might be films, slides, group presentations, and guest speakers.

Information technology contributes to these course web sites, the web in general, simulations, and the ability to collaborate remotely with experts in the field.

A.3.2 *Symbol Pads*

A notebook is an example of a symbol pad. Basically, any device that will let a student record information for retrieval at a later date.

Lab top computers provide a wealth of resources at the fingertips of the student with out having to leave the classroom. Other tools are cameras, still and video, which can capture

lectures for later review. In case a student misses an important lecture he/she can play it back and catch up.

A.3.3 *Construction Kits*

Laboratory instruments, formwork in a materials laboratory, and model materials in an architecture class are all construction kits. Information technology has had a big impact in this area taking virtual construction (modeling) to new heights.

Some of the most notable contributions are computer aided design and drafting (CADD), computer aided engineering (CAE) modeling, and virtual laboratories and testing facilities in the virtual reality modeling language (VRML).

A.3.4 *Phenomenaria*

Defined by Perkins as an area for the specific purpose of representing phenomena and making them accessible to scrutiny and manipulation.

In the conventional classroom this has been laboratory experiments meant to capture and demonstrate one phenomenon or law of nature. Examples that come to mind are the classic physics experiments “shooting the falling monkey” and the “pendulum”.

Information technology can represent phenomenaria in new and wonderful ways. The construction kits mentioned in the last section when tied to control programs can represent and recreate nature for experimentation. In fact, as the traditional way of experimenting in laboratories is often expensive and time consuming, computer modeling has replaced laboratory testing at many educational institutions.

A.3.5 *Task Managers*

The person or process that controls the learning activities of the student is the task manager. In the typical classroom this person is the professor.

Computer instruction provides the opportunity for programmed instruction control. The designer will take the student on the path he/she deems to be the best.

A.3.6 *Caveats for Applications of Information Technology*

Care should be given when isolating systems. When the designer is creating an application that isolates a part of a system, care must be taken to include some sense of the greater system into the design. "It is not that focused attention ... is disallowed, but rather that whatever focused attention they get should be part of some larger enterprise transparent not just to the teacher but the learner" (source ????).

Assessment is an integral part of learning; hence, computer systems aimed at teaching should provide the user with feedback. Computer systems are currently limited to user input to assess progress. The users mood, expressions, and gestures are challenges that have not been solved by artificial intelligence.

A.3.7 *Information Technology*

The tools discussed undeniably aid instruction. However, are they more than a tool? In the author's opinion they are.

While many tools are simply no more than a symbol pad, others are complex systems that can let a student interact with a system in ways that were previously impossible. Complex simulations of mechanical systems, for example, will let the user observe a system as it changes through time, as material properties are changed, and as changes to initial conditions are made.

Another example is a project that fellow researchers are collaborating on, in structural dynamics. "Teaching and learning in dynamics has great potential for improvement with the use of visual tools provided by computer graphics, especially since the phenomenon under study is time dependent. Computers offer the ability to create animations and interactive figures, which is not possible with current teaching tools such as textbooks and chalkboards. The advantage of using computer graphics is that they are able to

represent the dynamic phenomena adequately and allow the user to experiment with the effect of parameters" (source ???).

A.3.8 Educational Models

A.3.8.1 Theory One

Educators like David Perkins, in “Smart Schools”, list four minimal ingredients for learning to occur: 1) clear information, 2) thoughtful practice, 3) informative feedback, 4) strong motivation.

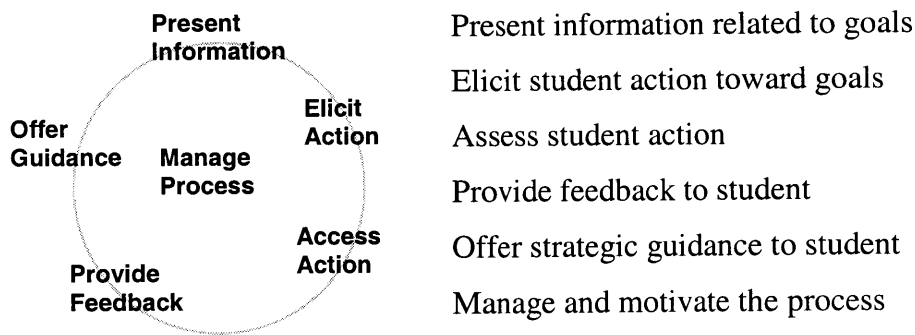
This can be applied to the proposed educational web sites in the following way:

- **Clear information** - Descriptions and examples of the goals, knowledge needed, and the performances expected. Transparent navigation, clean uncluttered graphics, and obvious grouping link sections.
- **Thoughtful practice** - Opportunity for learners to engage actively and reflectively whatever is to be learned. Wizards for common site tasks.
- **Informative feedback** - Clear, thorough counsel about performance, helping learners be more effective. Abundant help and quizzing of material.
- **Strong motivation** - Activities that are rewarded, because they are interesting in themselves, or because they feed other achievements that concern the learner. Animations and simulations.

A.3.8.2 Tutorial Cycle

Furthermore, when it comes to educational software the tutorial cycle as described by George Brackett presents a simple but effective instruction model.

Appendix Figure 13 Tutorial Cycle



These elements can vary in style and implementation. However, for an effective teaching tool all should be included.

A.3.8.3 Present Information

Information will be presented in various forms and categories. The obvious ones: lecture notes, assignments and solutions, readings, and announcements. In addition, discussion sections, calendars, and presentations will also add to class knowledge base.

A.3.8.4 Elicit Student Action Toward Goals

Students will be able to obtain information for the system and once processed the opportunity for online testing will be available.

A.3.8.5 Assess Students Action

The results of online testing will be immediate. In addition a record will be kept in order to measure the effectiveness of the available questions.

A.3.8.6 Provide Feedback to the Student

The results of online testing and simulations will be immediate. Users will be able to know the effect of their actions in real time.

A.3.8.7 Offer Strategic Guidance to the Student

Help Sections will be populated for each course on the system.

A.3.8.8 Manage and Motivate the Process

The user's interaction will be managed by scripts.

From the previous discussion it can be said that the internet and computer networks gives a students four key aids in the educational process: 1) access to information, 2)access to communication, 3)access to remote computing power, 4)access to collaboration.

A.3.9 *Designing Clear Information*

When designing educational content for the web, content guidelines and structural guidelines should established. In past sections the performances of understanding have been described. Now, to be more precise and describe the knowledge, skills, and understandings it is useful to use an input/output analysis modeling the learner, expert, and identify the suboutcomes that reduce the difference.

A.3.10 *Content Guidelines*

A.3.10.1 The Learner

In order to describe the learner we should consider his knowledge, skills, interests, and his strategies for learning.

In order to create an experience that will be fruitful to a user, the designer should know what is the knowledge level and common misconceptions of the learner. This knowledge will permit the designer to create an experience which will be at the correct level of complexity and will strive to point out the common misconception to the learner as he/she goes along.

Knowing the learner skills will allow a design which exploits the users strongest points. For example, for users that hate to read on the screen (as most people do) a graphical approach is better.

One of the key aspects for any project on the web is being able to capture the user's attention. Knowing before hand the interests of learners is a big advantage when designing to capture an audience. The designer should try to exploit the advantages of the medium and make the learning experience *fun*. In addition, to learners preferences, adding pleasing graphics, sound, and animation sections always spices up a web site.

Students in different disciplines have different learning habits. Learners vary a lot from the fine arts to the hard sciences. Thus, a design that targets the different academic cultures will be more effective.

It follows from the previous discussion, that the broader the audience the harder it is to optimize a design.

A.3.10.2 The Experts

In order to build a learning model the designer must answer the same question posed for the "the learners":

- knowledge, both immediate and retrievable, and its interconnections, if possible from protocols, interviews
- skills (what do experts *really do* when they perform, not always obvious)
- strategies for learning, acting in new situations

- interests, world view, etc....

This is important because experts, are the goal of learners.

Suboutcomes

After performing an analysis on learners and experts compare the two. The designer should set aims that support and strengthen similarities, diminish differences, and develop elements missing from learners but present in experts.

A.3.10.3 Structural Guidelines

Once the content has been determined the next step is to construct the containing structure. The designer should think on structure from the conceptual stage and revisit often his vision of structure.

When thinking about the structure the team should use visual aids to layout the suggested structures. Often in team meetings the suggested structure would be drawn as a flow chart or concept map on a white board. This way everyone could contribute and quick changes could be incorporated.

As the structure begins to take form it is useful to simulate the users experience to the design group and get feedback on shortcomings.

Presenting information in familiar forms helps the learner feel comfortable. Fortunately, there are many classic informational structures that work well and are easy to implement.

Among these are:

- linear or temporal (e.g., process)
- topical or conceptual (e.g., perspectives)
- hierarchic or categorical (e.g., analysis)
- logical (e.g., persuasion, discussion)

- pedagogical (e.g., tutorial)
- spatial (e.g., organization chart)

The forms of the structure should be varied and take advantage of web possibilities. Currently, the web presents a wonderful variety of possibilities like audio, text, streaming video, plus 3D environments. Making use of these tools the instructor can deliver content or interact with the learner.

These tools should also be taken advantage of to engage the learner. Virtual environments are very effective in this area. The most successful provide a varied landscape and manipulable landscape.

A.3.11 *Explicate Structure*

A web site structure is itself a form of content. Therefore, the same care that is taken to explain content should be taken to explain structure. Explanations should not just be in the form of graphics but also in the form of strategically placed help through out the system.

The structure should be such that the learner knows where he/she is at all times. Navigation should lead the learner in an obvious path. Transparent navigation will make the learner feel more secure in the web environment. It will also minimize or obliterate the learning curve (learning the structure).

Using common standard structural practices help the user by tapping into knowledge he already possesses. Examples are chapters, sections, introductions, headings, and other familiar forms.

A simple thing to overlook is links to home. The learner must always be able to go home from any part of the site. This simple feature encourages the user to explore the site with

out worrying about getting lost. No matter what happens the user knows he/she can go home.

A.3.12 *Emphasize Central Structure*

Most sites have complex file structures and many structures that are browsable. However, only one central structure should be emphasized to the user. Several techniques can be used to achieve this goal.

Layering can be used to provide on information to the user on demand. JavaScript permits the designer to add additional layers of information to a web page. A common practice of designers is to write help on a “onMouseOver” function. The way this works is that when a user places the mouse over the object in questions, a rectangle appears displaying the help text.

Large jumps in the site structure should be hidden from the average user. When users unfamiliar with the site structure are dropped in the middle of a site they can easily loose their way. Thus, jumping about in the site should be confined to a site map or another place as long as is not the main navigation.

Finally one must remember that with bookmarks users can access a site at any place. This means that regardless of where a user enters the site he/she must get some sense as to the structure and where he/she is.

Appendix 4 WSDL Generic Registration

Sample

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:s1="http://microsoft.com/wsdl/types/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://YourHostName/GenericReg/" xmlns:s2="http://YourHostName/GenericReg/AbstractTypes"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" targetNamespace="http://YourHostName/GenericReg/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://YourHostName/GenericReg/">
      <s:import namespace="http://microsoft.com/wsdl/types/" />
      <s:import namespace="http://www.w3.org/2001/XMLSchema" />
```

```

<s:element name="LogInWithoutSiteId">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="username" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="password" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="LogInWithoutSiteIdResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="LogInWithoutSiteIdResult" type="s1:guid" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="LogIn">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="username" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="password" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="siteId" type="s1:guid" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="LogInResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="key" type="s:base64Binary" />
      <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="LogOut">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="username" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="token" type="s:base64Binary" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="LogOutResponse">
  <s:complexType />
</s:element>
<s:element name="CreateAccount">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="username" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="password" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="siteId" type="s1:guid" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="CreateAccountResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="SaveAttributeValue">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
      <s:element minOccurs="0" maxOccurs="1" name="attName" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="attValue" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="SaveAttributeValueResponse">
  <s:complexType />
</s:element>
<s:element name="SaveUserPicture">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
      <s:element minOccurs="0" maxOccurs="1" name="imageData" type="s:base64Binary" />
      <s:element minOccurs="1" maxOccurs="1" name="length" type="s:int" />
      <s:element minOccurs="0" maxOccurs="1" name="imageType" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="SaveUserPictureResponse">
  <s:complexType />
</s:element>
<s:element name="GetUserPicture">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetUserPictureResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="imageData" type="s:base64Binary" />
    </s:sequence>
  </s:complexType>
</s:element>

```

```

        <s:element minOccurs="1" maxOccurs="1" name="length" type="s:int" />
        <s:element minOccurs="0" maxOccurs="1" name="imageType" type="s:string" />
    </s:sequence>
</s:complexType>
</s:element>
<s:element name="GetUserAttributes">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="username" type="s:string" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="GetUserAttributesResponse">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetUserAttributesResult">
                <s:complexType>
                    <s:sequence>
                        <s:element ref="s:schema" />
                        <s:any />
                    </s:sequence>
                </s:complexType>
            </s:element>
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="AddUserToGroupWithGroupId">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="username" type="s:string" />
            <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
            <s:element minOccurs="0" maxOccurs="1" name="role" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="creationUserName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="status" type="s:string" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="AddUserToGroupWithGroupIdResponse">
    <s:complexType />
</s:element>
<s:element name="AddUserWithUserIdToGroupWithGroupId">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
            <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
            <s:element minOccurs="0" maxOccurs="1" name="role" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="creationUserName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="status" type="s:string" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="AddUserWithUserIdToGroupWithGroupIdResponse">
    <s:complexType />
</s:element>
<s:element name="GetUserIdFromUsername">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="username" type="s:string" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="GetUserIdFromUsernameResponse">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="GetUserIdFromUsernameResult" type="s1:guid" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="IsUserInGroup">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="UserId" type="s1:guid" />
            <s:element minOccurs="1" maxOccurs="1" name="GroupId" type="s1:guid" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="IsUserInGroupResponse">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="IsUserInGroupResult" type="s:boolean" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="CreateGroupRole">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
            <s:element minOccurs="0" maxOccurs="1" name="role" type="s:string" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="CreateGroupRoleResponse">
    <s:complexType />
</s:element>
</s:element>
<s:element name="GetUserRoleInGroup">

```

```

<s:complexType>
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
    <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
  </s:sequence>
</s:complexType>
</s:element>
<s:element name="GetUserRoleInGroupResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetUserRoleInGroupResult" type="s0:ArrayOfAnyType" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:complexType name="ArrayOfAnyType">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="anyType" nillable="true" />
  </s:sequence>
</s:complexType>
<s:element name="UpdateUserRoleInGroup">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
      <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
      <s:element minOccurs="0" maxOccurs="1" name="role" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="UpdateUserRoleInGroupResponse">
  <s:complexType />
</s:element>
<s:element name="CreateUserGroupWithGroupName">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="typeName" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="groupName" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
      <s:element minOccurs="0" maxOccurs="1" name="status" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="newMemberPolicy" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="parentGroupId" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="isMultipleRoleAllowed" type="s:unsignedByte" />
      <s:element minOccurs="0" maxOccurs="1" name="notes" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="areNotesInHTML" type="s:unsignedByte" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="CreateUserGroupWithGroupNameResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="CreateUserGroupWithGroupNameResult" type="s1:guid" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="UpdateGroupByGroupId">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
      <s:element minOccurs="0" maxOccurs="1" name="groupName" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="status" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="newMemberPolicy" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="isMultipleRoleAllowed" type="s:unsignedByte" />
      <s:element minOccurs="0" maxOccurs="1" name="notes" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="areNotesInHTML" type="s:unsignedByte" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="UpdateGroupByGroupIdResponse">
  <s:complexType />
</s:element>
<s:element name="DeleteUserFromGroup">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
      <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="DeleteUserFromGroupResponse">
  <s:complexType />
</s:element>
<s:element name="GetRolesInUserGroup">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetRolesInUserGroupResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetRolesInUserGroupResult" type="s0:ArrayOfAnyType" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetUsersInGroup">

```

```

<s:complexType>
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
  </s:sequence>
</s:complexType>
</s:element>
<s:element name="GetUsersInGroupResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetUsersInGroupResult">
        <s:complexType>
          <s:sequence>
            <s:element ref="s:schema" />
            <s:any />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="ListGroups">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="ListGroupsResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="ListGroupsResult">
        <s:complexType>
          <s:sequence>
            <s:element ref="s:schema" />
            <s:any />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="ListGroupsUserIsIn">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
      <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="ListGroupsUserIsInResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="ListGroupsUserIsInResult">
        <s:complexType>
          <s:sequence>
            <s:element ref="s:schema" />
            <s:any />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetUsernameFromUserId">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetUsernameFromUserIdResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetUsernameFromUserIdResult" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetUsersAttributes">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="UserIds" type="s0:ArrayOfAnyType" />
      <s:element minOccurs="0" maxOccurs="1" name="Attributes" type="s0:ArrayOfAnyType" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetUsersAttributesResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetUsersAttributesResult">
        <s:complexType>
          <s:sequence>
            <s:element ref="s:schema" />
            <s:any />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:sequence>
  </s:complexType>

```

```

        </s:element>
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="GetUsersFromGroupWithThisRole">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
        <s:element minOccurs="0" maxOccurs="1" name="role" type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="GetUsersFromGroupWithThisRoleResponse">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="GetUsersFromGroupWithThisRoleResult">
          <s:complexType>
            <s:sequence>
              <s:element ref="s:schema" />
              <s:any />
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="GetUserGroupFromGroupId">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="groupId" type="s1:guid" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="GetUserGroupFromGroupIdResponse">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="GetUserGroupFromGroupIdResult">
          <s:complexType>
            <s:sequence>
              <s:element ref="s:schema" />
              <s:any />
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="ChangeUserPassword">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
        <s:element minOccurs="0" maxOccurs="1" name="password" type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="ChangeUserPasswordResponse">
    <s:complexType />
  </s:element>
  <s:element name="GetUserAttributesByUserId">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="userId" type="s1:guid" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="GetUserAttributesByUserIdResponse">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="GetUserAttributesByUserIdResult">
          <s:complexType>
            <s:sequence>
              <s:element ref="s:schema" />
              <s:any />
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="guid" type="s1:guid" />
  <s:element name="DataSet" nillable="true">
    <s:complexType>
      <s:sequence>
        <s:element ref="s:schema" />
        <s:any />
      </s:sequence>
    </s:complexType>
  </s:element>
</s:schema>
<s:schema elementFormDefault="qualified" targetNamespace="http://microsoft.com/wsdl/types/">
  <s:simpleType name="guid">
    <s:restriction base="s:string">
      <s:pattern value="[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}" />
    </s:restriction>
  </s:simpleType>

```



```

</s:schema>
<s:schema targetNamespace="http://YourHostName/GenericReg/AbstractTypes">
  <s:complexType name="StringArray">
    <s:complexContent mixed="false">
      <s:restriction base="soapenc:Array">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="String" type="s:string" />
        </s:sequence>
      </s:restriction>
    </s:complexContent>
  </s:complexType>
</s:schema>
</types>
<message name="LogInWithoutSiteIdSoapIn">
  <part name="parameters" element="s0:LogInWithoutSiteId" />
</message>
<message name="LogInWithoutSiteIdSoapOut">
  <part name="parameters" element="s0:LogInWithoutSiteIdResponse" />
</message>
<message name="LogInSoapIn">
  <part name="parameters" element="s0:LogIn" />
</message>
<message name="LogInSoapOut">
  <part name="parameters" element="s0:LogInResponse" />
</message>
<message name="LogOutSoapIn">
  <part name="parameters" element="s0:LogOut" />
</message>
<message name="LogOutSoapOut">
  <part name="parameters" element="s0:LogOutResponse" />
</message>
<message name="CreateAccountSoapIn">
  <part name="parameters" element="s0:CreateAccount" />
</message>
<message name="CreateAccountSoapOut">
  <part name="parameters" element="s0:CreateAccountResponse" />
</message>
<message name="SaveAttributeValueSoapIn">
  <part name="parameters" element="s0:SaveAttributeValue" />
</message>
<message name="SaveAttributeValueSoapOut">
  <part name="parameters" element="s0:SaveAttributeValueResponse" />
</message>
<message name="SaveUserPictureSoapIn">
  <part name="parameters" element="s0:SaveUserPicture" />
</message>
<message name="SaveUserPictureSoapOut">
  <part name="parameters" element="s0:SaveUserPictureResponse" />
</message>
<message name="GetUserPictureSoapIn">
  <part name="parameters" element="s0:GetUserPicture" />
</message>
<message name="GetUserPictureSoapOut">
  <part name="parameters" element="s0:GetUserPictureResponse" />
</message>
<message name="GetUserAttributesSoapIn">
  <part name="parameters" element="s0:GetUserAttributes" />
</message>
<message name="GetUserAttributesSoapOut">
  <part name="parameters" element="s0:GetUserAttributesResponse" />
</message>
<message name="AddUserToGroupWithGroupIdSoapIn">
  <part name="parameters" element="s0:AddUserToGroupWithGroupId" />
</message>
<message name="AddUserToGroupWithGroupIdSoapOut">
  <part name="parameters" element="s0:AddUserToGroupWithGroupIdResponse" />
</message>
<message name="AddUserWithUserIdToGroupWithGroupIdSoapIn">
  <part name="parameters" element="s0:AddUserWithUserIdToGroupWithGroupId" />
</message>
<message name="AddUserWithUserIdToGroupWithGroupIdSoapOut">
  <part name="parameters" element="s0:AddUserWithUserIdToGroupWithGroupIdResponse" />
</message>
<message name="GetUserIdFromUsernameSoapIn">
  <part name="parameters" element="s0:GetUserIdFromUsername" />
</message>
<message name="GetUserIdFromUsernameSoapOut">
  <part name="parameters" element="s0:GetUserIdFromUsernameResponse" />
</message>
<message name="IsUserInGroupSoapIn">
  <part name="parameters" element="s0:IsUserInGroup" />
</message>
<message name="IsUserInGroupSoapOut">
  <part name="parameters" element="s0:IsUserInGroupResponse" />
</message>
<message name="CreateGroupRoleSoapIn">
  <part name="parameters" element="s0:CreateGroupRole" />
</message>
<message name="CreateGroupRoleSoapOut">
  <part name="parameters" element="s0:CreateGroupRoleResponse" />
</message>
<message name="GetUserRoleInGroupSoapIn">
  <part name="parameters" element="s0:GetUserRoleInGroup" />
</message>
<message name="GetUserRoleInGroupSoapOut">

```

```

    <part name="parameters" element="s0:GetUserRoleInGroupResponse" />
</message>
<message name="UpdateUserRoleInGroupSoapIn">
  <part name="parameters" element="s0:UpdateUserRoleInGroup" />
</message>
<message name="UpdateUserRoleInGroupSoapOut">
  <part name="parameters" element="s0:UpdateUserRoleInGroupResponse" />
</message>
<message name="CreateUserGroupWithGroupNameSoapIn">
  <part name="parameters" element="s0:CreateUserGroupWithGroupName" />
</message>
<message name="CreateUserGroupWithGroupNameSoapOut">
  <part name="parameters" element="s0:CreateUserGroupWithGroupNameResponse" />
</message>
<message name="UpdateGroupByGroupIdSoapIn">
  <part name="parameters" element="s0:UpdateGroupByGroupId" />
</message>
<message name="UpdateGroupByGroupIdSoapOut">
  <part name="parameters" element="s0:UpdateGroupByGroupIdResponse" />
</message>
<message name="DeleteUserFromGroupSoapIn">
  <part name="parameters" element="s0:DeleteUserFromGroup" />
</message>
<message name="DeleteUserFromGroupSoapOut">
  <part name="parameters" element="s0:DeleteUserFromGroupResponse" />
</message>
<message name="GetRolesInUserGroupSoapIn">
  <part name="parameters" element="s0:GetRolesInUserGroup" />
</message>
<message name="GetRolesInUserGroupSoapOut">
  <part name="parameters" element="s0:GetRolesInUserGroupResponse" />
</message>
<message name="GetUsersInGroupSoapIn">
  <part name="parameters" element="s0:GetUsersInGroup" />
</message>
<message name="GetUsersInGroupSoapOut">
  <part name="parameters" element="s0:GetUsersInGroupResponse" />
</message>
<message name="ListGroupsWithIn">
  <part name="parameters" element="s0:ListGroupsWithIn" />
</message>
<message name="ListGroupsWithOut">
  <part name="parameters" element="s0:ListGroupsWithResponse" />
</message>
<message name="ListGroupsWithUserIsIn">
  <part name="parameters" element="s0:ListGroupsWithUserIsIn" />
</message>
<message name="ListGroupsWithUserIsInOut">
  <part name="parameters" element="s0:ListGroupsWithUserIsInResponse" />
</message>
<message name="GetUsernameFromUserIdSoapIn">
  <part name="parameters" element="s0:GetUsernameFromUserId" />
</message>
<message name="GetUsernameFromUserIdSoapOut">
  <part name="parameters" element="s0:GetUsernameFromUserIdResponse" />
</message>
<message name="GetUsersAttributesSoapIn">
  <part name="parameters" element="s0:GetUsersAttributes" />
</message>
<message name="GetUsersAttributesSoapOut">
  <part name="parameters" element="s0:GetUsersAttributesResponse" />
</message>
<message name="GetUsersFromGroupWithThisRoleSoapIn">
  <part name="parameters" element="s0:GetUsersFromGroupWithThisRole" />
</message>
<message name="GetUsersFromGroupWithThisRoleSoapOut">
  <part name="parameters" element="s0:GetUsersFromGroupWithThisRoleResponse" />
</message>
<message name="GetUserGroupFromGroupIdSoapIn">
  <part name="parameters" element="s0:GetUserGroupFromGroupId" />
</message>
<message name="GetUserGroupFromGroupIdSoapOut">
  <part name="parameters" element="s0:GetUserGroupFromGroupIdResponse" />
</message>
<message name="ChangeUserPasswordSoapIn">
  <part name="parameters" element="s0:ChangeUserPassword" />
</message>
<message name="ChangeUserPasswordSoapOut">
  <part name="parameters" element="s0:ChangeUserPasswordResponse" />
</message>
<message name="GetUserAttributesByUserIdSoapIn">
  <part name="parameters" element="s0:GetUserAttributesByUserId" />
</message>
<message name="GetUserAttributesByUserIdSoapOut">
  <part name="parameters" element="s0:GetUserAttributesByUserIdResponse" />
</message>
<message name="LoginWithoutSiteIdHttpGetIn">
  <part name="username" type="s:string" />
  <part name="password" type="s:string" />
</message>
<message name="LoginWithoutSiteIdHttpGetOut">
  <part name="Body" element="s0:guid" />
</message>
<message name="LogoutHttpGetIn">
  <part name="username" type="s:string" />

```

```

    <part name="token" type="s2:StringArray" />
</message>
<message name="LogoutHttpGetOut" />
<message name="GetUserAttributesHttpGetIn">
  <part name="username" type="s:string" />
</message>
<message name="GetUserAttributesHttpGetOut">
  <part name="Body" element="s0:DataSet" />
</message>
<message name="GetUserIdFromUsernameHttpGetIn">
  <part name="username" type="s:string" />
</message>
<message name="GetUserIdFromUsernameHttpGetOut">
  <part name="Body" element="s0:guid" />
</message>
<message name="LoginWithoutSiteIdHttpPostIn">
  <part name="username" type="s:string" />
  <part name="password" type="s:string" />
</message>
<message name="LoginWithoutSiteIdHttpPostOut">
  <part name="Body" element="s0:guid" />
</message>
<message name="LogoutHttpPostIn">
  <part name="username" type="s:string" />
  <part name="token" type="s2:StringArray" />
</message>
<message name="LogoutHttpPostOut" />
<message name="GetUserAttributesHttpPostIn">
  <part name="username" type="s:string" />
</message>
<message name="GetUserAttributesHttpPostOut">
  <part name="Body" element="s0:DataSet" />
</message>
<message name="GetUserIdFromUsernameHttpPostIn">
  <part name="username" type="s:string" />
</message>
<message name="GetUserIdFromUsernameHttpPostOut">
  <part name="Body" element="s0:guid" />
</message>
<portType name="GenericUserDatabaseSoap">
  <operation name="LoginWithoutSiteId">
    <input message="s0:LoginWithoutSiteIdSoapIn" />
    <output message="s0:LoginWithoutSiteIdSoapOut" />
  </operation>
  <operation name="Login">
    <input message="s0:LoginSoapIn" />
    <output message="s0:LoginSoapOut" />
  </operation>
  <operation name="Logout">
    <input message="s0:LogoutSoapIn" />
    <output message="s0:LogoutSoapOut" />
  </operation>
  <operation name="CreateAccount">
    <input message="s0:CreateAccountSoapIn" />
    <output message="s0:CreateAccountSoapOut" />
  </operation>
  <operation name="SaveAttributeValue">
    <input message="s0:SaveAttributeValueSoapIn" />
    <output message="s0:SaveAttributeValueSoapOut" />
  </operation>
  <operation name="SaveUserPicture">
    <input message="s0:SaveUserPictureSoapIn" />
    <output message="s0:SaveUserPictureSoapOut" />
  </operation>
  <operation name="GetUserPicture">
    <input message="s0:GetUserPictureSoapIn" />
    <output message="s0:GetUserPictureSoapOut" />
  </operation>
  <operation name="GetUserAttributes">
    <input message="s0:GetUserAttributesSoapIn" />
    <output message="s0:GetUserAttributesSoapOut" />
  </operation>
  <operation name="AddUserToGroupWithGroupId">
    <input message="s0:AddUserToGroupWithGroupIdSoapIn" />
    <output message="s0:AddUserToGroupWithGroupIdSoapOut" />
  </operation>
  <operation name="AddUserWithUserIdToGroupWithGroupId">
    <input message="s0:AddUserWithUserIdToGroupWithGroupIdSoapIn" />
    <output message="s0:AddUserWithUserIdToGroupWithGroupIdSoapOut" />
  </operation>
  <operation name="GetUserIdFromUsername">
    <input message="s0:GetUserIdFromUsernameSoapIn" />
    <output message="s0:GetUserIdFromUsernameSoapOut" />
  </operation>
  <operation name="IsUserInGroup">
    <input message="s0:IsUserInGroupSoapIn" />
    <output message="s0:IsUserInGroupSoapOut" />
  </operation>
  <operation name="CreateGroupRole">
    <input message="s0:CreateGroupRoleSoapIn" />
    <output message="s0:CreateGroupRoleSoapOut" />
  </operation>
  <operation name="GetUserRoleInGroup">
    <input message="s0:GetUserRoleInGroupSoapIn" />
    <output message="s0:GetUserRoleInGroupSoapOut" />
  </operation>

```

```

</operation>
<operation name="UpdateUserRoleInGroup">
  <input message="s0:UpdateUserRoleInGroupSoapIn" />
  <output message="s0:UpdateUserRoleInGroupSoapOut" />
</operation>
<operation name="CreateUserGroupWithGroupName">
  <input message="s0:CreateUserGroupWithGroupNameSoapIn" />
  <output message="s0:CreateUserGroupWithGroupNameSoapOut" />
</operation>
<operation name="UpdateGroupById">
  <input message="s0:UpdateGroupByIdSoapIn" />
  <output message="s0:UpdateGroupByIdSoapOut" />
</operation>
<operation name="DeleteUserFromGroup">
  <input message="s0:DeleteUserFromGroupSoapIn" />
  <output message="s0:DeleteUserFromGroupSoapOut" />
</operation>
<operation name="GetRolesInUserGroup">
  <input message="s0:GetRolesInUserGroupSoapIn" />
  <output message="s0:GetRolesInUserGroupSoapOut" />
</operation>
<operation name="GetUsersInGroup">
  <input message="s0:GetUsersInGroupSoapIn" />
  <output message="s0:GetUsersInGroupSoapOut" />
</operation>
<operation name="ListGroups">
  <input message="s0:ListGroupsSoapIn" />
  <output message="s0:ListGroupsSoapOut" />
</operation>
<operation name="ListGroupsUserIsIn">
  <input message="s0:ListGroupsUserIsInSoapIn" />
  <output message="s0:ListGroupsUserIsInSoapOut" />
</operation>
<operation name="GetUsernameFromUserId">
  <input message="s0:GetUsernameFromUserIdSoapIn" />
  <output message="s0:GetUsernameFromUserIdSoapOut" />
</operation>
<operation name="GetUsersAttributes">
  <input message="s0:GetUsersAttributesSoapIn" />
  <output message="s0:GetUsersAttributesSoapOut" />
</operation>
<operation name="GetUsersFromGroupWithThisRole">
  <input message="s0:GetUsersFromGroupWithThisRoleSoapIn" />
  <output message="s0:GetUsersFromGroupWithThisRoleSoapOut" />
</operation>
<operation name="GetUserGroupFromGroupId">
  <input message="s0:GetUserGroupFromGroupIdSoapIn" />
  <output message="s0:GetUserGroupFromGroupIdSoapOut" />
</operation>
<operation name="ChangeUserPassword">
  <input message="s0:ChangeUserPasswordSoapIn" />
  <output message="s0:ChangeUserPasswordSoapOut" />
</operation>
<operation name="GetUserAttributesByUserId">
  <input message="s0:GetUserAttributesByUserIdSoapIn" />
  <output message="s0:GetUserAttributesByUserIdSoapOut" />
</operation>
</portType>
<portType name="GenericUserDatabaseHttpGet">
  <operation name="LoginWithoutSiteId">
    <input message="s0:LoginWithoutSiteIdHttpGetIn" />
    <output message="s0:LoginWithoutSiteIdHttpGetOut" />
  </operation>
  <operation name="Logout">
    <input message="s0:LogoutHttpGetIn" />
    <output message="s0:LogoutHttpGetOut" />
  </operation>
  <operation name="GetUserAttributes">
    <input message="s0:GetUserAttributesHttpGetIn" />
    <output message="s0:GetUserAttributesHttpGetOut" />
  </operation>
  <operation name="GetUserIdFromUsername">
    <input message="s0:GetUserIdFromUsernameHttpGetIn" />
    <output message="s0:GetUserIdFromUsernameHttpGetOut" />
  </operation>
</portType>
<portType name="GenericUserDatabaseHttpPost">
  <operation name="LoginWithoutSiteId">
    <input message="s0:LoginWithoutSiteIdHttpPostIn" />
    <output message="s0:LoginWithoutSiteIdHttpPostOut" />
  </operation>
  <operation name="Logout">
    <input message="s0:LogoutHttpPostIn" />
    <output message="s0:LogoutHttpPostOut" />
  </operation>
  <operation name="GetUserAttributes">
    <input message="s0:GetUserAttributesHttpPostIn" />
    <output message="s0:GetUserAttributesHttpPostOut" />
  </operation>
  <operation name="GetUserIdFromUsername">
    <input message="s0:GetUserIdFromUsernameHttpPostIn" />
    <output message="s0:GetUserIdFromUsernameHttpPostOut" />
  </operation>
</portType>
<binding name="GenericUserDatabaseSoap" type="s0:GenericUserDatabaseSoap">

```

```

<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
<operation name="LogInWithoutSiteId">
  <soap:operation soapAction="http://YourHostName/GenericReg/LogInWithoutSiteId" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="LogIn">
  <soap:operation soapAction="http://YourHostName/GenericReg/LogIn" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="LogOut">
  <soap:operation soapAction="http://YourHostName/GenericReg/LogOut" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="CreateAccount">
  <soap:operation soapAction="http://YourHostName/GenericReg/CreateAccount" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="SaveAttributeValue">
  <soap:operation soapAction="http://YourHostName/GenericReg/SaveAttributeValue" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="SaveUserPicture">
  <soap:operation soapAction="http://YourHostName/GenericReg/SaveUserPicture" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="GetUserPicture">
  <soap:operation soapAction="http://YourHostName/GenericReg/GetUserPicture" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="GetUserAttributes">
  <soap:operation soapAction="http://YourHostName/GenericReg/GetUserAttributes" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="AddUserToGroupWithGroupId">
  <soap:operation soapAction="http://YourHostName/GenericReg/AddUserToGroupWithGroupId" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="AddUserWithUserIdToGroupWithGroupId">
  <soap:operation soapAction="http://YourHostName/GenericReg/AddUserWithUserIdToGroupWithGroupId" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="GetUserIdFromUsername">
  <soap:operation soapAction="http://YourHostName/GenericReg/GetUserIdFromUsername" style="document" />
  <input>

```

```

        <soap:body use="literal" />
    </input>
</output>
    <soap:body use="literal" />
</output>
</operation>
<operation name="IsUserInGroup">
    <soap:operation soapAction="http://YourHostName/GenericReg/IsUserInGroup" style="document" />
    <input>
        <soap:body use="literal" />
    </input>
    <output>
        <soap:body use="literal" />
    </output>
</operation>
<operation name="CreateGroupRole">
    <soap:operation soapAction="http://YourHostName/GenericReg/CreateGroupRole" style="document" />
    <input>
        <soap:body use="literal" />
    </input>
    <output>
        <soap:body use="literal" />
    </output>
</operation>
<operation name="GetUserRoleInGroup">
    <soap:operation soapAction="http://YourHostName/GenericReg/GetUserRoleInGroup" style="document" />
    <input>
        <soap:body use="literal" />
    </input>
    <output>
        <soap:body use="literal" />
    </output>
</operation>
<operation name="UpdateUserRoleInGroup">
    <soap:operation soapAction="http://YourHostName/GenericReg/UpdateUserRoleInGroup" style="document" />
    <input>
        <soap:body use="literal" />
    </input>
    <output>
        <soap:body use="literal" />
    </output>
</operation>
<operation name="CreateUserGroupWithGroupName">
    <soap:operation soapAction="http://YourHostName/GenericReg/CreateUserGroupWithGroupName" style="document" />
    <input>
        <soap:body use="literal" />
    </input>
    <output>
        <soap:body use="literal" />
    </output>
</operation>
<operation name="UpdateGroupByGroupId">
    <soap:operation soapAction="http://YourHostName/GenericReg/UpdateGroupByGroupId" style="document" />
    <input>
        <soap:body use="literal" />
    </input>
    <output>
        <soap:body use="literal" />
    </output>
</operation>
<operation name="DeleteUserFromGroup">
    <soap:operation soapAction="http://YourHostName/GenericReg/DeleteUserFromGroup" style="document" />
    <input>
        <soap:body use="literal" />
    </input>
    <output>
        <soap:body use="literal" />
    </output>
</operation>
<operation name="GetRolesInUserGroup">
    <soap:operation soapAction="http://YourHostName/GenericReg/GetRolesInUserGroup" style="document" />
    <input>
        <soap:body use="literal" />
    </input>
    <output>
        <soap:body use="literal" />
    </output>
</operation>
<operation name="GetUsersInGroup">
    <soap:operation soapAction="http://YourHostName/GenericReg/GetUsersInGroup" style="document" />
    <input>
        <soap:body use="literal" />
    </input>
    <output>
        <soap:body use="literal" />
    </output>
</operation>
<operation name="ListGroup">
    <soap:operation soapAction="http://YourHostName/GenericReg/ListGroups" style="document" />
    <input>
        <soap:body use="literal" />
    </input>
    <output>
        <soap:body use="literal" />
    </output>
</operation>

```

```

</operation>
<operation name="ListGroupUsersIn">
  <soap:operation soapAction="http://YourHostName/GenericReg/ListGroupsUserIsIn" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="GetUsernameFromUserId">
  <soap:operation soapAction="http://YourHostName/GenericReg/GetUsernameFromUserId" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="GetUsersAttributes">
  <soap:operation soapAction="http://YourHostName/GenericReg/GetUsersAttributes" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="GetUsersFromGroupWithThisRole">
  <soap:operation soapAction="http://YourHostName/GenericReg/GetUsersFromGroupWithThisRole" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="GetUserGroupFromGroupId">
  <soap:operation soapAction="http://YourHostName/GenericReg/GetUserGroupFromGroupId" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="ChangeUserPassword">
  <soap:operation soapAction="http://YourHostName/GenericReg/ChangeUserPassword" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="GetUserAttributesByUserId">
  <soap:operation soapAction="http://YourHostName/GenericReg/GetUserAttributesByUserId" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
</binding>
<binding name="GenericUserDatabaseHttpGet" type="s0:GenericUserDatabaseHttpGet">
  <http:binding verb="GET" />
  <operation name="LogInWithoutSiteId">
    <http:operation location="/LogInWithoutSiteId" />
    <input>
      <http:urlEncoded />
    </input>
    <output>
      <mime:mimeXml part="Body" />
    </output>
  </operation>
  <operation name="LogOut">
    <http:operation location="/LogOut" />
    <input>
      <http:urlEncoded />
    </input>
    <output />
  </operation>
  <operation name="GetUserAttributes">
    <http:operation location="/GetUserAttributes" />
    <input>
      <http:urlEncoded />
    </input>
    <output>
      <mime:mimeXml part="Body" />
    </output>
  </operation>
  <operation name="GetUserIdFromUsername">
    <http:operation location="/GetUserIdFromUsername" />
    <input>

```

```

        <http:urlEncoded />
    </input>
    <output>
        <mime:mimeXml part="Body" />
    </output>
</operation>
</binding>
<binding name="GenericUserDatabaseHttpPost" type="s0:GenericUserDatabaseHttpPost">
    <http:binding verb="POST" />
    <operation name="LogInWithoutSiteId">
        <http:operation location="/LogInWithoutSiteId" />
        <input>
            <mime:content type="application/x-www-form-urlencoded" />
        </input>
        <output>
            <mime:mimeXml part="Body" />
        </output>
    </operation>
    <operation name="LogOut">
        <http:operation location="/LogOut" />
        <input>
            <mime:content type="application/x-www-form-urlencoded" />
        </input>
        <output />
    </operation>
    <operation name="GetUserAttributes">
        <http:operation location="/GetUserAttributes" />
        <input>
            <mime:content type="application/x-www-form-urlencoded" />
        </input>
        <output>
            <mime:mimeXml part="Body" />
        </output>
    </operation>
    <operation name="GetUserIdFromUsername">
        <http:operation location="/GetUserIdFromUsername" />
        <input>
            <mime:content type="application/x-www-form-urlencoded" />
        </input>
        <output>
            <mime:mimeXml part="Body" />
        </output>
    </operation>
</binding>
<service name="GenericUserDatabase">
    <port name="GenericUserDatabaseSoap" binding="s0:GenericUserDatabaseSoap">
        <soap:address location="http://YourHostName:81/GenericReg/GenericUserDatabase.asmx" />
    </port>
    <port name="GenericUserDatabaseHttpGet" binding="s0:GenericUserDatabaseHttpGet">
        <http:address location="http://YourHostName:81/GenericReg/GenericUserDatabase.asmx" />
    </port>
    <port name="GenericUserDatabaseHttpPost" binding="s0:GenericUserDatabaseHttpPost">
        <http:address location="http://YourHostName:81/GenericReg/GenericUserDatabase.asmx" />
    </port>
</service>
</definitions>

```


Appendix 5 Federation

IBM and Microsoft have recently proposed a standard for federation. “A federation is a collection of realms/domains that have established trust. The level of trust may vary, but typically includes authentication and may include authorization.” [Bajaj et al,2003]

Federation is a timely topic with the resurgence of distributed computing, the World Wide Web, and grid computing. Central to federated systems is the problem of authentication across organizations. In practice many of the inter-organization problems are mirrored internally in large organizations. Smaller organizations like universities with diverse information technology infrastructures also mirror inter-organization issues.

The difficulties of authentication stem from several authentication sources. Converging authentication is not always possible when aggregating sources go beyond the organization. The authentication systems need to consider employees, business partners, and customers. Like human relationships, organization’s relationships change often and the cost of managing identities is non-trivial.

The management of identities has privacy and liability issues for organizations. The legal issues make outsourcing very difficult.

The end result for users has been to manage multiple identities. Users have multiple accounts at work, multiple accounts in their private life, multiple accounts with clients, multiple accounts with customers, and multiple accounts for online shopping to name a few of the most prevalent account sources. Managing all these accounts is both a nuisance and a waste of time. Federation management offers an alternative.

Identity management in a federated system offers a number of advantages. Among them are inter-organization flexibility, outsourcing identity management, and business integration. Organizations that offer services to consumers based on identity can couple services through out the enterprise and establish trust with other identity managers to

facilitate the user's experience. Organizations doing business together needing to exchange user's identity information defer to an identity manager and delegate through federation. Federation also is attractive for the supply chain. Organizations implementing enterprise applications need to put employees, customers, and suppliers on the same page. For portals there is a need for the organization to integrate/aggregate services based on identity.

Federation goal is to reduce the cost of identity management by reducing duplication efforts. Instead of every organization reinventing the wheel and developing its own identity management solution, federation offers a standardized approach to the problem. Part of the standard is to leverage existing identity managers. The standard permits organizations to continue using the current identity managers while converging to a more efficient solution. Federation preserves the autonomy of the identity manager using XML messaging and loosely couple connections. The operation of the identity manager can persist unchanged. An XML interface wrapper is added to join the federation. Granular trust relationships are needed for flexibility. The trust relationship can be limited to the trust target and not affect partners or customers as the case maybe.

Formally [Bajaj et al,2003]:

Federation starts with a notion of identity. That is, the requestor or the requestor's delegate (an identity provider who is the authoritative owner for that identity data) asserts an identity and the Identity Provider verifies this assertion. Federation then becomes a function of trust (direct, brokered, and delegated) between identity providers and those relying on the provider's determination of identity. Sometimes the relying party needs the ability to correlate the identities from multiple providers - for example, correlation of an identity on a check, on a credit card, and on a driver's license.

A security token service (STS) is a generic service that issues/exchanges security tokens using a common model and set of messages. As such, any Web service can, itself, be an STS simply by supporting the WS-Trust specification. Consequently, there are different types of security token services which provide different supplemental services. An Identity Provider (IP) is a special type of security token service that, at a minimum, performs peer entity authentication and can make

identity or affiliation claims in issued security tokens. Note that in many cases an IP and STS are interchangeable and many references within this document identify both.

The propose standard includes identity providers, single sign-in and sign-out, and attributes and pseudonyms. Attributes are extensible properties associated with an identity. Attributes are subject to authorization and privacy rules. Pseudonyms are aliases to the identity principal. The pseudonym offers level of obfuscation to the identity.

Future portal work needs to address federation in more levels than security. However, identity management is central to the federation concept and a good place to start.

BIBLIOGRAPHY

- [Anson, 2001]
Anson, el al. "Using Group Support Systems to Facilitate the Research Process." *Proceedings of the Twenty-fifth Annual Hawaii International Conference on Systems Sciences*, January 1992----- 56
- [Bajaj et al,2003]
Bajaj,S. et al "Web Services Federation Language (WS-Federation)." IBM, 8 July, 2003 <<http://www-106.ibm.com/developerworks/webservices/library/ws-fed/>> -----229
- [Blair, 1991]
Blair, G. "Groups that Work." *Engineerin Management Journal*, October, 1991 ----- 56
- [Box, 2000]
Box, D. "A Young Person's Guide to The Simple Object Access Protocol:----- 20
- [Cabrera, 2003]
Cabrera, F. "The Future of Web Services". *Faculty Summit, Microsoft Research*. Redmond, WA: 26 July, 2003 <<https://faculty.university.microsoft.com/2003/Default.aspx>> ----- 17
- [Campione, 2000; Liberty, 2003]
Campione, M. et al. "The Java Tutorial: A Practical Guide for Programmers." *Addison-Wesley*, 15 January , 2000 <<http://java.sun.com/docs/books/tutorial/reflect/>>----- 76
- [Foster et al, 2001]
Foster, I, Kesselman, C., Tuecke, S. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations" *International J. Supercomputer Applications*, 15(3), 2001.----- 11
- [Foster et al, 2002]
Foster, I, Kesselman, C., Tuecke, S. "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration." *Open Grid Service Infrastructure WG, Global Grid Forum*, June 22, 2002. ----- 17
- [Foster, 1985]
Foster R., Linden, L., Whitely, R., Kantrow, A. "Improving the Return on R&D - I,II" *Research Technology Management*, January-April 1985 ----- 152
- [Foster, 1986]
Foster, R. "Innovation: The Attacker's Advantage." *Summit Books*, 1986----- 152
- [Globus, 2003]
"Globus Toolkit." *The Globus Project*. 14 August, 2003 <<http://www-unix.globus.org/toolkit/>>----- 43
- [Hidalgo, 1997]
Hidalgo, C. "Educational uses of the Web : extending a teacher's communication and mediation capabilities through the Internet" *Thesis (M.S.), Massachusetts Institute of Technology, Dept. of Civil and Environmental Engineering*, 1997----- 38

[IBM-WS, 2003]	
"WS-Protocols." <i>IBM</i> , 14, 2003 < http://www-106.ibm.com/developerworks/views/webservices/standards.jsp >-----	156
[Industrial Research Institute, 1993]	
Industrial Research Institute " <i>survey of 248 research directors</i> " May 17, 1993-----	148
[Jacob, 2003]	
Jacob, B., et al. "Enabling Applications for Grid Computing with Globus." <i>IBM Redbook</i> . May,2003 -	47
[JavaSoft, 2003; Obermeyer, 2001]	
"Java Object Serialization Specification" <i>Sun</i> , 14 August, 2003 -----	80
[McConnell, 1996]	
McConnell, S. "Rapid Development: Taming Wild Software Schedules." <i>Microsoft Press</i> ;July 2, 1996	
-----	56
[Meyer & Lehnerd, 1997]	
Meyer, M., Lehnerd, A. "Sharing Technologies and Architectures Across Product Lines The Black and Decker Case." <i>Excerpt from: The Power of Product Platforms</i> . The Free Press, 1997	
< http://web.cba.neu.edu/~mmeyer/research/9/bookchap1/chap1.html > -----	109
[Meyer, 1998]	
Meyer M., Seliger R. "Product platforms in software development." <i>Sloan Management Review</i> , 1998	
-----	117
[Meyer,1997; Baldwin, 2002]	
Meyer M, Lehnerd L. "The Power of Product Platforms." <i>New York: The Free Press</i> , 1997 -----	107
[Offall et al 1996; OMG, 2003]	
Offall R., Harkey D., Edwards J.: "The Essential Distributed Object Survival Guide." Wiley & Sons, 1996. -----	11
[Patterson, 1983]	
Patterson, W. "Evaluating R&D Performance at Alcoa Laboratories." <i>Research Management</i> , March - April, 1983-----	149
[PBS, 2003]	
"Timeline" <i>Public Broadcasting</i> . 14 August,2003 < http://www.pbs.org/opb/nerds2.0.1/timeline/ >-----	44
[Rammer, 2002]	
Rammer, I. "Advanced .NET Remoting" <i>APress</i> , 15 April, 2002-----	83
[Rockwell & Particelli, 1982]	
Rockwell, J., Particelli, M. "New Product Strategy: How the Pros Do It." <i>Industrial Marketing</i> , May 1982-----	148
[Sall, 2002]	
Sall, K. "XML Family of Specifications: A Practical Guide." <i>Addison-Wesley</i> May 31, 2002 < http://kensall.com/big-picture/bigpix22.html >-----	65

[Shohoud,2003]	
Shohoud, Y. "RPC/Literal and Freedom of Choice" <i>MSDN, Microsoft Corporation</i> . April 2003 < http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnwebsrv/html/rpc_literal.asp#rpc_literal_topic1 > -----	80
[SOAPLite-Toolkits, 2003]	
"Toolkits." <i>SOAP::Lite</i> . 14 August, 2003 < http://www.soaplite.com/#TOOLKITS > -----	19
[Sundgren, 1995]	
Sundgren, N. "Introducing interface management in new product family development." <i>Journal of Product Innovation Management</i> , 1995 -----	117
[Tabrizi, 1997]	
Tabrizi B., Walleigh R. "Defining next-generation: An inside look." <i>Harvard Business Review</i> , November-December, 1997-----	108
[UDDI, 2003]	
"Universal Description, Discovery and Integration of Web Services." <i>OASIS</i> . 14 August, 2003 < http://www.uddi.org > -----	22
[Ulrich, 2000]	
Ulrich K., Eppinger S. "Product design, and development." <i>Irwin/McGraw-Hill</i> , 2000 -----	108
[Utterback, 1993]	
Utterback, J. "Mastering the Dynamics of Innovation." <i>Harvard Business School Press</i> , 1993 -----	154
[W3C-Schema, 2003]	
"XML Schema." <i>World Wide Web Consortium</i> . 14 August, 2003 < http://www.w3.org/TR/xmlschema-2/ > -----	76
[W3C-Schema-Types, 2003]	
"Buil in types." <i>World Wide Web Consortium</i> . 14 August, 2003 < http://www.w3.org/TR/xmlschema-2/#built-in-datatypes >-----	78
[W3C-SOAP, 2003]	
"SOAP Primer." <i>World Wide Web Consortium</i> . 14 August, 2003 < http://www.w3.org/TR/soap12-part0/ > -----	19
[W3C-WSDL, 2003]	
"Web Services Description Language." <i>World Wide Web Consortium</i> . 14 August, 2003 < http://www.w3.org/TR/wsdl12-bindings/ > -----	23
[WfMC, 2003]	
"The Workflow Management Coalition." 14, August 2003< http://www.wfmc.org/ > -----	157
[Wheelwright, 1992]	
Wheelwright, S., Clark K. "Revolutionizing new product development." The Free Press, 1992 -----	117
[Williams, 1998a, 1998b, 1990]	
Williams, Anthony. "Object Architecture Closures." <i>Microsoft Corporation</i> December 1988 -----	11

