UNIVERSITY of CALIFORNIA
Santa Barbara

Enabling Content-Driven Applications in Resource-Constrained Environments

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Sami Nicole Rollins

Committee in charge:

Professor Kevin C. Almeroth, Chair
Professor Divyakant Agrawal
Professor Peter Cappello

June 2003

The dissertation of Sami Nicole Rollins is approved:

Divyakant Agrawal

Peter Cappello

Kevin C. Almeroth, Chair

June 2003

Enabling Content-Driven Applications in Resource-Constrained Environments

to my niece and nephew
Stephanie and Aaron Ellison

Acknowledgements

First, I would like to thank my advisor Kevin Almeroth for his guidance and support. He has given me the freedom to explore new ideas but has always helped me to visualize where those ideas would lead.

I would also like to thank Divy Agrawal and Peter Cappello for their support over the past five years. They have helped me to complete this process and achieve my objectives.

Next, I would like to thank the various mentors who have helped me along the way: Dejan Milojicic, Neel Sundaresan, Ellen Spertus, and Barbara Li Santi. They have all provided invaluable encouragement and have helped me to become a better researcher and computer scientist.

I would like to thank my brother, Joe Rollins, for his encouragement and his example. If it weren't for him, I am certain that I would not be where I am today.

I would also like to thank my parents, Nina and Joe Rollins. They have always believed in me, encouraged me, and supported my decisions.

Finally, I would like to thank my partner in life, Karlo Berket. My last two years as a graduate student have tested me in many ways. Karlo has helped to keep me grounded and helped me to prioritize the pieces of my life. His support and encouragement have given me the resolve to complete my degree and realize my career goals.

Thank you.

Curriculum Vitæ

Sami Nicole Rollins

Personal

Name	Sami Nicole Rollins
Email	srollins@cs.ucsb.edu

Education

1998–2003	Ph.D. in Computer Science, University of California, Santa Barbara. Dissertation: Enabling Content-Driven Applications in Resource-Constrained Environments
1998–2000	M.S. in Computer Science, University of California, Santa Barbara. Thesis Topic: Audio XmL: Aural Interaction with XML Documents
1995–1998	B.A. in Computer Science, Mills College.

Research Experience

2002 – 2003	Research Assistant – UC Santa Barbara
Summer 2001	Intern – Hewlett Packard Laboratories
2000 – 2001	Research Assistant – UC Santa Barbara
1999 – 2000	Intern – IBM Almaden Research Center
Summer 1998	Intern – IBM Almaden Research Center

Teaching Experience

Fall 2002	Teaching Assistant – UC Santa Barbara Introduction to Computer Communication Networks

Summer 2002	Instructor – UC Santa Barbara
	Introduction to Programming in Java
Spring 2000	Teaching Assistant – UC Santa Barbara
	Programming Methods
1996 – 1998	Teaching Assistant – Mills College
	Introductory Programming in C++ and Java

Journal Publications:

S. Rollins and N. Sundaresan, "AVoN Calling: AXL for Voice-enabled Web Navigation", Computer Networks, Volume 33, Issues 1-6, pages 533-551, June 2000.

M. Neary, S. Brydon, P. Kmiec, S. Rollins, and P. Cappello, "Javelin++: Scalability Issues in Global Computing", Concurrency: Practice and Experience, vol. 12, pages 727-753, 2000.

Conference Publications:

J. Humfrey, S. Rollins, K. Almeroth, and B. Bimber, "Managing Complexity in a Networked Learning Environment ", in *Proceedings of the World Conference on Educational Multimedia, Hypermedia, and Telecommunications (ED MEDIA 2003)*, Honolulu, Hawaii, USA, June 2003.

K. Almeroth, S. Rollins, Z. Shen, and B. Bimber, "Creating a Demarcation Point Between Content Production and Encoding in a Digital Classroom ", in *Proceedings of the World Conference on Educational Multimedia, Hypermedia, and Telecommunications (ED MEDIA 2003)*, Honolulu, Hawaii, USA, June 2003.

S. Rollins and K. Almeroth, "Pixie: A Jukebox Architecture to Support Efficient Peer Content Exchange", in *Proceedings of ACM Multimedia*, Juan Les

Pins, France, December, 2002.

S. Rollins, R. Chalmers, J. Blanquer, and K. Almeroth, "The Active Information System (AIS): A Model for Developing Scalable Web Services", in *Proceedings of Internet and Multimedia Systems and Applications (IMSA 2002)*, Kauai, HI, August 2002.

S. Rollins and K. Almeroth, "Seminal: Additive Semantic Content for Multimedia Streams", in *Proceedings of Internet and Multimedia Systems and Applications (IMSA 2002)*, Kauai, HI, August 2002.

S. Rollins and K. Almeroth, "Deploying an Infrastructure for Technologically Enhanced Learning", in *Proceedings of the World Conference on Educational Multimedia, Hypermedia, and Telecommunications (ED MEDIA 2002)*, Denver, Colorado, USA, June 2002.

S. Rollins and N. Sundaresan, "A Framework for Creating Customized Multi-Modal Interfaces for XML Documents", in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2000)*, New York City, NY, July 2000.

S. Rollins and N. Sundaresan, "AVoN Calling: AXL for Voice-enabled Web Navigation", in *Proceedings of the 9th International World Wide Web Conference (WWW9 2000)*, Amsterdam, Netherlands, May 2000.

M. Neary, S. Brydon, P. Kmiec, S. Rollins, and P. Cappello, "Javelin++: Scalability Issues in Global Computing", in *Proceedings of the ACM 1999 Java Grande Conference*, pages 171 - 180, San Francisco, California, June 12-14, 1999.

Workshop Publications and Presentations:

S. Rollins, K. Almeroth, D. Milojičić, and K. Nagaraja, "Power-Aware Data Management for Small Devices", *Workshop on Wireless Mobile Multimedia (WoWMoM 2002)*, Atlanta, GA, USA, September, 2002.

S. Rollins and K. Almeroth, "A Model for Distributed Collaboration in a Distance Learning Application", Poster Presentation, *Workshop on Networked Group Communication (NGC 2000)*, Palo Alto, CA, USA, November, 2000.

Non-refereed Publications:

S. Rollins and K. Almeroth, "Lessons Learned Deploying a Digital Classroom", December, 2002.

S. Rollins and K. Almeroth, "Evaluating Performance Tradeoffs in a One-to-Many Peer Content Distribution Architecture", November 2002.

D. Milojičić, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z, Xu, "Peer-to-Peer Computing", HP Labs Technical Report HPL-2002-57, March, 2002.

Professional Activities and Service:

Co-Editor – Peer-to-Peer subtopic area of IEEE Distributed Systems Online `http://dsonline.computer.org/os/related/p2p/index.htm` (Fall 2001 – present)

Co-Web Chair – IFIP/IEEE International Conference on Management of Multimedia Networks and Services 2002

Founder and organizer of quarterly lunch for female graduate students and faculty (Winter, Spring 2002)

Graduate student representative – Graduate Admissions Committee (2000 – 2001)

Volunteer – Expanding Your Horizons in Science and Math Conference (Spring 1997)

Student experiment volunteer, workshop on Teaching and Learning Object Design in the First Academic Year (OOPSLA 1996)

Honors and Awards:

Nomination for Best Student Paper – ACM Multimedia 2002

Outstanding Paper Award – ED MEDIA World Conference on Educational Multimedia, Hypermedia, and Telecommunications 2002

Travel Scholarship – Grace Hopper Celebration of Women in Computing 2002

Doctoral Scholars Fellowship – UC Santa Barbara (Fall 1998 – Spring 2002)

Member – Phi Beta Kappa

Arthur Vining Davis Scholarship – Mills College (Fall 1995 - Spring 1998)

Abstract

Enabling Content-Driven Applications in Resource-Constrained Environments

by

Sami Nicole Rollins

The vision of pervasive computing is quickly becoming a reality. Unfortunately, the range of applications supported by devices such as personal digital assistants (PDAs), digital watches, and iPod-like devices remains somewhat limited. Many current efforts promise to support services such as web browsing, personal file management, and peer content exchange on these devices. However, supporting these kinds of *content-driven* applications in the next-generation computing environment promises to be a challenge. Most pervasive devices are constrained with respect to resources such as bandwidth, processing power, disk space, energy supply, and even display capabilities. These constraints often render traditional technical solutions insufficient for supporting functionality such as content location, delivery, and display. In order to evolve current applications to meet the changing demands and limitations of next-generation computing environments, these challenges must be addressed.

The goal of this dissertation is to develop system-level techniques to overcome resource constraints that restrict the use of pervasive devices for content-driven applications such as web browsing and peer content exchange. In particular, we focus on enabling three subfunctions of content-driven applications: content access, content management, and content exchange. First, we describe a technique to enable content access on devices with limited display capabilities. Second, we describe a technique to enable management of personal content in a power-constrained environment. Finally, we describe a technique that enables new services and improved performance in peer-style content ex-

change networks. This set of techniques represents a stepping stone in the evolution from the current computing environment to the next-generation, pervasive computing environment.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past several years, computing technology has become faster, more powerful, and *smaller*. Moreover, networking technology has become faster, more powerful, and *wireless*. Devices such as laptop computers, personal digital assistants (PDAs), and digital watches that can seamlessly connect to the Internet, or to other devices, are becoming more commonplace. However, applications for this kind of technology are not advancing as quickly as the technology itself. There is a call to evolve current applications and to develop new applications to take advantage of new technology as well as overcome the new technological challenges that next-generation devices introduce.

In this chapter, we introduce a class of application we define as *content-driven*. We explore how new computing devices enable new, next-generation computing environments, and how new environments can affect content-driven applications. Next, we explore the challenges that must be addressed in order to enable content-driven applications in next-generation computing environments. Finally, we identify the scope of this work and outline the organization of the remainder of this dissertation.

1.1 *Content-Driven* Applications

This dissertation defines a *content-driven* application as an application with the fundamental motivation of bringing together users and information. With the proliferation of personal computers and networking technology, these kinds of applications are figuring more prominently in the everyday lives of end users. An example application is web-based news. Users may visit websites like CNN and MSNBC many times a day to keep abreast of late breaking news stories. Other example applications include email and web-based radio.

This definition of *content-driven* is quite broad in scope. However, to further refine the definition of a content-driven application, we specify three fundamental functions that are supported by the applications we target:

Content Access. Content access defines how an end-user views or otherwise consumes information. A user accessing web-based news is likely to rely on a web browser to parse HTML code and present a combination of visual text and graphics on a computer screen. As other forms of media, such as live video, have become more popular, users rely on content-specific software to provide access to information in a content-specific way. For example, an audio and/or visual file may launch a viewer such as QuickTime.

Content Management. Content management defines how information is best stored and managed by the the information source or content provider (e.g., CNN). The definition of "best" can vary from application to application. However, typically the goal of an effective content management scheme is to ensure that content is stored such that it can be quickly and easily accessed by the end user. A website such as CNN might employ techniques such as replicating content on multiple servers and/or placing content on servers that are located closer to the end user.

Content Exchange. Content exchange involves the actual transfer of content from the information source to the end user. Certainly, intelligent content management can help to alleviate some of the burden on the content exchange scheme. However, content exchange is primarily concerned with two tasks: (1) location of the desired information and (2) delivery of information across the network. In the web news application, a user might use a search engine to locate a particular story of interest. Once the user finds the story, the content is typically delivered across the network using HTTP over a reliable unicast protocol such as TCP. In the web news example, this function could also be considered *content distribution*. However, content exchange is a generalization of content distribution and assumes that a participant many not necessarily only act as either a content provider or end user, but perhaps both.

In general, it can be very challenging to support access, management, and exchange of content. For an application such as news, an ever growing user population and an ever increasing content base make it difficult to efficiently and effectively satisfy all user requests. However, for many common web and Internet-based applications, the challenges that arise in providing these functions have been relatively well studied and usable solutions have been developed [6, 13, 20, 23, 80].

1.2 Evolving Computing Environments

Over the past several years, computing environments have begun to evolve. Next-generation computing environments promise to incorporate small computing devices such as PDAs and digital watches along with new networking technology such as 802.11 and Bluetooth. While these environments offer a host of new opportunities for content-driven applications, the traditional web model that has often been used to enable access, management, and exchange of information will no longer be sufficient.

The Web. In the web environment, an end user generally uses his/her home, desktop computer to access content that is retrieved, over the wired Internet, from a content provider. The content provider is often a company or other organization that uses powerful, centrally maintained and administered servers to manage and distribute information. In this environment, content-driven applications are typically implemented using a centralized, client/server model. The underlying assumption with the web model is that the user acts as a content consumer, relying on the content provider to reliably and efficiently provide information.

Peer-to-Peer. As end user desktop computers have become more powerful, a new kind of computing environment has emerged. Content providers are no longer exclusively organizations with centrally maintained and administered servers. In fact, any end user device can act as a content provider, managing and distributing information; a content consumer, accessing information; or both. To support applications in this environment, a decentralized, peer-to-peer (P2P) computing model has emerged. In the P2P model, all participants are considered *peers* – heterogeneous nodes with varying capabilities that may join and leave the network at will. Decentralized and resilient algorithms that enable users to communicate directly and exchange information have begun to develop. However, the general assumption is that the P2P model supports desktop-to-desktop communication over the wired Internet.

Pervasive Computing. The vision of pervasive computing [67, 75] goes beyond P2P to imagine a world in which computing devices are everywhere. Small computing devices may be carried, worn, or even embedded into the environment. Moreover, these devices can seamlessly communicate using wireless technology. To support content-driven applications in this kind of an environment, the goal of the pervasive computing model is to provide completely seamless access, management, and exchange of information in an ad hoc and dynamic way. *Anytime, anywhere* a user should be able to use whatever device

is available to access whatever piece of information he/she is interested in. It is clear that the pervasive computing environment introduces a host of new challenges that the pervasive computing model has yet to fully address.

1.2.1 Enabling New Applications

New environments coupled with new computing models such as P2P and pervasive computing both enable enhanced functionality for existing applications, as well as enable completely new applications. The ultimate vision is that applications can take advantage of the property that anywhere, at any time, users can access and exchange information. A frequently used, motivating example demonstrates how a business person can take advantage of the pervasive computing model to have seamless access to her personal information. She wakes up and checks her email on her home desktop computer. As she leaves home for the office, she picks up her laptop, cell phone, PDA, and digital watch. Her email along with any other content she may have worked on at home the previous evening is automatically transferred from her desktop to the devices. As she sits on the subway train, her PDA exchanges news articles with the devices the other riders are carrying. When she arrives at the office, her devices download new email from the servers embedded into the office environment as she runs to a meeting. As she arrives at the conference room for the meeting, her laptop automatically detects the conference room printer so that our user can print the slides she plans to use for her presentation. She wraps up her work day early to leave for a conference. As she leaves the office for the airport, her conference presentation along with all the data she has been recently working on is downloaded onto her devices so that she can continue working on the plane.

Another motivating application is disaster recovery. Sensor devices may be embedded into the structure of a building when it is constructed. If a disaster, such as an earthquake, were to occur, the devices can seamlessly communicate to determine the soundness of the structure. Additionally, relief

workers or robots carrying PDA-like devices can monitor the scene and gather information from the sensor devices. Based on the information produced by the sensors, plans for recovery efforts can be developed.

A third example application is a digital classroom [55]. In the classroom of the future, instructors will be able to prepare digital material that can be displayed in class, or can be digitally exchanged with local or remote students who are using technology from laptops to digital watches. The pervasive computing model opens up a world of opportunity for enabling students and instructors to seamlessly communicate and share information, fostering collaboration across time and space. We save further explanation of this application for Chapter 2 where we describe the classroom as a motivating application in more detail.

1.3 New Challenges for Evolving Computing Environments

While new computing environments provide a world of opportunity for content-driven applications, they also introduce a host of new challenges that make it vastly more difficult to support access, management, and exchange of content. The challenges we define fall into two primary categories: *capability* and *availability*.

Capability. Capability describes the functions that a content provider or consumer is able to perform. As we move toward the P2P and pervasive environments, content providers are no longer large mainframe or server machines, and content consumers are no longer home desktop computers. They may be anything from laptops to embedded sensors. These smaller devices are limited in terms of resources such as CPU, display, memory, bandwidth, disk space, and power. While technology is continuously becoming more powerful, each generation of devices is smaller and more constrained than the previous.

Additionally, resources like battery power and display will continue to be constrained. These limitations restrict the functions that the devices are able to perform. Content displayed for the user may have to be transcoded into an alternate format to meet the constraints of the display; loss of power on a device may mean that a user can no longer access important information stored on that device; or limited bandwidth may prevent a device from uploading or downloading the desired quantity of information. In this dissertation, we focus on the challenges associated with the reduced capability of next-generation devices.

Availability. Availability describes the likelihood that a content consumer or provider is reachable. In the P2P and pervasive models, availability becomes more of a challenge for a number of reasons. First, end-point computers and devices are dynamically connected. Much of the time, nodes join and leave the network frequently either because users are mobile, or simply choose to disconnect. Further, networks, especially wireless networks, can be lossy, slow, and otherwise unpredictable. Connectivity problems may occur, through no fault of the user, leaving participants unavailable. Finally, resource constraints of devices, especially small devices, can lead to lower availability. Suppose a laptop computer runs out of battery power. It will no longer be available as a participant.

1.4 Scope of Work

This dissertation addresses the challenges associated with providing content access, management, and exchange in an environment where devices have reduced capability. The thesis of this dissertation is that system-level techniques can be used to overcome the resource constraints that restrict the use of next-generation devices for content-driven applications. We focus on three primary areas:

Content **access** is limited by the display constraints of a particular de-

vice. As devices become smaller, the visual access we are accustomed to is no longer a viable solution in all cases. In order to support access to information using a variety of devices, more flexible solutions that can adapt to varying display capabilities must be developed. Ideally, a user should be able to access information in a device-specific way based on the type of device he/she has available.

Content **management** is also a challenge when the devices storing content are resource-constrained. Specifically, we address the problem of managing content across a collection of personal devices as in the previous example of the business person. The loss of a resource, such as power, can make all content on a particular device inaccessible unless the content has been moved prior to resource loss. Relying on the user to manage data may not be effective, or worse, the user may waste the same resources which are scarce. In addition, as the shear number of devices increases, a user cannot manually manage content across possibly tens or hundreds of personal devices.

Finally, content **exchange** is a challenge in a resource-constrained environment for a number of reasons. In this work, we focus on two main problems and their application in peer-based environments. First, content location in peer environments is typically not static as it is in the web model. In a more dynamic environment, locating content is often done on-the-fly and may require use of the resources of a number of participants, content providers and requesters. Participants limited by resources such as power or bandwidth may not be able to participate in an unlimited way. Thus, developing data location techniques that consume fewer resources in integral. Additionally, delivering content in a straightforward, one-to-one fashion to all users that request it can require more resources than a content provider may have available. Scalable, resource-aware techniques must be employed to ensure that delivery is done in an efficient manner.

1.4.1 Organization

This dissertation is organized as follows: In Chapter 2, we discuss a digital classroom as a motivating application and point out the challenges that arise in that context. In Chapter 3, we survey related work and provide a more concise description of our contributions. Chapter 4 describes our work on enabling content access via non-traditional displays. Chapter 5 describes our work on resource-aware data management for small devices. Chapter 6 describes Pixie, a jukebox-style architecture to support efficient data location and delivery in large-scale peer environments. We conclude in Chapter 7.

Chapter 2

The Next-Generation Classroom

The goal of this chapter is to motivate the work in this dissertation by describing the challenges associated with implementing a digital classroom application. The chapter describes what a digital classroom is, explores how the content-driven functions of access, management, and exchange are supported in the classrooms of today, and illustrates the challenges of supporting the same functional objectives in the classroom of the future.

2.1 Defining a Content-Driven Application in a Digital Classroom

A number of university campuses have undertaken the goal of developing *digital classrooms*. One of the earliest experiments with this kind of technology was the AT&T Learning/Teaching Theater at the University of Maryland [69]. More recent examples include 405 Soda at UC Berkeley [77] and Georgia Tech's eClass [1]. A *digital classroom* is a classroom meeting space that provides both the ability to present information using multimedia tools as well the capability to digitally record an account of the classroom activity. A digital classroom can be considered a content-driven application because the primary goal of the classroom is to bring together users and information. Typically, the users are

the students and the information is the material presented by the instructor. As technology advances, the material presented by the instructor has become more sophisticated, evolving from chalkboard notes and illustrations to digital material. Additionally, information now extends beyond visual aids (such as PowerPoint slides) to include a complete digital record of classroom activity, including video and audio.

Digital classrooms today typically aim to support four primary activities:

Multimedia Presentation. In a traditional classroom, an instructor typically uses a chalkboard to write notes or hand-draw illustrations to clarify the material presented. In a digital classroom, computers and display technology, such as data projectors, can be used to display an array of digital material for students in the classroom. Multimedia presentation enhances the instructor's ability to illustrate and demonstrate the material.

One-Way Webcasting. In a traditional classroom, students must be physically present in order to attend the class. In a digital classroom environment, the classroom activity can be digitally captured using technology such as cameras and microphones and sent, via the Internet, to a remote site. A remote site might be a student's dorm room where he/she is watching the lecture on a desktop computer, or may be a remote classroom. One-way webcasting completely redefines the meaning of the term classroom.

Remote Collaboration. In a traditional classroom, students can ask questions of the instructor and interact with other students at will. To enable this functionality in a digital classroom, activity at the classroom site and remote sites must be digitally captured, using technology such as cameras and microphones, and exchanged between sites in real time. Remote collaboration enables a complete *distance education* scenario.

Archival/Retrieval. Finally, in a traditional classroom, students rely on their handwritten notes to help them study for exams or later review the material. In a digital classroom, a digital record of the classroom activity and material can be archived and later retrieved and reviewed. In addition, a lecture given by an "expert" may be reviewed by anyone interested in gaining knowledge in the field [74]. Archival and retrieval of material can fundamentally change the role of the instructor in the classroom.

2.2 Classroom Support for Functional Objectives

It is difficult, even today, to support the functions required of a content-driven application in the classroom environment. As new technology such as PDAs, laptop computers, and wireless networking becomes more ubiquitous, integrating the technology into the classroom application to support more advanced activities will be even more challenging. This section describes how the functional objectives of access, management, and exchange of content are met in the classroom application today and the new challenges that arise with regard to supporting these functions in the future.

2.2.1 Access

Content access in a digital classroom of today consists of two main pieces. First, the technology to display multimedia material must be available in the classroom. In most cases, this includes a computer to use to present material and a data projector to project the image from the computer screen. The main challenge in supporting this functionality lies in obtaining the right equipment and connecting it together in the right way. The second piece is the display of material such as audio and video at a remote site. Generally, the second piece is accomplished using off-the-shelf tools such as RealMedia. Though, it

can be challenging to ensure that off-the-shelf tools provide all of the intended functionality.

In the classroom of the future we can imagine that, in addition to large, projected displays and/or desktop displays, each student may have one or a collection of devices that he/she might like to use to access information the instructor is presenting. A student might want to download slides onto his/her PDA and take notes there while displaying a digital worksheet on his/her laptop computer. The next student may want to take notes on his/her Tablet PC while looking at the worksheet on a cell phone. Each of these devices has slightly different display and input/output capabilities and is used in a slightly different way, potentially to access the same material. One of the primary challenges in this environment is to meet all of the demands of the different devices. In Chapter 4, we discuss a general solution that enables multimodal access to content via non-traditional displays.

2.2.2 Management

The main goal of content management in the classroom of today is to ensure that prepared materials, such as slides, and generated media, such as video, are stored and later made available to the user. Generally, the user population for the classroom application is sufficiently small that this task is not particularly challenging. A single, centrally located, powerful server can often accomplish the task. However, one of the biggest remaining challenges in this area is the indexing of material to ensure that it can be synchronized for later retrieval and viewing.

In the classroom of the future, content may extend beyond slides prepared by the instructor and video generated by an infrastructure embedded into the classroom. With the proliferation of personal devices, students may use their laptops, PDAs, and cell phones to contribute to the classroom content base. A student may share notes from a PDA and a cached web page from a laptop computer. The challenge in this environment is to ensure that student-

generated material is available even if it is stored on a cell phone rather than a powerful server. In Chapter 5, we discuss techniques for managing content across a collection of resource-constrained devices.

2.2.3 Exchange

In the classroom of today, content exchange involves delivering archived content from servers as well as delivering or exchanging live video, audio, or other media in real time between the classroom and remote sites. Delivering archived content is generally straightforward. However, delivering live content can be challenging. Often, to overcome jitter caused by the network, playout tools buffer video and audio. The delay caused by buffering is intolerable if the classroom and remote sites are collaborating in real time.

Looking toward the future, we can imagine a world in which students are all capable of generating and distributing content. Therefore, the task of distributing content will no longer be restricted to a centrally located and administered server machine. Students may want to share content available on their hand-held or wearable devices. Delivering content in this environment will be challenging since techniques must be geared toward resource-constrained devices. Imagine a student serving a video file to many classmates, from her watch. In Chapter 6, we propose a content exchange solution that provides a more scalable alternative to current systems.

The digital classroom is a prime example of a content-driven application. Not only do the application participants bring content into the environment, the environment itself can actually generate content. While supporting the functions of a content-driven application can be challenging in the digital classroom environment today, future computing technology promises to make supporting access, management, and exchange of content even more challenging. The goal of this dissertation is to develop techniques to overcome some of these challenges and enable applications such as digital classrooms in next-generation computing environments.

Chapter 3

Related Work

The goal of this dissertation is to address the challenge of enabling content-driven applications in the face of constrained resources. There are three fundamental functions that support content-driven applications; content access, management, and exchange. This chapter investigates related efforts to support each of these functions in resource-constrained environments and demonstrates where existing solutions fail to meet the demands of new and evolving application scenarios. The chapter concludes with a summary of the three primary contributions of this dissertation.

3.1 Capability and Content Access

The input and output capabilities of pervasive computing devices are limited, especially when compared to their desktop counterparts. A device such as a digital watch may only have a 1-inch diameter face as compared to a 17-inch desktop monitor. Moreover, a watch certainly cannot accommodate a full sized keyboard or traditional mouse. The impact of this property is that most pervasive devices cannot use traditional paradigms for accessing, navigating, and modifying information. As devices become more commonplace, there is a call to develop alternative mechanisms to support content access in varied environments under varied conditions.

3.1.1 Interfaces for Pervasive Devices

Research on interfaces for pervasive devices ranges from creating the low level technology to recognize user input, to performing user studies and analyzing user behavior. In this thesis, we narrow the playing field by specifically focusing on providing system-level support for accessing, navigating, or modifying content on pervasive computing devices. The question at hand is: how can an application programmer ensure that application data is accessible to the widest audience possible? And, of course, the ideal scenario is to also limit the workload of the application programmer.

In recent years, a number of research projects have investigated how to build user interfaces and/or applications such that they are more accessible on devices with diverse capabilities. One of the most promising models that has emerged is that of *single authoring* [39]. Similar to the Model/View/Controller paradigm [32], the idea behind single authoring is that content should be authored independent of its presentation. Similar to Java's *write once, run anywhere* motto, the single authoring strategy could be considered *write once, view anywhere*. There are numerous benefits to such a strategy, not the least of which is that the application programmer or content provider does not have to reinvent the wheel, creating new interface components every time a new device is introduced on the market. However, the strategy itself has yet to be fully realized.

A step in the right direction has been the development and subsequent push of the eXtensible Markup Language (XML). XML, often touted as the next step for the World Wide Web, is an ideal solution for separating content from its presentation. It provides a semantically-enhanced, generic model for representing information. The information can then be presented in a variety of formats, using the added semantics inherent in the representation.

A number of research projects have taken advantage of XML's benefits with respect to interface adaptability [2, 27, 38, 41, 43]. The common theme that underlies these projects is to represent a generic user interface using an XML

document. The document contains basic, device independent information; for example *input values will be in the range 1-10.* This document can then be transcoded, using XSL or similar technology, into the appropriate format for the end-user device (such as HTML for desktops or WML for cell phones). However, current solutions typically require that the application programmer either write the XML document, or the original user interface (which can then be automatically translated into the document [38]). Also, the application programmer is responsible for creating the mapping (e.g., XSL) between the generic XML document and the resulting device interface.

The previous set of projects use XML as a tool to represent the user interface, but not necessarily the content itself. From a slightly different perspective, approaches like JAXB, Castor, and XOBE [31] attempt to automatically build application components from XML data. The assumption is that the XML document contains the underlying information, for example web content or an eBook. An XML schema that specifies the structure of the document is fed into the architecture and the result is a set of customized application components (e.g., JavaBeans), to manipulate and access the information. These approaches are typically useful for a developer who wants to integrate the application components into a larger system. However, they do not address the creation of interface components specifically geared toward providing the user with access to information.

3.1.2 Speech-based Interfaces

The previous section primarily discusses how standard interfaces can be adapted to work on devices with varying capabilities. The bulk of the work discussed attempts to adapt a full-sized visual interface to be a more limited visual interface on a small device. However, an alternative approach is to use a different input and/or output modality all together. In fact, speech is often considered an important component of next-generation interfaces [70].

For a number of years, researchers have been investigating speech-based in-

terfaces. Again, the range of research in this space is quite broad and encompasses everything from improving speech recognition software to evaluating user experience. However, one of the primary challenges that speech-based interfaces pose is the challenge of navigation. Especially in the era of the World Wide Web, access is rarely serial. For example, a user who accesses a newspaper using a traditional web browser is likely to jump from article to article, selecting items of interest based on cues like font size and page layout. Typical speech-based interfaces do not provide an equivalent navigational mechanism. In the typical case, the *listener* of a newspaper would have to listen to every article from beginning to end, in a sequential manner.

There have been a number of efforts that have addressed this challenge. Aster [51] proposes that by using the semi-structured nature of LaTex [37], a user can "listen to" information in a more interactive manner. The fundamental idea of Aster is to use the structure inherent in a LaTex document to provide a listener more freedom to navigate, and hear customized renderings of the content. For example, the structural cues in a LaTex document can be used to generate an aural summary of the document. This provides the listener with the equivalent ability to "scan" the entire document and select a portion of interest. Unfortunately, the structure of LaTex is static. A developer who wanted to use Aster would have limited flexibility in terms of the functionality an application could support. Moreover, it is unlikely that a developer would choose LaTex as the format for content access on a small device.

Other efforts have focused on navigation and document structure and representation in a more general sense. DAHNI [42] focuses largely on navigation *between* documents in a non-visual hypermedia system. The Hyperspeech system [7] looks at presenting "speech as data" by focusing on providing the ability to navigate through recorded speech. Other types of systems have been constructed that look at how to use sound, speech or non-speech, to represent the structure of a document. Brewster [11] develops a set of guidelines for integrating non-speech audio into user interfaces. The work presents a

comprehensive look at the components of sound and how each can be used to convey information. Barrass presents a similar system [9] that looks at how to design sounds to represent the types of activities performed in an application. Portigal [50] looks specifically at how sound can convey structure and location within a document. Finally, Wilson [76] examines the area of data sonification by examining what it means to represent data with sound. While these efforts certainly underscore the challenges of navigating using speech as an input and/or output mechanism and representing information using speech and sound, they do not directly address the challenge of building interfaces for pervasive devices.

A more related area is the investigation of using speech for web browsing and navigation. There is a large body of work that has addressed the question of how to make the current World Wide Web accessible through voice-based interfaces. Most of the work done has focused around how to make the web accessible to users with print disabilities. However, many of the techniques proposed may be applicable to users of pervasive devices as well.

James [29, 30] looks at how to make the web more accessible by integrating sound components like voice and pitch. Her work focuses on designing experiments to determine how a user would best react to different aural representations. It examines the parameters which are involved with designing an aural interface. Other work on making the web accessible has taken a similar approach, looking specifically at HTML and determining, for example, how one should best represent an <H1> tag. Krell and Cubranic [35], Zajicek, Powell, and Reeves [82], Zajicek and Powell [81], Oogane and Asakawa [46], and Albers [4] all look specifically at HTML. Unfortunately, all of these approaches are very specific, and directly targeted toward HTML content. Though some of the proposed techniques may be appropriate for the scenario we target, they have yet to be applied there.

A final body of work that deserves mention is the family of XML-based languages that are being developed as solutions for voice-enabling the web.

Solutions like SpeechML, VoiceXML, JSML, TalkML, and VoxML all provide a language that can be used by a content provider to specify how a user can interact with the system to retrieve information. For example, the document might specify which commands the user can use, or the rate and volume with which a piece of text should be rendered. However, the model is that a content provider must author the VoiceXML, SpeechML, or VoxML document. This makes these solutions limited with respect to their ability to be generalized.

3.1.3 Summary

As pervasive devices with limited display capabilities become more commonplace, there is a call to support diverse input and output access mechanisms. Most current solutions are either too specific, or require a great deal of effort on the part of the content provider. Few efforts have tried to solve these problems, and many that have focus on the automatic creation of generalized application components rather than interface components. The goal of this work is to provide a generalized solution to automatically create interface components to support diverse content access mechanisms. Moreover, the solution must support the generation of components that address the challenges inherent to the given modality, such as speech.

3.2 Capability and Content Management

The fundamental idea behind pervasive computing is that small devices will be ubiquitously deployed throughout the environment. Those devices may range from minute sensors to laptop computers, and may be used for a variety of applications. One class of applications is that of personal computing. The predicted scenario depicts a single user who carries a number of computing devices, from a laptop computer to a digital jacket. These devices can be used to accomplished advanced, personal tasks on behalf of the user. Personal tasks may range from address book maintenance to sending and receiving of email

to code compilation.

Unfortunately, the variety of tasks that a device can accomplish will be limited by the resource availability of the device. Any pervasive device is likely to have limited resources in one respect or another. Bandwidth, processing power, disk space, and battery lifetime are all concerns for the next-generation of computing devices. However, recall that a user may have an entire collection of devices available at any given time. Given this property, it would be useful to be able to aggregate device resources such that devices can be used in concert to overcome resource constraints.

There are a number of challenges associated with coordinating the aggregation of devices and device resources. Examples include the challenge of assigning tasks to the most appropriate device(s) and the challenge of making the coordination appear seamless to the user. In this work, we focus on the challenge of content management. A user will have a subset of his/her personal information stored on each device in his/her collection. For example, a user might store email on a laptop computer, MP3s on a PDA device, and an address book on a cell phone. In order to be able to use the collection of devices together (e.g., to make a phone call using a number that was jotted on a PDA), the *right* content should be made available on the *right* device, and at the *right* time.

Content or data management is not a new idea. Traditional data management schemes address concerns such as scalability, high availability, and concurrency control. However, the new dimension with respect to personal computing in the pervasive environment is resource availability. The fundamental idea is to ensure that the *right* content is available on the device with the *right resources*. This idea of resource-aware content management is the synthesis of many existing, somewhat disparate bodies of work.

3.2.1 Data Prefetching

Data prefetching, in many respects, addresses data management with respect to the dimension of mobility. The idea is that a mobile device, such as a laptop computer, should *prefetch* information while it is able to connect to an infrastructure-based network, or another device. The general motivation is that, in mobile environments where hosts are frequently disconnected, it is important to ensure that, before disconnection, the mobile host downloads any necessary information to continue working in disconnected mode. A number of projects have developed schemes for data prefetching. Coda [66] focuses on caching portions of a larger file system on a mobile host (e.g., laptop) so that the host may continue to have access to relevant data even if not connected to the file system. The concept of *info-stations* has been explored by a number of projects. In the Map-on-the-Move project [79], Ye et al. propose a prefetching strategy to be used in an environment where a user briefly passes through an area of high bandwidth connectivity. Finally, the 7DS project [49] explores the concept of prefetching from a peer-to-peer perspective. It supposes an environment where small, mobile peer devices such as PDAs come in contact for short periods of time, such as on subway trains. While connected, the devices can probe one another to exchange information. One application the work focuses on is exchanging news stories with the goal of obtaining an entire newspaper before departing from a subway train.

These prefetching strategies are primarily concerned with the dimension of mobility. Certainly, mobility is a unique and important property of the pervasive environment. However, these strategies do not address the equally important property of constrained resources. Additionally, the prefetching scenarios typically assume that data is prefetched to a single device. Even today, a single user often carries an entire collection of devices. Our hypothesis is that by taking advantage of the collection's resources, services can be provided more effectively.

3.2.2 Device Cooperation

The idea of aggregating devices in general is not unique to our work. First, a number of projects address the question of how to aggregate a user's personal devices to ease the burden on the user. The MPA project [61] uses an infrastructure that tracks users and chooses the *best* device on which to contact them. Roma [73] aggregates data stored across a collection of devices by storing metadata about all data on a single, portable device. These projects are primarily concerned with device cooperation from a user-centric perspective, but do not address the challenge from a resource-centric perspective.

There are also a collection of projects that take a more resource-centric view of the problem. More specifically, the MOPED project [33] addresses device aggregation from the network layer by using well-connected gateways to communicate on behalf of other devices. While MOPED primarily seeks to integrate the devices belonging to a single user, a similar project has looked at aggregating connectivity across a larger collection of devices [48]. Papadopouli and Schulzrine propose that, in a large-scale environment, some users may have access to connectivity, while others do not. Those that do may act as gateways for those users that would not otherwise have access.

These projects primarily address the resource of "connectivity". However, pervasive devices are constrained with respect to a variety of other resources. In fact, battery lifetime tends to be the resource of most concern. While resources like processing power and bandwidth are improving at a very rapid pace, battery lifetime lags behind. Pervasive devices such as laptop computers may rival the processing, bandwidth, and storage capabilities of desktop computers from only a few years ago. However, battery technology has not improved to meet the demands of the newest computing devices and their users.

3.2.3 Power Conservation

To overcome the limitations of constrained battery lifetimes, a number of research projects investigate power conservation schemes. The idea behind power conservation is to reduce the amount of energy a particular device consumes in the process of normal operation. The Odyssey project [19] focuses on application-level power conservation. In Odyssey, the application receives a callback when the energy supply on the device gets low. The application can then take application-specific action to reduce the amount of energy it consumes. The MillyWatt project [16] advocates a similar idea proposing that power should be managed as a high-level resource. Alternatively, work at UCLA [64] has looked at reducing energy consumption on small devices by offloading computation onto more powerful servers. These servers likely exist as part of a well-connected infrastructure in the environment where the mobile user is currently located. Finally, a number of projects have looked at reducing the power consumption of the network interface on small devices [34, 71]. These schemes work by turning the network interface card off when it is not in use. The tradeoff in this case is managing and accounting for the messages that are sent during the time the network interface is off.

Each of these schemes has proven to reduce the energy consumption on individual small devices. However, even with the most conservative of schemes, a device is still in danger of running out of power. Fortunately, if a user carries not one, but a collection of devices, it would seem logical that one or more of the alternate devices could be used to accomplish tasks when one devices runs out of battery.

A collection of work has also looked at the idea of "aggregating" power by managing data in a power-aware way. Both Directed Diffusion [28] and SPIN [25] focus specifically on the application of sensor networks. In sensor networks, small, power-constrained sensor devices are distributed throughout an environment. As the devices sense and gather data, the data must be propagated throughout the network in a resource-efficient manner. This is

accomplished by building efficient paths throughout the network in the case of Directed Diffusion, and by exchanging high level metadata to determine interest, and sending only the requested data in the case of SPIN.

Unfortunately, the sensor network application has a number of unique properties that do not apply generally to pervasive computing. First, in a sensor network, data is continuously sensed and streamed. In a personal network of devices, content is likely to be comparatively static. That is to say that only a few files may be modified over the course of a few hours. Since updates are not continuous, a batched model is more likely to apply. Additionally, the number of devices in a sensor network is quite large. In a small, personal network, additional optimizations can be considered.

3.2.4 Summary

Intelligent content management is essential to enable cooperation between devices. Such cooperation provides a number of advantages including more effective use of the limited resources available on individual pervasive computing devices. However, the pervasive environment introduces a new dimension into a content management scheme. In order to more effectively use device resources, those resources must be considered when making content management decisions. The idea of resource-aware content management is a synthesis of a number of related areas. Work on data prefetching is concerned with developing a data management strategy, but does not consider the dimension of constrained resources or the benefit of the availability of multiple devices. Work on device cooperation addresses the aggregation of devices, but considers only connectivity as a device resource. Finally, work on power conservation focuses specifically on the resource of power, the primary constrained resource in the pervasive environment, but does not address the cooperation of personal devices. The goal of this work is to synthesize these areas to enable cooperation between devices by being aware of available resources and managing data accordingly. The resource of primary concern is power since it is, by far, the

most constrained resource in the pervasive environment.

3.3 Capability and Content Exchange

The goal of content exchange is to allow end hosts to efficiently share information. In the web model, content exchange happens only in one direction. A participant acts only as a producer or a consumer of information, but not both. Therefore, instead of supporting all of the functionality of content exchange, the web model supports a more limited subfunction – content distribution. However, as next-generation computing environments emerge, there is a call to support the full functionality of content exchange. Envisioned uses of pervasive devices include scientists in the field collecting and sharing data and iPod users roaming about sharing media files. In these kinds of scenarios, all users are on a level playing field and information exchange is likely to be two-way.

Over the past few years, there has been a great deal of interest in supporting the full functionality of content exchange as P2P file sharing has become popular [40, 47]. The P2P computing model has promised to support a variety of applications including global computation [44, 45] and collaborative computing. However, the most popular P2P application thus far has been exchange of MP3s. Due in part to the popularity of the application, quite a few research projects have started to address the question of how to support content exchange in the P2P environment. Research in this space has focused on three main challenges:

Peer Discovery and Group Management. In order to share information, peers need to find out about other peers in the network. The dynamic, ad hoc nature of peer groups makes it difficult to implement peer discovery and group management algorithms. Centralized solutions largely defeat the purpose of a peer network and can be too restrictive if a centralized infrastructure is not available. On the other hand, distributed solutions can be resource inefficient because they require a great deal of overhead in terms of state kept about

other peers and messaging required to maintain that state.

Data Location. To support content exchange, a system must enable users to search for and locate content of interest. The distributed nature of peer networks makes finding information of interest a difficult problem. The solution at one end of the spectrum is to have centralized index or catalogue of available content. However, centralizing the task of data location defeats the purpose of a P2P solution and may not be possible if no centralized infrastructure exists. At the other end of the spectrum, a fully replicated index could be maintained at each peer. However, this solution wastes resources at each peer and it would be difficult, if not impossible, to maintain consistency of a fully replicated index in such a dynamic environment.

Reliable and Efficient Content Delivery. Once a peer locates a piece of information of interest, that peer requests that the storing peer deliver the file across the network. However, end-user peers are inherently resource constrained. Especially when compared to centrally administered servers, end-user devices (e.g., desktop computers) are restricted with respect to bandwidth, disk space, processing power, as well as up-time since peers cannot be relied upon to remain connected for any specific length of time. This limitation makes reliable content delivery more challenging in the P2P environment. New and innovative schemes must be employed to provide fast downloads and avoid overloading the resources of peers that store *hot* items.

In a typical P2P environment composed of desktop computers, these functions are particularly difficult to support. Hosts can come and go at will, and have constrained resources with respect to their server-grade counterparts. As we move toward the pervasive computing environment, hosts will be even more constrained with respect to resources such as bandwidth, processing power, and disk space. Schemes to support peer discovery, data location, and data delivery must be efficient in order to be viable solutions to enable content exchange in pervasive computing environments. The remainder of this section

looks at current research in these areas and identifies where current solutions for the P2P environment become insufficient for the pervasive environment.

3.3.1 Peer Discovery and Group Management

Peer discovery and group management algorithms used in a traditional P2P system can be implemented using a centralized solution, a distributed solution, or a hybrid solution. Centralized solutions such as those used in Napster, Magi, and Groove are most efficient because peers need not keep state about other peers. Moreover, peers can locate each other with a single request to the centralized directory. The problem with this approach is that it requires a centralized infrastructure. Such an infrastructure may not always be available, as is the case in the pervasive environment, or may introduce a central point of failure which minimizes the benefit of a P2P system.

Distributed solutions such as Gnutella, FreeNet [14], Chord [72], CAN [52], Tapestry [83], and Pastry [62] generally rely on using a well-known peer to discover the rest of the peer group. However, the group management protocols employed by these solutions are distributed. In Gnutella and FreeNet, a peer keeps track of a constant number of other peers. This is efficient in terms of the state kept at each peer. The problem with the approach is that searching the peer network may be slow.

Chord, CAN, Tapestry, and Pastry represent the second-generation of P2P discovery and group management algorithms. In each of these algorithms, the network is organized such that peers keep track of a logarithmic number of other peers (with respect to the number of peers in the network). When searching, the protocols can guarantee, or guarantee with high probability, that the desired item can be located in a logarithmic number of peer hops.

There has also been some exploration into the tradeoffs between centralized and distributed solutions A group of peers, particularly those using mobile devices, may have intermittent access to a centralized infrastructure. Therefore, it may be beneficial to have the ability to tradeoff between centralized and dis-

tributed solutions based on the currently available infrastructure. However, research in this space is still somewhat immature.

Peer discovery and group management algorithms have, so far, been a primary focus of research on P2P content exchange. The solutions proposed are quite promising and could likely be applied to the pervasive computing environment as well. Therefore, in this work, we focus our attention on the following two challenges of content exchange.

3.3.2 Data Location

Most of the work on supporting data location in peer networks has focused on on-demand searches for information. Systems like Gnutella and Napster, as well as CFS [15], OceanStore [36], and PAST [63], systems built on top of Chord, Tapestry, and Pastry respectively, allow the user to search for a particular document. The user must know the name of the document prior to making the request. When the request is made, the search message is sent to the appropriate peer (or centralized entity in the case of Napster). While many of these systems claim to support file system-like functionality, the infrastructures do not support file system-like content location. Providing that kind of support would require the application to keep track of metadata about each user's files. Even so, this facility would not support exchange of content between users.

Most of these protocols can be resource-inefficient. Gnutella is particularly inefficient since each search message is potentially sent to every peer in the network. Many of the other protocols can guarantee logarithmic bounds. However, even a logarithmic number of search messages may be too consumptive of network-wide resources in a large peer network. Moreover, a user may not always know which item he/she wants to download. This could lead to additional searching on the part of the user as well as a poor user experience.

3.3.3 Reliable and Efficient Content Delivery

In the P2P space, techniques for making content distribution more reliable and efficient have relied on replicating data within the network. Most deployed systems such as Napster and Gnutella rely on the assumption that data are inherently replicated throughout the network. First, the user selects the best peer from which to download content. If the download request fails, generally because the other peer is not reachable, the user must try a different peer.

This model is not always sufficient. It begins to break down when *hot* data is stored on only a small number of peers. Especially if the peers are resource-constrained, they may not be able to support multiple simultaneous requests from the remainder of the network. This problem is further exaggerated by the fact that peer networks are often composed primarily of *freeriders* [3, 65], peers that are only part of the network long enough to retrieve content from other peers.

Solutions to increase the efficiency of content delivery have largely targeted a streaming model. The basic idea is that content can be more efficiently delivered if the task of streaming media is distributed between multiple peers as opposed to just one source. The chaining approach [68] pipelines the data stream through a chain of receiving clients to reduce the burden on the server. Xu et al. [78] have developed strategies to choose the best server(s), and to increase the capacity of the entire system as quickly as possible. Jungle Monkey [26] focuses on building the tree structure to support end-host connectivity. There has also been work focused on streaming in a strictly mobile environment [22]. However, all of these solutions address very specific problems. In fact, the goal is to provide the underlying communication mechanisms to support more efficient content delivery. The remaining challenge is to develop an integrated solution that will employ one or more of these communication mechanisms to support content exchange in a content-driven application.

3.3.4 Summary

Efficient support for content exchange is essential to enable many of the envisioned applications of the pervasive computing environment. Unfortunately, the resource-constrained nature of the environment makes locating and delivering content particularly challenging. Many research efforts have addressed the primary challenges of content exchange with respect to traditional P2P environments of wired, desktop computers. Some of these solutions may be usable in the pervasive environment as well, but an integrated solution remains to be realized. The goal of this work is to take an incremental step in that direction by proposing a more efficient searching and delivery scheme. Rather than reinvent the wheel, the goal of this work is to integrate existing concepts and evaluate them in the context of a larger, content exchange solution.

3.4 Contributions

There are three main contributions of this dissertation:

1. Chapter 4 presents a novel method for automatically generating interface components from a defined document schema. The **MakerFactory** architecture employs the method by supporting the ability to analyze an XML schema and automatically produce the Java code for interface components that allow access, navigation, and modification of documents conforming to the schema. **Audio XmL (AXL)** validates the feasibility of the architecture by demonstrating that usable, speech-based interfaces can be automatically generated by analyzing an XML DTD. This method provides a user and developer-friendly way to support access to information using devices that have varying, and possibly limited user interface capabilities. [54, 59, 60]

2. Chapter 5 presents a novel method for integrating a new dimension, resource availability, into a content management scheme. The architec-

ture presented monitors resource availability across a user's collection of personal devices and enables resource-aware management of the user's data files. The experimental results validate the usefulness of such an architecture by comparing techniques for **power-aware content management** and demonstrating that, in many scenarios, these techniques improve the aggregate device usability. A resource-aware content management scheme enables device cooperation and can ultimately enable users to accomplish more advanced tasks on devices which have inherently constrained resources. [57]

3. Chapter 6 presents a novel architecture to make content exchange more efficient by integrating existing communication and content delivery solutions to provide additional peer network services and improve peer network performance. The **Pixie** architecture provides push-based data location and uses batching and one-to-many delivery to service download requests. Evaluation of the architecture demonstrates that these techniques can reduce resource usage across the network as well as provide better service to the end users. The *Pixie* architecture is an incremental step toward supporting large-scale content exchange for applications using pervasive devices which have constrained resources. [56, 58]

Chapter 4

Enabling Content Access Via Non-Traditional Displays

4.1 Motivation

The WIMP (windows, icons, menus, pointing devices) model is no longer sufficient for providing access to content. As pervasive computing technology explodes, anyone anywhere may have access to the power of a processor and be connected to the Internet. With the explosive growth of portable devices such as Palm Pilots and cellular phones, there is a growing demand for technology that will allow users to interact with their computers in non-traditional ways. Moreover, historically the WIMP model has posed problems for print disabled users [21]. The graphical representation of data on the web and desktop alike prevents many users from accessing the real content underlying the icons and buttons.

The underlying goal of this work is to provide access to information using the most appropriate input and output mode(s) in any given scenario. Today, this goal is typically met by having content providers provide content in different formats. For example, a website might be published in standard HTML as well as in WML for those users who want to access the site using a cell phone. Unfortunately, this translates into more work for content

providers and is unlikely to meet the demands of all possible users. In an ideal scenario, the content provider publishes one, presentation-independent representation of the content. Then, the user can employ tools that present device and user-specific access to the same underlying information.

This work presents an architecture for automatically developing tools that provide customized, multi-modal access to semi-structured content. The eXtensible Markup Language (XML) is emerging as a standard that provides a common format for storing and communicating data. It allows content providers to separate underlying data from its presentation. Our goal is to promote the use of XML, and devise a method that will provide the means for a user to interact with XML content using the input and output modes that are most appropriate for that user. Moreover, XML provides the means to specify a schema; a document that provides metadata about the contents of any document conforming to that schema. By using the schema definition in developing access tools, we can automatically generate and provide more customized tools to support content access.

The MakerFactory is an architecture that supports the generation and instantiation of multi-modal interface components for XML documents. The architecture provides a framework in which multiple interface components can interact and simultaneously provide access to the same content. However, in order to truly demonstrate the ability to automatically build customized interface components based on XML schemas, specific interface components must be developed.

Recall that the focus of this work is on developing tools to enable content access in next-generation environments made up of many small, pervasive computing devices. While most of these devices will not have the resources to provide traditional access to information (e.g., using a mouse and monitor), researchers predict, and current trends indicate, that they will support speech-based input and output. However, there are a number of challenges to developing speech-based interface components.

First, audio is serial. By nature, a listener must assume a passive role where a viewer takes a more active approach [51]. For example, a screen reader that trivially reads a flat piece of text does not give a listener abilities afforded a viewer such as freedom to navigate. A listener cannot scan the information, move back or forward, or infer an underlying structure based on attributes such as font size or page layout. In order to provide a listener with as much control as a viewer, an aural interface must provide a mechanism to allow users to scan information and actively select portions of the document to be read.

Further, it is difficult at best to represent graphical UIs using speech [10]. The current solution employed by most screen readers used to access web content is to rely on "alt" tags to provide information about the content of the images, although a general solution does not exist. However, by separating the content of the data from the presentation of that data, an interface can display the underlying information in the most appropriate manner. Information need not be represented *only* using graphics. The concept of multiple representations of the same data is also useful when an environment must be multi-modal as could be the case with a visually impaired student and a sighted teacher.

Finally, it is imperative to avoid the cumbersome task of generating a specialized interface for different types of data. Solutions like VoiceXML are useful, but require manual intervention when new data need to be communicated. An all-purpose solution should provide an automatic way of generating a data to speech conversion.

Audio XmL (AXL) is a MakerFactory component that can automatically generate customized, speech-based interfaces for XML content. AXL demonstrates the feasibility of the MakerFactory abstraction, and overcomes many of the challenges that must be addressed in order to build usable, speech-based interfaces. This chapter explores the design and implementation of the MakerFactory, and the design, implementation, and evaluation of AXL, the speech-based MakerFactory component.

4.1.1 The eXtensible Markup Language

The eXtensible Markup Language (XML) is gaining popularity as a model for data representation. XML is a standard way to separate underlying data (i.e. the chapters of a textbook or the content of a web page) from the representation of the data (i.e. fancy graphics or just plain text). XML allows a user to define the structure of a document and hence add semantic information to the data. By providing the user that freedom, the user can store the data in such a way that it can be later retrieved in a manner convenient to the user's current set of circumstances.

Like its predecessor, the Standard Generalized Markup Language (SGML), XML uses *tags* to add a semantic, hierarchical structure to the data itself. A *start-tag* marks the beginning of an XML *element*. A start-tag consist of a left angle bracket (<) followed by a *tag name* that describes the content of the element, followed by a right angle bracket (>). The start tag is followed by the content of the element itself. An *end-tag* marks the end of the element. The end-tag looks similar to the start-tag and should have the same *tag name*, however a forward slash (/) follows the left angle bracket. The following example shows a "Name" element with two element children, "First" and "Last".

```
<Name>
  <First> John </First>
  <Last> Doe </Last>
</Name>
```

The semantic structure implies that the data "John" is a *firstname* while the data "Doe" is a *lastname*. However, the same data might have a very different meaning if surrounded by different semantic tags.

The third type of tag is the *empty* tag. An empty tag is used if there is no content for the element and is equivalent to a start-tag immediately followed by an end-tag. An empty tag looks much like a start-tag, except that the right angle bracket is preceded by a forward slash (/). An empty tag can be useful if an element contains *attributes*. An attribute is a *<name, value> pair* where the value is the data itself and the name is a semantic description of the value.

The data from the previous example can also be represented using attributes and an empty tag. The following example illustrates:

```
<Name First=''John'' Last=''Doe''/>
```

In this case, "Name" is an empty tag even though it contains attributes. An empty tag can also exist without attributes. For example, between the <First> and <Last> tags of the first example, the empty tag <Middle/> would indicate that a middle name could be present, but is not present in the given case.

A benefit of XML is that it allows the user to define the hierarchical structure that surrounds the underlying data. A Document Type Definition (DTD) allows the user to specify the structural template of the elements and attributes of an XML document. While XML Schema is another emerging schema specification standard, this thesis focuses on DTD schema specification. A DTD for the first XML example given would look like the following:

```
<!ELEMENT Name (First, Middle?, Last)>
<!ELEMENT First (#PCDATA)>
<!ELEMENT Middle (#PCDATA)>
<!ELEMENT Last (#PCDATA)>
```

DTDs use a regular expression-style language to indicate which elements are required, which are optional, and which can occur in a list. Furthermore, a DTD specifies the order in which elements must occur and may also indicate that an element may contain text or *parsed character data* by using the keyword PCDATA.

The latter example would have a DTD similar to the following:

```
<!ELEMENT Name EMPTY>
<!ATTLIST Name  First CDATA #REQUIRED
                Middle CDATA #IMPLIED
                Last CDATA #REQUIRED>
```

In this case, the attributes are of type CDATA. The *First* and *Last* attributes are required while the *Middle* attribute is optional.

XML has two parsing models. The first is the Simple API for XML (SAX). SAX is an event-driven model that parses a stream and notifies the application when a parsing event occurs. SAX parsing avoids building an entire XML structure in memory and is useful when parsing large documents. The alternate model is the Document Object Model (DOM). DOM parses the entire XML document and builds a tree structure in memory. Moreover, DOM provides an API for navigation within the parsed XML tree. This work employs the DOM model of XML tree navigation.

XML can be used in a slew of different application scenarios. XML not only represents the future of the World Wide Web, it is a far reaching solution that can be used to represent any data kept in electronic form. Electronic books, electronic commerce, and distance learning all promise to make use of XML.

4.2 Design of a Tool to Support Automatic Generation of XML-based Interfaces

One of the largest benefits of using XML for applications such as web content representation, electronic commerce, and distance learning is that XML separates content from its presentation in a semantically-enhanced, semi-structured way. An XML document provides basic information and semantics to allow a variety of presentations, or uses of that information. The goal of this work is to leverage that property. By representing content in XML, we can exploit the inherent structure to provide customizable, intuitive interfaces that allow navigation, rendering, and modification of the document itself. XML not only gives us the benefit of a structural data representation, it also provides a schema definition mechanism. By exploiting the schema definition, we can *automatically* create a customized interface for each family of documents conforming to a given schema. The functionality of the interface should include linking, navigation, creation, and modification of XML documents. The Mak-

erFactory architecture supports this goal. It is interactive, customizable, and useful not only for the XML reader, but for the XML writer as well.

We have developed a customizable system that provides a user-specified, multi-modal view of any XML document. The MakerFactory allows the user to select a set of rendering components as well as define an optional rule document that further specifies the rendering. Additionally, the MakerFactory defines a link traversal mechanism. Therefore, a user can navigate not only within the document itself, but may navigate between XML documents using our Renderer. This system eliminates many of the constraints imposed by current browsing systems. First, the user is not constrained to a visual representation of the data. In fact, the system eliminates the requirement of a monitor. Moreover, the system eliminates the requirement of keyboard/mouse input. A further benefit of the system is that we take advantage of the XML schema model. Given a published schema, the MakerFactory generates a renderer that is unique for the given schema.

The MakerFactory is a general architecture that can support a variety of methods to access information. To further justify the architecture, in this work we develop an auditory-based MakerFactory component. We call this component Audio XmL (AXL). AXL aims to provide a customizable user interface that uses speech as both its input and output mode. We have found that the semi-structured nature of XML helps to overcome many of the inherent challenges in the aural environment. Using the XML structure, users can more easily navigate a given document and *actively* choose which piece of information should be read by the rendering system. Additionally, by analyzing the XML schema of a potential document before hand, we can extract a great deal of information that helps to make the interface as intuitive as possible. AXL also provides a customization language. This facility allows the user to specify the type of interaction he/she wishes to have with the document. This section provides an overview of the design of the MakerFactory architecture and the AXL component.

4.2.1 The MakerFactory

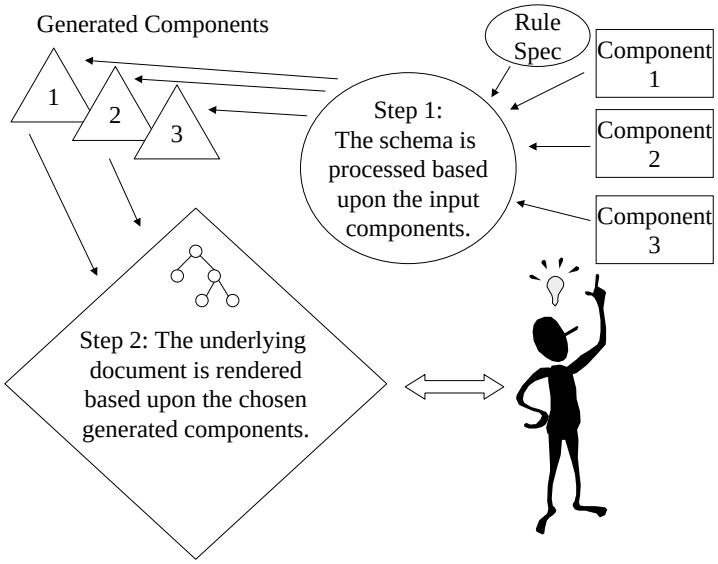

Figure 4.1: An overview of the system design.

The MakerFactory is designed to operate in two phases (see Figure 4.1). The first phase is the code-generation phase. In this phase, the user selects a series of interface generation components from the library provided. To generate a customized interface, the user need only select those components that will be relevant to the run-time rendering scenario. Each component is responsible for *making* a specialized interface for an XML document and hence must implement our Maker interface. In addition, the user may optionally specify a set of customization rules that further refine how the document will be rendered. The result of code generation is a set of Java classes designed to *mediate* communication between the user and the synchronized tree manager. Therefore, the Maker-generated classes should minimally implement our Mediator interface. Since each Mediator is designed to be independent of the others, the user need only select to invoke the Mediators that are relevant to the current scenario and hence not incur the overhead of having to run all Mediators simultaneously.

The second phase is the run-time rendering phase. The MakerFactory provides a Renderer that is responsible for controlling synchronized rendering of the XML tree. Each Mediator acts as an intermediary between the Renderer and the user allowing its own specialized input and output mode. For example, AXL is designed to act as the auditory Mediator allowing the user to navigate the XML document and traverse its links via a speech-based interface.

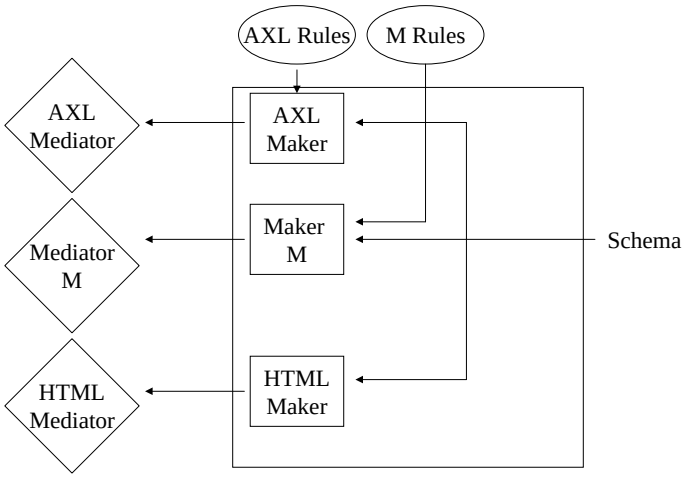

Figure 4.2: The MakerFactory code-generation phase.

Code-Generation. Figure 4.2 shows the architecture of the code-generation system. A given schema is analyzed and the results of the analysis are passed into a series of user-selected Maker classes. Given the schema and any optional customization rules specified by the user, the system produces a set of customized Mediator classes that can be used with the Renderer to interact with any XML document conforming to the given schema. For example, if the user specifies the AXL Maker (a component used to generate auditory input and output methods) and the HTML Maker (a component used to produce an HTML representation of the data), the result should be two independent

Mediator classes. The AXL Mediator will listen for spoken commands from the user and provide spoken output whereas the HTML Mediator may display the XML in HTML format and disallow input from the user. Additionally, if the given schema were a NEWSPAPER with a list of ARTICLES each containing HEADLINE and BODY elements, the AXL Maker might determine that the AXL Mediator should render an ARTICLE by rendering the HEADLINE whereas the HTML Maker may determine that the HTML Mediator will render an ARTICLE by textually displaying both HEADLINE and BODY.

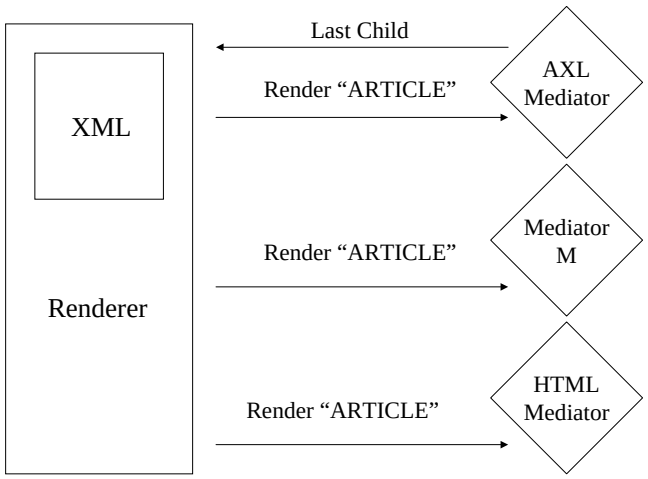

Figure 4.3: The MakerFactory Renderer.

Rendering. The architecture of the Renderer is shown in Figure 4.3. The Renderer controls synchronized access to the XML tree. Each Mediator may receive input from the user in the mode supported by the Mediator. The Mediator interprets the command and issues a corresponding set of commands to the Renderer. The Renderer changes the tree view based upon the command and updates the synchronized view for all other Mediators. Once the Mediator view changes, it may provide output to the user via the Mediator's supported

mode of output.

The Renderer employs the concept of a cursor. At any given point, all of the registered Mediators should be rendering the portion of the tree pointed to by the cursor. When the cursor is moved, the new view of the tree should be rendered. However, it is possible that a Mediator will have to move the cursor more than one time to achieve the desired view. This is because the methods to move the cursor are generally incremental and somewhat limited. For example, moving to the grandparent of a node would require two requests to move the cursor to the current node's parent. To accommodate this situation, the Renderer defines a locking mechanism. Before calling a method that will move the cursor, the given Mediator must acquire the lock. After all movement is complete, the lock is released. When the lock is released, all of the Mediators are notified that the cursor has changed. For example, if a Mediator directs the Renderer to move the current cursor to an ARTICLE node, the Renderer will in turn ask all of the other Mediators to render the given ARTICLE.

4.2.2 AXL

To MakerFactory architecture is quite general and can support a variety of functions. In order to best evaluate its use, we have developed a specific MakerFactory component, AXL. The goal of the AXL component is to demonstrate the feasibility of the MakerFactory goals by producing a usable, speech-based interface for any XML document. The choice to focus on a speech-based component follows from the computing environment that we imagine for the future. In the future, we imagine lots of small computing devices that may be carried (such as PDAs and cellular phones), worn (such as clothing and jewelry), or embedded into the environment (such as sensor devices). Many of these devices may not have 17-inch monitors and keyboards for input and output. However, most are likely to have the ability to accommodate a speaker and microphone. Many researchers predict that speech-based interaction will play a large role in the pervasive computing environment of the future.

The primary goal in the design of AXL was to create a component that would produce a usable, speech-based interface for any XML document. There are many questions we sought to answer in conjunction with this goal. First, we wanted to produce user interfaces that would give the user as much control as possible over the navigation of the document. In the case of a large web page, the user would likely want the ability to choose which portion of the page she wanted to hear. However, we recognized that there are situations where the user may actually wish to be the passive participant while allowing the rendering system to navigate and present the XML document. For example, a user that reads an electronic newspaper in the same order everyday might wish to simply configure the system to read the newspaper in that order without requiring user interaction. Therefore, we wanted our design to allow the user the freedom to choose the level of interaction she wishes to have with the document.

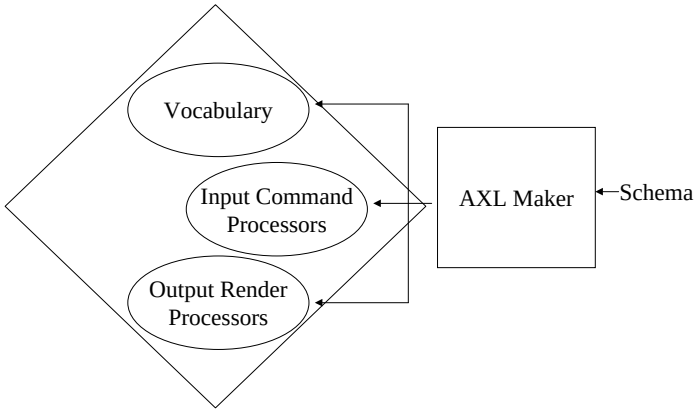

Figure 4.4: The AXL code-generation phase.

Code-Generation. In the code-generation phase, the AXL Maker analyzes the schema and produces a vocabulary for interacting with a document conforming to the given schema. For example, the previously cited NEWSPAPER example might generate a vocabulary {article, body, headline}. Each word in the vocabulary has a semantic meaning associated with traversing a document of the given type. The "headline" command may mean to read the HEADLINE of the current article. Given that vocabulary, the AXL Maker generates a set of classes that process commands within the vocabulary as well as a set of classes that know how to render nodes of the given schema via a speech synthesizing engine as shown in Figure 4.4. The resulting set of classes compose the AXL Mediator.

Rendering. The AXL Mediator operates as shown in Figure 4.5. When the Mediator is asked to render a node, AXL will determine the type of node it is being asked to render and then perform the appropriate operations. The rendering can include everything from a simple rendering of the node's tag

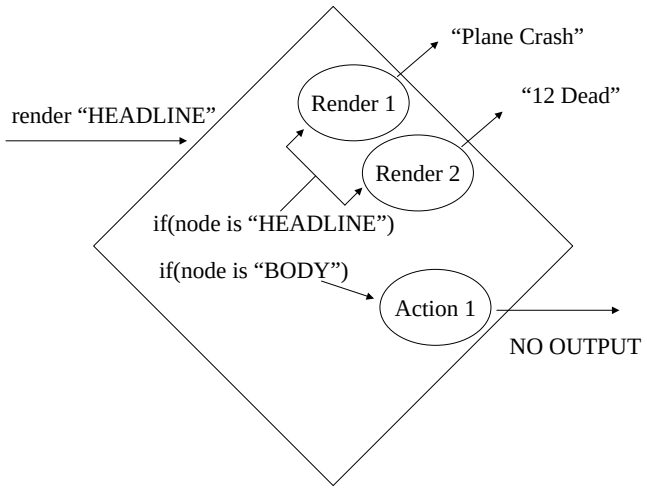

Figure 4.5: The AXL Rendering phase.

name to performing the action specified in a pre-compiled Java class. Perhaps a HEADLINE node has two parts, TITLE and SUBTITLE. The rendering of a HEADLINE may indicate to render first the title and then the subtitle. Correspondingly, perhaps rendering a BODY node simply means to perform some action and provide no output to the user.

4.2.3 Link Traversal Support

Linking is an important feature of XML. Not only is it important for a web scenario, linking is important in a variety of XML applications. For example, an electronic book might provide links to annotations of the material.

At the most basic level, we want to support the kinds of links defined in HTML using the "A" tag and "href" attribute. For XML, the XLink standard is being developed. Not only does XLink provide the mechanism to define HTML style links, it also has the benefit of attaching semantic meaning to the action of link traversal. We investigate how to design our system to support XLink style link traversal.

MakerFactory Linking. In order to support a linking mechanism, the Renderer must provide a mechanism for the Mediator classes to request that a link be followed. We investigate this requirement by looking at the different attributes that may occur within the XLink namespace. Currently, we are looking only at XLink *simple* links.

The XLink standard defines four classes of attributes. Each type of attribute can be specified in an XLink node and helps the parser to determine how to traverse and display the attribute as well as provides metadata describing the function and content of the link itself. The MakerFactory considers how to integrate all of the possible values of those attributes given the XLink definition of their functions.

The *href* location attribute is necessary for determining where to find the given node. The **arc end** attributes are not required at this time for the purposes of the MakerFactory. The **behavior** attributes help the MakerFactory in determining how to load the referenced resource. The resource may be appended to the current tree, replace the current tree possibly causing the system to instantiate a new set of Mediator classes, or may require that a new Renderer be instantiated. Additionally, the Renderer may automatically load the resource, or may wait for a Mediator to explicitly ask to load it. Finally, a Mediator may use the **semantic** attributes in its own rendering of a node to give the user more information about the link.

AXL Linking. It is the job of AXL to appropriately interpret the MakerFactory supported commands and provide the user with a suitable method of accessing those commands via a speech-based interface. Therefore, AXL must allow the user to issue a command that *follows* a link. If the user has reached a node that is identified as a link, the user may ask to follow the link. AXL will subsequently invoke the Renderer method used to perform the traversal. The result may be a new tree which will then be rendered appropriately by all of the Mediators including AXL, or may simply be the current tree with an additional subtree.

Conceptual Classification	**Attribute**	**Function**	**MakerFactory Implications**
Location	href	provides URI of resource	Renderer uses value to load resource
Arc End	from	defines where link originates	NONE
	to	defines where link traverses	NONE
Behavior	show	defines how the resource should be presented to the user	**parsed** - resource is appended as child of current node **replace** - resource replaces current data; new Mediators may be instantiated **new** - requires a new Renderer be instantiated; see future work on *Speech Spaces*
	actuate	defines how link traversal is initiated	**user** - a Mediator must request link traversal **auto** - the Renderer traverses the link upon rendering the XLink node
Semantic	role	defines function of link	may be used to enhance rendering
	title	description of link	may be used to enhance rendering

Table 4.1: The XLink Specification Attributes.

XLink provides the definition of *role* and *title* attributes. These define the link's function and provide a summary of the link. AXL can take advantage of both of these attributes. If the node currently being rendered is a link node, AXL will render the node by synthesizing both semantic attributes for the user. These attributes provide a built-in way for the XML writer to indicate to AXL the best rendering for the node.

Our design supports a concept of data-specific navigation and editing tools. The following sections describe our implementation of the framework, and of an aural component to support speech-based navigation and modification of XML data.

4.3 Implementation of a System for Navigating and Editing XML Data

The MakerFactory architecture is designed to provide an infrastructure for automatically generating interfaces for navigating and modifying XML data. This section examines the implementation decisions of the MakerFactory. Many of the implementation details of the MakerFactory are more clearly illustrated by using the speech component, AXL, as an example.

The entire system has been implemented in Java using the IBM Speech for Java implementation of the Java Speech API. Furthermore, the Speech for Java implementation requires that IBM's ViaVoice be installed and used as the synthesizing and recognizing engine.

4.3.1 The MakerFactory

The general concept of the MakerFactory is to traverse a schema and notify each Maker of the content model of each node in the schema. Based upon the schema and a given rule specification file, each Maker generates an interface for the user and the end result is a completely customized, multi-modal user

interface to any XML document conforming to the given schema.

MakerFactory Configuration Schema. When the MakerFactory is invoked, a configuration file must be specified. The following is the DTD schema that defines the MakerFactory configuration file:

```
<!ELEMENT MakerConfig (Maker*) >
<!ATTLIST MakerConfig
          SchemaProcessor CDATA #REQUIRED
          SchemaLocation CDATA #REQUIRED>

<!ELEMENT Maker EMPTY>
<!ATTLIST Maker
          Classname CDATA #REQUIRED
          Rulefile CDATA #IMPLIED>
```

The MakerConfig element has two attributes. The first attribute defines where the program should find a SchemaProcessor class. The value of this attribute should be a string that indicates the class name of a class that implements the SchemaProcessor interface. The current implementation provides a DTDSchemaProcessor class that supports the DTD schema definition. However, there are multiple ways to define a schema for an XML document. XML Schema is another example. Therefore, to support other methods, any other schema processor conforming to the SchemaProcessor interface may be used. The second attribute of the MakerConfig element is the SchemaLocation. The value of this attribute should be a string that indicates where the actual schema definition can be found e.g. the file location of the DTD. This string should be a valid URL.

The MakerConfig may have a list of Maker elements as its child nodes. Each Maker element indicates a Maker to be used in interface generation. The first attribute of the Maker element is the Classname. The focus of this work is the SpeechMaker or AXL. Other examples might include an HTML Maker that transforms an XML document into an HTML document or a Picture Maker that creates a pictorial view of the XML data by querying an image

repository. The second attribute is the name of a Rulefile. The MakerFactory defines a general schema for the set of rules, however each Maker may refine the MakerFactory rule schema to allow a user to write highly customized rules for rendering the elements in the manner defined by the Maker.

The SchemaProcessor Interface. The SchemaProcessor is defined by the following interface:

```
public interface SchemaProcessor {
    void init(String loc) throws InvalidSchemaException;

    boolean hasMoreDecls();

    SchemaNode getNextDecl();
}
```

The current implementation supports a DTD processor, however any class implementing the interface may be used in place of the DTDSchemaProcessor. The processor acts as an enumerator. It should process the given schema and generate a generic SchemaNode representation of each element declaration. A SchemaNode is the MakerFactory defined structure to hold all of the relevant information given in the schema definition of an element. The MakerFactory will iterate once through the schema and process each node.

Any implementation must support three methods. The first is the **init** method that takes as a parameter the location of the actual schema file. If the schema is found to be invalid, an *InvalidSchemaException* should be thrown. The schema defined in the file should be in the format that the SchemaProcessor is expecting. For example, the DTDSchemaProcessor should be given the location of a ".dtd" file.

The second method that must be supported by a SchemaProcessor is the **hasMoreDecls** method. This method is called to determine if all of the nodes in the schema have been processed. If the processor has iterated through all of the nodes, this method should return false, otherwise it should return true.

The final method that a SchemaProcessor must define is the **getNextDecl** method. This method should return the next unprocessed node from the schema. Each node should be processed once. The first step in processing a node of the schema definition is to create a generic SchemaNode object. This MakerFactory defined object contains all of the relevant information from a given node. By analyzing a SchemaNode, a Maker should be able to derive all necessary information in order to build the interface. This gives rise to the second step in processing a node. Each Maker should extract all of the information it needs from the SchemaNode and use it to create the appropriate components of the interface.

The Maker Interface. Each Maker must conform to the following Maker interface:

```
public interface Maker {
    public void processNode(SchemaNode snode);

    public void addRulefile(RuleFileProcessor rfp);

    public void export();
}
```

A Maker is notified of each node in the schema and should ultimately produce a set of one or more Java classes that will allow navigation of a document conforming to the schema specified.

A Maker must support three methods. The first is the **processNode** method. This method should take a SchemaNode as a parameter and do the processing necessary to create a Mediator. The processing may include using the SchemaNode API to extract information about the XML instance defined in the schema and generating the appropriate Java code.

The second method the Maker must support is the **addRulefile** method. If the configuration document specifies a rule file, the rules defined will be given to the Maker. The Maker should then use the defined rules when building the customized interface. At the very least, a Maker should be able to handle

rules conforming to the rule schema defined for the MakerFactory in general. If a given Maker wishes to extend the general rules, it must also implement a way to analyze those extensions.

The final method that must be supported by a Maker is the **export** method. This method will be called when the entire schema has been traversed. The Maker should do any cleanup and make the results of the schema traversal persistent. Generally, this means that the Maker will create one or more Java files containing code that implements the Mediator interface.

The RenderRule Language. Each Maker in the factory can be given an XML document containing user-specified rendering rules. The default Maker generated rendering might not be sufficient for all user's needs. Therefore, the MakerFactory defines a RenderRule schema that may be extended by any given Maker. Any Maker implementation should provide the necessary code to process such a rule definition. At the most basic level, a RenderRule document must conform to the following DTD:

```
<!ELEMENT RenderRules (ElementRule|AttributeRule)+>
<!ELEMENT ElementRule (ElementRendering)+>
<!ATTLIST ElementRule
          elementid CDATA #REQUIRED>
<!ELEMENT ElementRendering (Render|Action)+>
<!ATTLIST ElementRendering
          cond CDATA #IMPLIED>
<!ELEMENT Render EMPTY>
<!ATTLIST Render
          depth (NONE|NODE_ONLY|CHILDREN_ONLY|
                 ATTRS_ONLY|NODE_CHILDREN|
                 NODE_ATTRS|CHILDREN_NODE|CHILDREN_ATTRS|
                 ATTRS_NODE|ATTRS_CHILDREN|NODE_ATTRS_CHILDREN|
                 NODE_CHILDREN_ATTRS|ATTRS_CHILDREN_NODE|
                 ATTRS_NODE_CHILDREN|CHILDREN_NODE_ATTRS|
                 CHILDREN_ATTRS_NODE|NAME|VALUE|NAME_VALUE|
                 VALUE_NAME) ''NODE_ONLY''
          cond CDATA #IMPLIED>
<!ELEMENT Action EMPTY>
<!ATTLIST Action
          classname CDATA #IMPLIED
          ismodifier (TRUE|FALSE) ''FALSE''>
<!ELEMENT AttributeRule (AttributeRendering)+>
<!ATTLIST AttributeRule
          attribteid CDATA #REQUIRED>
<!ELEMENT AttributeRendering (Render|Action)*>
<!ATTLIST AttributeRendering
          cond CDATA #IMPLIED>
```

The RenderRules element should contain a list of ElementRule and/or AttributeRule children. The ElementRule element is used to specify a rendering for a given element in the input XML document to be rendered and the AttributeRule element is used to specify a rendering for an attribute in the given document. We explain each type of rule in the following paragraph.

The ElementRule should contain a list of ElementRendering elements. Each ElementRendering with a condition that evaluates to true will be invoked. Any ElementRendering child with no condition will be invoked by default. In addition, the ElementRule should have an *elementid* attribute.

This attribute should specify the tag name of the elements that will match this rendering rule.

An ElementRendering element can contain one attribute *cond*. If specified, this attribute indicates the name of a class that implements the UnaryPredicate interface. The UnaryPredicate should take one argument, the node in question, and produce a boolean value. When the parent ElementRule is invoked for an element, the class indicated by the *cond* attribute is instantiated and passed the node currently being rendered as an argument. If the UnaryPredicate returns true, the rendering defined by the children of the ElementRendering element will be used to render the node.

The children of the ElementRendering element define a single rendering. The rendering is defined by a list of Render or Action children. The children are invoked in the order they are defined. A Render child will produce one view of an element and an Action child will invoke a user-defined Java code block. Any Action class must implement the following interface:

```
public interface Action {
  public void notify(Node node);
}
```

Like the Java event model, a render event causes the **notify** method of the Action class to be invoked. The current node to be rendered is passed as an argument. If the action modifies the node it its subtree, the Renderer should be notified of the change following the completion of the **notify** method. This allows a user to render an element, change the content through the use of an action block, and then render it again.

The Render element has two attributes, *depth* and *cond*. The *cond* attribute acts the same as the ElementRendering *cond* attribute. Before a node is rendered it must meet the condition in the UnaryPredicate class. The *depth* attribute indicates the depth to which the node should be rendered. The valid values for this attribute are:

```
NONE
NODE_ONLY
CHILDREN_ONLY
ATTRS_ONLY
NODE_ATTRS
NODE_CHILDREN
CHILDREN_ATTRS
CHILDREN_NODE
ATTRS_CHILDREN
ATTRS_NODE
NODE_ATTRS_CHILDREN
NODE_CHILDREN_ATTRS
CHILDREN_ATTRS_NODE
CHILDREN_NODE_ATTRS
ATTRS_NODE_CHILDREN
ATTRS_CHILDREN_NODE
```

The node, its children, or its attributes may be rendered. In addition, any combination of the three may be specified and in any order. For example, NODE_CHILDREN means to render the node and then its children whereas ATTRS_CHILDREN_NODE means to render the attributes of the node, the children of the node, and then the node itself. Finally, NONE is the option that indicates that the node should have no rendering at all.

The Action element contains two attributes. The first is the *classname* attribute. This should contain the name of a class which implements the Action interface. When invoked, the **notify** method of this class will be passed the node currently being visited. The second attribute is the *ismodifier* attribute. This attribute should be TRUE if the Java action code modifies the node it is passed and FALSE otherwise. If this attribute is TRUE, the Maker that generates the code should generate the appropriate code to replace the global XML in the Renderer itself with the modified local copy. In the future, the rule language will be extended to support Java code embedded within the XML rules.

The ElementRule and AttributeRule elements are very similar. The only difference between the two is that the valid values for the *depth* attribute of an

AttributeRendering node are NODE, NAME, VALUE, NAME_VALUE, and VALUE_NAME. This allows the user to specify whether just the name, just the value, or both properties of an attribute are rendered.

It is up to the programmer to extend the schema for a given Maker. See the AXL discussion in the next section for an example extension to these base rules.

4.3.2 The MakerFactory Renderer

The MakerFactory also provides a Renderer that may be used to navigate and modify an XML document. The Renderer interacts with each Mediator in two ways. First, the Renderer receives directives from the Mediators that indicate that the current cursor should be moved or that the tree should be modified. Second, the Renderer issues directives to the Mediators to update each Mediator's current view of the tree. Like the MakerFactory, the Renderer also defines a configuration file schema that specifies which Mediators to use. The Renderer can interface with any class that implements the Mediator abstract class.

The Renderer reads in an XML document and instantiates a series of Mediators that will listen for changes in the traversal of the document. Each Mediator interfaces with both the Renderer and the user. Based upon the user input, the Mediators notify the Renderer to change the current view of the document or modify its contents. When one Mediator updates the view, all Mediators are notified by the Renderer.

Renderer Configuration Schema. The following is the DTD schema that defines the Renderer configuration file:

```
<!ELEMENT RendererConfig (Mediator*) >
<!ATTLIST RendererConfig
          XMLFile CDATA #REQUIRED >

<!ELEMENT Mediator EMPTY>
<!ATTLIST Mediator
          Classname CDATA #REQUIRED >
```

The RendererConfig element has one attribute. The XMLFile attribute defines the name of the XML file that is to be traversed. This string should be a valid URL of a file that contains a valid XML document. The XML document should conform to the schema used when creating the Mediators defined.

A RendererConfig element contains a list of Mediator elements. It is likely that each Mediator will be an automatically generated MakerFactory Mediator, however, any class extending the Mediator abstract class can be specified. Each Mediator element specifies a *Classname* attribute. The Classname should be a string indicating the name of the Mediator class.

The Renderer Class. The Renderer is a fixed class that exports a series of methods that may be called by a Mediator class. The methods are generally wrappers around the org.w3c.dom.Node interface. The methods exported support navigation through and modification of an XML document. The Renderer implements the concept of a cursor. There are conceptually two cursors. One points to the current element, and the other points to the current attribute of the current element. When the element cursor moves, the attribute cursor will point to the first attribute of the new element. The methods exported by the Renderer can move the cursors, or change the data to which the cursors point. The methods supported by the Render class are defined below:

appendChild(Node newChild)

Behavior - appends the newChild as the last child of the current element

Return Value - *true* if the append was successful

attribute(String attrname)

Behavior - moves the current attribute cursor to the attribute of the current element with the given name

Return Value - *true* if the move was successful

element(String eltname, int n)

Behavior - moves the current element cursor to the nth occurrence of the element "eltname"

Return Value - *true* if the move was successful

firstChild()

Behavior - moves the current element cursor to the first child of the current element

Return Value - *true* if the move was successful

insertBefore(Node newChild)

Behavior - inserts newChild as the previous sibling of the current element node

Return Value - *true* if the insert was successful

lastChild()

Behavior - moves the current element cursor to the last child of the current element

Return Value - *true* if the move was successful

name()

Behavior - gets the tag name of the current element

Return Value - *String* - tag name of the current element node

nextAttr()

Behavior - moves the current attribute cursor to the next attribute of the current element

Return Value - *true* if the move was successful

nextSibling()

Behavior - moves the current element cursor to the next sibling of the current element

Return Value - *true* if the move was successful

parent()

Behavior - moves the current element cursor to the parent of the current element

Return Value - *true* if the move was successful

previousAttr()

Behavior - moves the current attribute cursor to the previous attribute of the current element

Return Value - *true* if the move was successful

previousSibling()

Behavior - moves the current element cursor to the previous sibling of the current element

Return Value - *true* if the move was successful

remove()

Behavior - removes the current element and moves the current element cursor to the parent

Return Value - *true* if the remove was successful

replaceChild(Node newChild)

Behavior - replaces the current element with newChild

Return Value - *true* if the replace was successful

setAttr(String name, String value)

Behavior - sets attribute name of the current element to the given value

Return Value - *true* if the set was successful

setData(Document newDoc)

Behavior - replaces the XML tree with newDoc and renders the root

Return Value - *true* if the set was successful

value()

Behavior - returns a cloned copy of the current element node

Return Value - *Node* - cloned copy of the current element and its children

Because most of the commands are iterative, that is to say that attribute (attributename) and element (elementname, indexOfElement) are the only methods that allow a user to randomly access a node, a Mediator may want to move a cursor multiple hops before settling on the node that should be rendered. For example, to access the grandparent of the current node, two calls must be made to the **parent** method. To accommodate this case, the Renderer implements the concept of a lock. Before the Mediator can call any of the methods available to alter the cursor state, the lock must be acquired. Once the lock is acquired, the Mediator may move the cursor as many times as it likes. Only when the lock is released are the remaining Mediators notified of the new state of the tree. If the lock is released and the cursor has moved, the node that the cursor now points to is queued for update. Each Mediator is notified of the current state of the cursor.

The locking mechanism is also useful for write interaction. Most of the discussion of this thesis focuses around navigation within an XML document. However, modification is also a desired quality of such a system. In order to allow multiple Mediators to participate and possibly modify the XML tree itself, the Renderer allows a Mediator to lock the tree and replace the current node with a new node specified by the Mediator.

The Renderer interfaces with any Mediator class. In most cases, the Mediators will be automatically generated by the MakerFactory. However, it is possible that a Mediator is physically implemented by a developer using the Renderer. All Mediators must conform to the Mediator abstract class definition.

The Mediator Class. The purpose of a Mediator is to provide a mode-specific interpretation of the basic DOM methods supported by the Renderer. For example, the AXL Mediator provides a means to navigate the XML tree

using speech as the input/output mode while a Picture Mediator might provide the facility to drag and drop a pictorial representation of the data.

Any Mediator must implement the **render** method. This method should render the given node in a manner appropriate for the Mediator. The parameter to the method is the node to be rendered. This node is a deep cloned copy of the actual node and all of the subtrees of the node.

Mediators must also implement a **cleanup** method. This method should deallocate all resources appropriately and stop listening for user input.

The **init** method of the Mediator is provided. This method sets the parent Renderer object. This is necessary so that the Mediator can make calls to the parent to traverse the XML document.

4.4 An Auditory Component for XML Navigation and Modification

The MakerFactory implementation provides a framework to support generation of multi-modal interface components. In order to fully understand how the architecture can be used, we have chosen to implement an auditory component called AXL. The goal of AXL is to *automatically* generate audio-based components for navigating and modifying XML data. This section evaluates the implementation choices of AXL. At the lowest level, AXL provides an auditory version of the Document Object Model. Beyond that, it provides a customized interface through schema analysis and user-customization.

Speech DOM. At the most basic level, AXL provides a speech version of the Document Object Model. DOM defines a series of methods that allow access to a parsed XML tree. Our Speech DOM provides a set of spoken commands that perform the same function. At the core level, an XML tree is a collection of org.w3c.dom.Node objects. Therefore, the AXL Speech DOM provides a set of commands conforming to the org.w3c.dom.Node API.

The basic commands available are:

Parent
Next Sibling
Previous Sibling
Next Attribute
Previous Attribute
First Child
Last Child
Name
Value
Remove
Insert Before
Append
New Text
Set Attribute

When each command is spoken, the Mediator simply calls the corresponding method of the Renderer class. The only commands that warrant explanation are *Insert Before*, *Append*, *New Text*, and *Set Attribute*. Each of these commands requires more input than the command itself. When the command *Insert Before* is spoken, the system beeps, indicating that it is waiting for more user input. It waits for the user to speak the tag name of the node to be inserted. When it hears the name it inserts the node and returns to normal operation mode. *Append* is exactly the same, except that the new element is appended as the last child of the current node rather than inserted as the current node's previous sibling. *New Text* and *Set Attribute* both require actual dictation from the user. Therefore, the *New Text* command causes the system to beep indicating that it is waiting for free dictation. It records everything the user speaks until the user issues the command *Complete*. At that point, all of the text is appended as the last child of the current node. Finally, *Set Attribute* is much like the *New Text* command. However, there are two parts to the new node, the name and the value. Therefore, *Set Attribute* puts the system in the dictation mode. The first word spoken in this mode is interpreted as the name of the attribute. Anything spoken before the *Complete* command is issued is interpreted as the value of the attribute. For example, "*Set Attribute* [Title] [A Midsummer Night's Dream] *Complete*" would set the

Command	Behavior
Complete	ends dictation mode
Pause	pauses input; engine only recognizes Resume command
Resume	ends paused state; engine listens for all commands
Command List	lists the set of commands available at the current node
Save	saves the current XML tree
Stop	stops the rendering of the current node
Exit	performs any necessary cleanup and exits the program

Table 4.2: Command Set Used to Direct the AXL Engine.

Title attribute of the current element to be "A Midsummer Night's Dream".

Table 4.2 outlines commands the user may employ to direct the engine itself. Each command tells the engine to perform a function independent of the underlying document. This command set should not result in an update to the view of the XML structure.

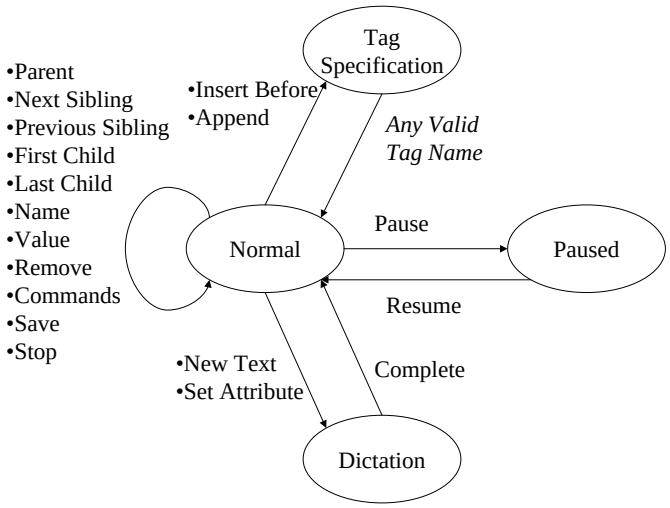

Figure 4.6: The AXL engine state transition diagram.

The AXL Speech DOM engine can be in one of four states (see Figure 4.6). The transition from one state to another is initiated by a spoken command from

the user. To transition from *Normal* to *Paused*, the user speaks the command *Pause*. *Resume* returns the system to normal operation. *Dictation* mode is entered by speaking either the *New Text* or *Set Attribute* command. After the user finishes dictating, *Complete* returns the system to normal operating mode. *Tag Specification* is entered by either the *Insert Before* command or the *Append* command. Once the tag name is spoken, the system automatically returns to the *Normal* state. The remaining commands cause the system to perform some action which may or may not result in output to the user, however the system itself remains in the *Normal* state.

This basic implementation does not require a schema or user-specified rules. In fact, this interface can be created one time and used to navigate any XML document. However, it is not enough to provide a generic, low-level interface to an XML document. We would like to leverage the functionality provided by the Speech DOM, and build a higher-level interface on top of it to give users a more intuitive way to navigate a document. A schema-specific interface not only allows more intuitive navigation, it provides the means to customize the output rendering of each type of node. Ultimately, given an XML document conforming to one schema, we would to interact with it in a way specific to that schema.

4.4.1 A Schema-Driven Interface

Primarily, the DTD tells us what kind of element children and attributes an element is allowed to have. The first task is to use that information to provide a rendering of the node. For example, in addition to rendering the node by speaking the tag name, we can also speak the names of the children and attributes that the node is allowed to have. For example, the rendering of an ARTICLE node might be "An ARTICLE is a HEADLINE and a BODY". This provides the user with a brief look at the general structure at any given node without having to know the structure before hand.

The second task is to use the schema derived information to produce a

more intuitive vocabulary with which we can interact with the document. First, we can use the knowledge of the elements and attributes that exist in the document to develop a vocabulary. Suppose that a user wished to navigate to the BODY element of an ARTICLE. Instead of navigating all of the children using the Speech DOM vocabulary, we allow the user to actually speak the command "BODY". This is more intuitive for the user. In addition, developing such a vocabulary allows us to easily disallow illegal directives. If the user asks for the NEWSPAPER node while at the BODY element, we can immediately give the user feedback indicating that no such node exists without having to actually look for the node in the subtree. This creates a more efficient system.

Even though this design allows us to query using a more high-level language, it still does not provide all of the facilities that one might desire. The user may want to have more control over how each type of node should be rendered. The automatically generated interface may not be sufficient for a user's desired usage of the system. In that case, user input is required.

4.4.2 The ABL Rule Language

There are many reasons why a user might want to specify how they would like a document to be rendered. The complete semantic definition of the structure might not be specifiable in an XML schema. Furthermore, the user may just have a different preference than that which is defined by the default AXL maker. Therefore, AXL defines its own rule language, the Audio Browsing Language (ABL). ABL allows a user to provide input about the rendering of the element and attribute nodes in the XML tree.

ABL Implementation of the MakerFactory Rules. The general MakerFactory architecture provides a high-level rule language that we can use to specify general rules. However, the meaning of those rules can be interpreted differently by different types of Makers. This section examines the ABL interpretation of the MakerFactory rules.

The *cond*, *elementid*, and *attributeid* attributes are straightforward and implemented as they are described in section 4.1.4 of this thesis. Both *elementid* and *attributeid* specify the tag name of the given node that should invoke the defined rule. For example, if "Person" occurs as the value of the *elementid* attribute, then the rendering defined in the rule will be invoked when a Person element is encountered. When the *cond* attribute occurs as a child of an ElementRendering element, the given ElementRendering will only be invoked if the condition is met. If the *cond* attribute is a child of an AttributeRendering element, the same logic applies. When *cond* occurs as a child of a Render element, that specific rendering will only be invoked when the condition is met. Suppose that the content model of an A node were the following:

```
<!ELEMENT A ((B|C)*)>
```

Suppose that the logic behind the rendering were as follows:

```
if A has less than 5 children
  render all B children then render all C children
else
  render all C children then render all B children
```

To implement this logic, there should be two ElementRendering nodes. The first should be invoked if the condition $numChildren(A) < 5$ is met, and the second should be invoked in all other cases. Furthermore, the first rendering element should have two Render children. The first should indicate that CHILDREN_ONLY should be rendered upon condition $tagName = B$ and the second should indicate that CHILDREN_ONLY should be rendered upon condition $tagName = C$. The second ElementRendering node would also have two Render children. The first of its Render children would indicate that CHILDREN_ONLY should be rendered upon condition $tagName = C$ and the second would indicate CHILDREN_ONLY upon condition $tagName = B$.

The ABL implementation of the *depth* attribute also warrants explanation. In the speech interface, all rendering is done recursively. The default implementation specifies that only a node is rendered. However, the user may wish

to change that implementation. It may make sense to render an entire subtree rather than just the node at the root. To illustrate, we use the following XML fragment:

```
<Name>
  <First> John </First>
  <Last> Doe </Last>
</Name>
```

Arriving at the Name node does not give the user much information. In order to be able to identify whether or not this is the entry that the user is looking for, the user needs more information. Furthermore, since the subtree itself is very small, the user may actually want to specify that when the cursor arrives at a Name element, the entire subtree should be rendered. This is the purpose of the *depth* attribute. In the ElementRendering case, the user can specify that the node and its children should both be rendered by specifying NODE_CHILDREN as the value for the attribute. In this case, a depth of NODE_CHILDREN would cause the system to render "Name, First John, Last Doe" when it arrived at the Name node. The *depth* attribute when used in the AttributeRendering context is very similar. If the user wants only the name of the attribute to be rendered, or perhaps just the value, then the *depth* attribute should change accordingly.

Finally, if the speech rules define an Action element, the *classname* attribute indicates that a class of the given type will be instantiated. If the *ismodifier* attribute is TRUE, then the AXL Mediator will acquire the Renderer lock before invoking the given class. After the class is invoked, the Mediator will replace the current node with the modified node and unlock the Renderer. This is the expected behavior of any Mediator.

ABL-Specific Rules. ABL also defines its own set of rules. ABL aims to allow the user to customize each Render element. The current implementation supports the following children of the Render element.

```
<!ELEMENT Tag EMPTY>
<!ATTLIST Tag
          rendername CDATA #REQUIRED>
<!ELEMENT Phrase EMPTY>
<!ATTLIST Phrase
          prenode CDATA #IMPLIED
          postnode CDATA #IMPLIED
          prechild CDATA #IMPLIED
          postchild CDATA #IMPLIED
          preattr CDATA #IMPLIED
          postattr CDATA #IMPLIED>
```

The Tag element allows the user to change the rendering from the tag name of the element to some user-specified value. This is useful for many reasons, one in particular being the discrepancy between tag names and actual English. Instead of speaking a tag name such as "STGDIR", which the engine may not even be able to convert to speech, the user could specify that the output should be "stage direction", a more understandable phrase.

The Phrase element allows the user to define different phrases that will be spoken before and after the rendering of each type of node. For example, a *prenode* phrase might be "You have arrived at a ", while a *postnode* phrase might simply be " node". The resulting rendering of a Name node would be "You have arrived at a Name node". Similarly, a *postchild* phrase might be "followed by" such that the rendering of the children might be something like "First followed by Last". Of course, all of these can be left blank. However, any or all of the attributes can be defined such that nodes can be connected together with a more sentence-like flow.

Another desired function is to specify the way in which the user can access a node of a given type. For example, the Tag element allows the user to specify how the tag of a given node should be interpreted in the spoken rendering of the node. It makes sense to have the semantics of this element attached to a given Render element because a given node may be spoken in a variety of different ways depending upon the given condition. However, it is also useful to allow the user to specify the way in which they wish to access a node of

the given type. However, this specification does not depend on any given condition of the node. For example, every time the user wishes to hear the "stgdir" node, the user should speak the same command. Therefore, we attach a *Speak* attribute to the ElementRule element of the rules. If the user wishes to access a "stgdir" node by saying "Give me the stage direction", then the value of the *Speak* attribute will be "Give me the stage direction".

The final extension that we have made to the schema is to add a Default element to the content model of the RenderRules element. The Speak and Default modifications yield the following changes to the base schema:

```
<!ELEMENT RenderRules ((ElementRule|AttributeRule)+, Default?)>
<!ELEMENT ElementRule (ElementRendering)+>
<!ATTLIST ElementRule
          elementtid CDATA #REQUIRED
          Speak CDATA #IMPLIED>
<!ELEMENT Default EMPTY>
<!ATTLIST Default
          parent CDATA #IMPLIED
          next CDATA #IMPLIED
          previous CDATA #IMPLIED
          nextAttr CDATA #IMPLIED
          previousAttr CDATA #IMPLIED
          name CDATA #IMPLIED
          value CDATA #IMPLIED
          exit CDATA #IMPLIED
          pause CDATA #IMPLIED
          resume CDATA #IMPLIED
          whatcanisay CDATA #IMPLIED
          stop CDATA #IMPLIED>
```

Each attribute represents one of the default commands in the current version of AXL. If the user wishes to change one of the default commands, the new command should be given as the value of the attribute. For example, if there were an element "name" in the actual schema of the document, there would clearly be a conflict between the command to access the name element and the command to get the name of the element pointed to by the current

cursor. This conflict can be resolved if the user changes the default name command to something else.

```
<Default name=''my identifier''>
```

The preceding example would change the default name command to "my identifier". When the computer hears the user say that phrase, it will reply with the tag name of the current node.

4.5 Evaluation Using a Study of User Experience

The goal of AXL is to demonstrate that we can automatically produce a usable system to access content using non-traditional devices and displays. The section demonstrates the usability of AXL by presenting a set of user-based experiments. First, we discuss how the experiments were designed and conducted. We then examine the observations and conclusions we have made.

Our main goal was to produce a system that would be usable for people with a variety of skill levels. We hoped that the command set we provided was intuitive enough to allow any user to extract information from an XML document without knowing about the underlying data representation. Further, we hoped that any user would be able to extract information from the document with a minimal learning curve. Finally, we hoped that users would be able to understand the context of the information they received from the document.

4.5.1 Design of the User Tasks

The study we conducted asked a set of users to navigate and extract information from an XML document outlining possible travel plans. The concept was that a user could plan a vacation online by navigating such a document to find the necessary information and choose a destination, find a flight to that destination, etc. The document we designed followed the following schema:

```
<!ELEMENT Travel (Flight-List, Rental-Car-List,
                  Vacation-Package-List)>

<!ELEMENT Flight-List (Flight*)>
<!ELEMENT Flight (Destination, Airline, Price)>
<!ELEMENT Destination (#PCDATA)>
<!ELEMENT Airline (#PCDATA)>
<!ELEMENT Price (#PCDATA)>


<!ELEMENT Rental-Car-List (Rental-Car*)>
<!ELEMENT Rental-Car (Type, Price)>
<!ELEMENT Type (#PCDATA)>

<!ELEMENT Vacation-Package-List (Vacation-Package*)>
<!ELEMENT Vacation-Package (Description, Airfare,
                            Hotel-Cost, Price)>
<!ELEMENT Airfare (#PCDATA)>
<!ELEMENT Hotel-Cost (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
```

Users were first asked to read six words and two sentences as minimal training for the ViaVoice system. Then, the subjects were given a brief set of oral directions and were shown the structure shown in 4.7 and told that the document they would be navigating would follow the same structure.

Following the oral directions, any questions were answered and the user was given a set of written instructions that told the user which commands to issue to navigate within the document. The user was then expected to execute each instruction without help from the proctor.

The first set of written directions was intended to familiarize the user with the system and the commands provided. The directions led the user down one path of the tree to a *Flight* element and investigated all of the information at the leaf level including price, destination, and airline. The users then continued to the second set of instructions which were designed to test whether or not the user could remember and apply the commands from the first section and successfully navigate the structure without explicit directions. The

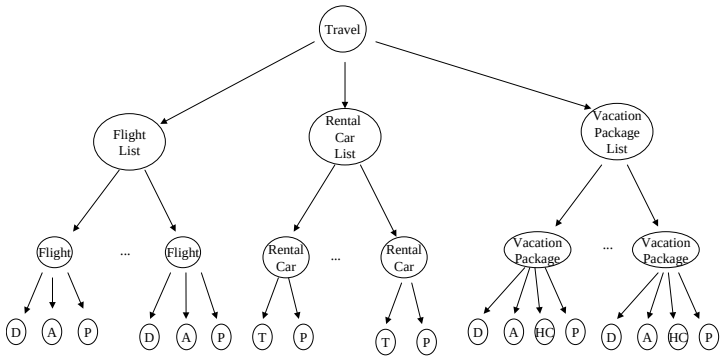

Figure 4.7: A tree representation of a travel document.

instructions asked the user to navigate from the root of the tree to a sibling *Flight* element and retrieve the same information retrieved in the first set of directions. The third set of directions led the user down a different path of the tree to the *Rental Car* element and tested the user's ability to find the price without explicit instructions on how to find the price of a *Rental Car*. The final set of instructions asked the user to navigate the tree to a *Vacation Package* element and asked them to insert a new node into the tree. The intention was to examine the user's ability and reaction to modifying the XML structure.

Finally, the users were asked a series of questions about their experience using the system. The questions involved rating and commenting on the command set provided, the appropriateness of the feedback provided by the system, and the user's prior experience including familiarity with XML.

4.5.2 Subject Profiles

We surveyed three preliminary subjects and found that the study itself needed to be modified. The subjects found that some of the instructions needed to be enhanced with further explanation. After slight modification of the study we surveyed five other people, all of whom were able to perform the given tasks. We found that the observations of both sets of subjects were very similar and thus discuss observations of all eight subjects.

All three preliminary subjects were familiar with XML. Of the subjects, two were male and one female. Moreover, two subjects had non-American accents. Of the five remaining subjects, four were female and one was male. One of the female subjects had a non-American accent and only one was familiar with XML.

4.5.3 Observations

We made five main observations:

1. The users' frustration with the rendering of some nodes facilitates the need for customization rules.

2. The automatic generation of a command set based upon schema yields commands considered natural by users.

3. The limited vocabulary produced by schema analysis aids the ViaVoice recognition engine.

4. There is a need to develop a better *help* command.

5. There is a need to develop enhanced feedback mechanisms.

The Need for Customization Rules. The study itself did not provide customization rules nor did it allow the users to write their own customization rules. Therefore, the rendering of all elements was simply the rendering of the

tag name or the tag name and text data if the element had the content model (PCDATA). Many users expressed frustration with the fact that when they asked for "Flight eight", that is to say that they moved the current cursor to the eighth flight element, the response from the computer was simply "Flight". Many of the users wanted the computer to say "Flight eight". One user even suggested that when she asked for the "Flight List" the response should be the price, destination, and airline of each flight in the list.

Despite the users' frustrations, this observation indicates that our customization language is imperative to allow users to easily specify their preferred method of rendering. By writing a simple set of customization rules using the customization language we have designed, the user could easily program the system to render each node in a more suitable manner.

The Command Set. We observed the users' perceptions of the command set in two ways. First, the initial set of questions we asked the subjects was designed to probe the user's opinion of the commands provided. We wanted to know how natural or intuitive the commands were for the user and whether or not it was difficult to understand the function of any of the commands. Moreover, we wanted to know if the subjects felt that the command set provided was complete or if the users found themselves in want of different commands. Second, the directions provided included instructions to access information, but allowed the user the freedom to choose between a Speech DOM level command or a schema-based command. For example, the users were asked to get the price of a given flight, but were able to choose between the command *price* and *last child*. We expected that the Speech DOM level commands would seem less natural to users and that the commands generated through schema analysis would be more natural.

We found that the users were generally pleased with the command set provided. As we had anticipated, most users indicated that the natural commands were the generated commands like *price* and *destination* while commands like *first child* and *parent* were less natural. When able to choose between a Speech

DOM command and a schema based command, most users chose the schema based commands (i.e. *price*). However surprisingly, one user did choose to use a Speech DOM command. We propose that those familiar with tree structures and the terminology related to them would be more inclined to use the Speech DOM commands. However, more novice users find the more natural commands most useful.

Recognition Accuracy. We were somewhat skeptical of the recognition accuracy of the ViaVoice system. It has been given accuracy ratings in the 85 percent range, but tends to require extensive training in order to perform well. However, we were encouraged to discover that with only two sentences worth of training, the system was able to understand a variety of people who had a variety of accents with high accuracy. We attribute this benefit to the restricted vocabulary used by the system. Because there are only a limited number of commands a user can speak, ViaVoice performs much better than in a general dictation scenario.

There were some instances of misunderstanding however. Background noise and misuse of the microphone (i.e. touching the microphone causing extra noise) accounted for some instances of the system not hearing or mishearing the user. In some cases, the system's complete lack of response caused the user frustration. We attribute this behavior to the speech recognition software and hence it is beyond the scope of this thesis.

A Help Command. The only help facility provided by the system is the *command list* command which lists all possible commands at the given node including all schema based commands as well as all Speech DOM commands. We made a couple of observations based upon the use of this command.

First, users expressed the desire for two separate help commands, one to hear the schema based and another to hear the Speech DOM commands. Some wanted to hear only the schema based commands, but were inundated with all of the possible commands for the system. Second, one user expressed

frustration by the fact that the commands listed were all *possible* commands. She wanted to hear only the commands that were valid for the given node in the given subject tree. For example, if an optional node were absent in the subject tree, the command to access that node would not be given.

We can first conclude that a thorough help facility needs to be integrated into the system. Because the system is designed to be used by a person who cannot see the document or its structure, we need to examine how to best communicate possible navigation paths and how to access them using only auditory input and output.

However, we also anticipate that there is a reasonable learning curve related to AXL. If each user underwent a short period of AXL training, they would be less likely to need such a help facility. Moreover, we predict that more advanced users would be likely to use the Speech DOM level commands more often simply because they would not want to learn a new set of commands for every new schema.

System Feedback. AXL attempts to provide feedback by using different sounds to convey different errors. In the initial instructions given to the users, they were told what each of the sounds meant, but most were unable to recall the meaning when they encountered the sounds while later traversing the document.

We believe that more extensive research must be done to determine which sounds are best suited for providing error feedback. In this case, one uniform beep would likely have been less confusing than the three different sounds the system emitted. However, three different sounds could be most useful if the sounds were carefully selected. Moreover, users expressed frustration in general with the feedback of the system in terms of communicating positioning within the document. A deployable system should integrate a mechanism to aurally communicate a relative position within a document in an understandable way.

4.6 Discussion

Next-generation environments promise to pose new challenges to providing access to information. Pervasive computing devices such as cellular phones, PDAs, digital clothing and jewelry, and embedded sensor devices do not support the traditional tools that a user would employ to navigate information. For example, a typical web user is accustomed to having a 17-inch monitor, mouse, and full sized keyboard to access web content. The user of a cellular telephone may not have any of the same tools available. Therefore, the challenge is to develop alternative mechanisms that can enable content-driven applications, such as web surfing, by providing access to information using whatever input and output resources are available.

The MakerFactory and AXL address that challenge. The MakerFactory is an architecture designed to support device and user-specific access to semi-structured, presentation-independent content. The MakerFactory can support a variety of mechanisms to access the same underlying content, represented in XML format. More specifically, AXL is a speech-based component that automatically generates customized, usable speech-based interfaces. For next-generation devices that have display constraints and cannot support traditional access tools, the MakerFactory and AXL enable content-driven applications by providing a usable solution to support content access and provide speech-based access and navigation of information.

Chapter 5

Resource-Aware Content Management for a Network of Personal Devices

5.1 Motivation

Personal computing devices are becoming increasingly pervasive. In the future, and even now, a person may go to work, to school, or simply walk down the street adorned with an entire collection of devices. These devices include PDAs, cell phones, laptop computers, and even clothing or jewelry. While each device is often limited in its capabilities because of constraints such as limited bandwidth, by improving coordination between the set of devices, the collection can become more powerful than any single device. One of the primary challenges in using a collection of devices in a coordinated way is managing the content stored on each device, and placing content where it will be most useful. This chapter presents a generalized architecture to manage content across a collection of devices in a resource-aware way. To validate the architecture, we evaluate a power-aware scheme that manages content by taking into account the remaining energy supply across a collection of devices.

5.1.1 Target Applications

As devices become more advanced, the range of applications they support is ever increasing [67]. In particular, increased disk space, memory, and bandwidth have enabled small devices to be used for more advanced, content-driven applications. While the first generation of PDA devices supported tasks such as calendar and address book maintenance, newer PDAs support access to text documents and multimedia content such as audio and video files. In the next generation, the same functionality may be supported by a user's jacket (http://www.media.mit.edu/hyperins/levis/). Moreover, increased connectivity between devices enables more advanced, content-driven applications. Today, we can imagine users on a subway train exchanging sections of the morning newspaper [49]. In the future, we can imagine millions of computers embedded in the environment, each gathering information and making sure the information is available where it is likely to be accessed.

Managing information in such a large-scale environment promises to be a great challenge. In fact, even in a more limited environment challenges arise. As devices become smaller, a single user is likely to carry a collection of devices. Even today we see business people who carry a PDA to keep track of appointments, a cellular phone to speak with colleagues, and a laptop computer to aid in presentations and perform other complex chores. A high school student might carry an MP3 player to listen to music, a cellular phone to send messages to friends, and a laptop computer to take notes in class. In the future, the same student may be wearing digital earrings to listen to music, a digital shirt to capture her voice and transmit her instant voice message to friends, and a digital jacket with a keypad to allow her to take notes in class.

Figure 5.1 provides a generic view of how a collection of devices can be used. When the user wants to view, create, modify, or remove a document from a device is his/her collection, he/she should be able to do so without regard for where that information is actually stored. In other words, the user should be able to pick up any device and access any piece of information he/she is

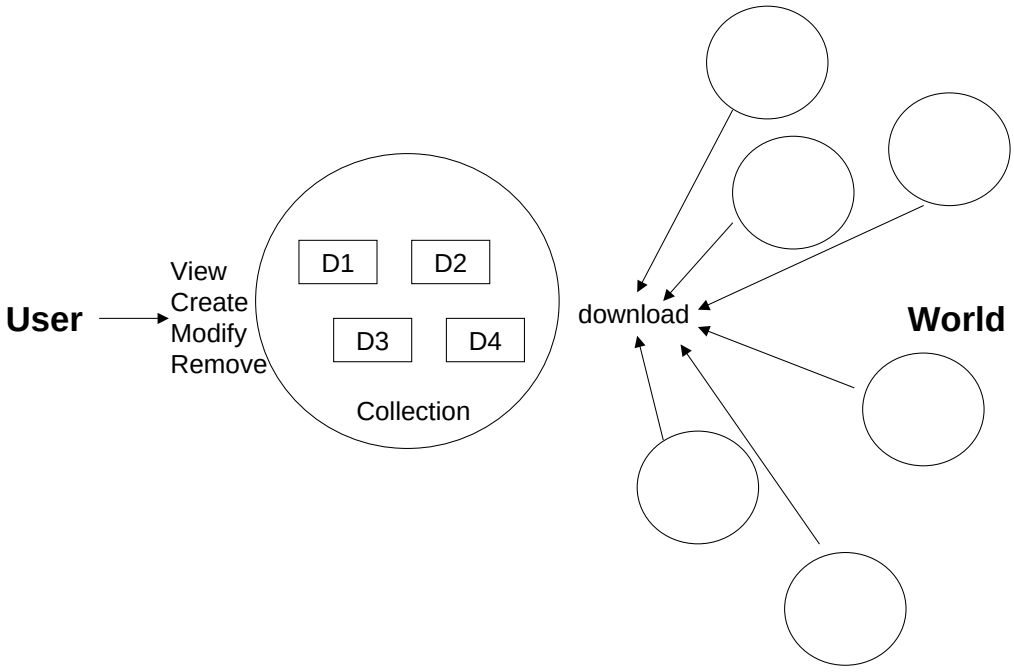

Figure 5.1: An overview of our model for use.

interested in accessing. Similarly, if the user is participating in an application such as peer content sharing, a user in the "outside world" should be able to seamlessly request download of content from device owner without having to specify on which device the content is stored.

To illustrate a more specific scenario, imagine a user on an airplane. She picks up her laptop and starts to modify a document. When the laptop runs out of battery, she picks up her PDA and continues to modify the document using the more limited interface. Eventually, the PDA runs out of battery and she switches to her cell phone. Finally, when her cell phone runs out of battery she resorts to her digital watch where she can dictate text using speech recognition. Today, state-of-the-art technology such as Bluetooth enables a user to manually transfer content from device to device. However, the burden is on the user to realize that a device is running low on battery and to select and transfer the files he/she might be using in the next few hours. In the ideal scenario, the user would be able to seamlessly switch from device to device

without having to worry about the state of the resources on each device or explicitly transfer the content from laptop, to PDA, and so on.

5.1.2 Device Limitations

As the practice of carrying multiple devices on a day-to-day basis becomes more commonplace, creating tools for automatically managing and coordinating them becomes necessary. The ideal situation from the user perspective is to accomplish *any task on any device*. For example, a task that a user would normally perform on a laptop should still be possible, even if the laptop's battery has died. The goal of content management is to automatically and seamlessly overcome many of the limitations of a collection of small devices by enabling devices to cooperate to accomplish content-driven tasks. Device limitations fall into two main categories:

Resource-Centric. Small, wireless devices are extremely limited with respect to resources such as bandwidth, processing, storage, and power. Uploading and downloading content using a PDA or even smaller device can be time consuming, and may not even complete in the alloted time slot if a user is highly mobile. Processing tasks such as decoding a video stream may be too CPU intensive for a small device. Constraints on the storage space of devices may prevent a user from having access to all the content she needs for a given application. Finally, energy is especially limiting since battery technology is not advancing as quickly as other resources [24]. Often, a single battery will only sustain a laptop for a couple of hours or less under a normal workload and, once a battery dies, a device becomes completely unusable.

User-Centric. While it would seem intuitive that a user who carries a collection of devices would be able to accomplish more than if the user carried a single device, managing and using the collection is often difficult. Despite redundant functionality, multiple devices often cannot be used to accomplish

the same task. The primary problem is lack of coordination. Especially with content-driven applications, if the content a user wants to access is not available on the device where the user is trying to access it, the task cannot be completed.

A generic framework for managing and coordinating devices should address both of these limitations by monitoring available resources, and making content available on the devices that the user is most likely to use to access that content. Resources can include CPU, network bandwidth, disk space, and battery lifetime. Moreover, the framework should perform tasks in a seamless manner such that the user can transition from device to device and access any content that may be of interest.

5.2 An Architecture for Supporting Content Management

The focus of this work is on developing an architecture to support resource-aware management of content. The idea behind a resource-aware content management architecture is to monitor the resources available across a collection of devices and predict where content is likely to be used or accessed. For example, if one device in a collection has a much faster CPU than the others, that device is likely to be used for compute-intensive tasks. Therefore, a data set likely to be used for a compute-intensive task should be migrated to the device with the fast CPU. This section outlines a generic architecture for monitoring resources and managing information based on knowledge of available resources.

5.2.1 Architectural Overview and Components

Figure 5.2 illustrates a high-level view of our architecture. A user carries a collection of devices (e.g., a Jornada, laptop, and digital watch). Each device

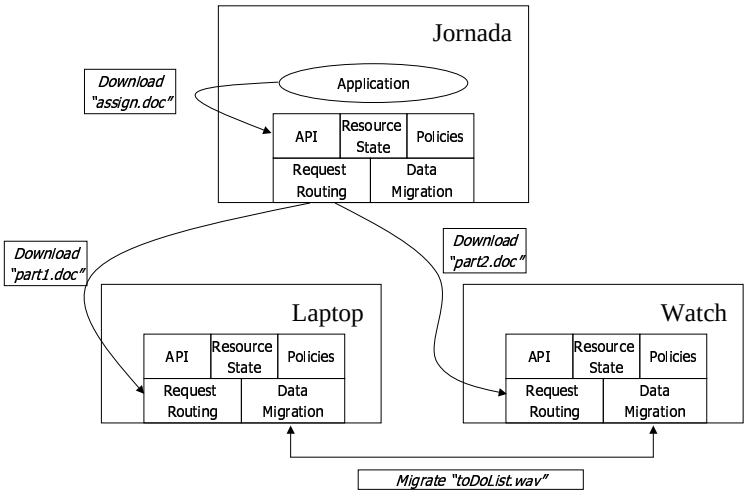

Figure 5.2: An architectural overview.

runs a middleware component that is configured such that it knows about the other devices in the collection and can communicate with them. Each device monitors its own resources, and content is managed based upon the resource view. For example, if a user's laptop and watch have higher bandwidth than his/her Jornada, the task of downloading content from an external source could be passed from the Jornada to the other devices. Similarly, the "todo" list a user stores on his/her watch could be migrated to his/her laptop if the watch starts to run low on battery.

The function of each architectural component is as follows:

- **Application Interface** - The application interface interacts with the application. It allows the application to request execution of tasks as well as provides a transparent view of the content that is stored across the entire collection of devices.

- **Policies and Preferences** - Policies and preferences keep track of user-specific information and are used to make decisions about where content should be placed for optimal access. For example, the user may specify that he/she will never read email on any device but his/her laptop

computer. Using this information, the architecture can reduce wasteful content migration by choosing never to migrate email to the user's Jornada or watch.

- **Resource State** - Each node in the device group maintains a view of the state of its resources, and any information it has about the resources available on the other devices in the group. Each device can communicate with the other devices in the collection to request information about remote resources when necessary. The resource state is used when making decisions about where content should be placed. For example, if the battery is low on a particular device, the content stored there should be moved from that device to another device in the collection. The resource state can provide crucial information used in deciding to where content should be migrated.

- **Request Routing** - When a task is issued by the application, the request routing component evaluates whether or not the task should be reassigned to another device in the collection. For example, if the user issues a download request on his/her Jornada, the request routing component may recognize that the user's laptop and watch have higher bandwidth, and may reroute the user's request to the alternate devices.

- **Data Migration** - The data migration component is responsible for ensuring that content is available where it is most likely to be used. For example, if the task of downloading is rerouted from a Jornada to a laptop and, subsequently, the user wants to access the downloaded content on the Jornada, the data migration component is responsible for transferring the content from the laptop. Similarly, if the battery on a particular device is dangerously low, the data migration component is responsible for migrating content away from the dying device to a device where it is more likely to be used.

5.2.2 Example Implementations

The goal of this architecture is to support content-driven applications by ensuring the content is available on the devices which have the best set of resources to provide access to that content. Examples of how this architecture can be used include:

- **Bandwidth-Awareness** - Applications such as peer-to-peer content exchange are bandwidth intensive. A user may want to upload and download large amounts of content, but may not be able to complete the task simply because the bandwidth is not available. The problem is exacerbated by the highly mobile nature of personal computing devices. However, by allocating bandwidth-intensive tasks across a collection of devices, tasks can be accomplished in a more efficient manner. A bandwidth-aware content management scheme can allocate the task of downloading information to the devices with the highest bandwidth while still making content available to the user on the user's preferred device.

- **CPU-Awareness** - Compute-intensive tasks such as decoding of video are often slow or even impossible on small devices. By dividing or offloading CPU-intensive tasks, or even simply assigning the entire task to a more powerful device, the collection can accomplish a task that would have otherwise been impossible on a single device. CPU-aware content management can ensure that data required to perform computation is made available on the device which has the CPU power to carry out the computation.

- **Storage-Awareness** - A small device may only have the disk space available to store a limited number of large data files, such as MP3s or MPEG movies, at any given time. By storing unused data on alternate devices, or even dividing a file such that it can be stored across a set of devices, the total available space can be used more efficiently.

Storage-aware content management can provide more efficient use of storage space while continuing to provide the user with easy access to information on the most appropriate device.

- **Power-Awareness** - Small devices have an inherently limited battery lifetime. Additionally, if the battery runs out on a particular device, the user cannot subsequently access any of the content stored on that device. A power-aware content management scheme can monitor the battery levels of each device in a collection, and migrate any critical data from a device that is dying. Doing so can extend the amount of time the collection remains usable because critical data can continue to be accessed from an alternate device.

5.3 Implementing a Scheme for Power-Aware Content Management

The architecture presented in Section 5.2 is very general. In order to provide a quantifiable measure of its usefulness, we must investigate how effectively an implementation of the architecture can monitor a particular resource, and manage content based upon the current state of that resource. This section investigates the implementation of a power-aware content management scheme. By focusing on a specific resource, power, we can gain a quantitative understanding of how beneficial content management can be in a given implementation. Additionally, power is perhaps the most limited resource in small device environments. While resources such as CPU and disk space continue to advance quickly, battery power lags behind. The impact of this can be seen in a variety of potential scenarios. In an environment such as a disaster site, an energy source such as an outlet is unlikely to be available at all. Alternatively, an environment such as an airport may offer a few outlets. However, there is no guarantee that a given traveler will be able to access an outlet in

the interim between flights. As devices with energy consuming network interfaces and displays become more commonplace, schemes to manage and reduce energy consumption will become more crucial.

Recall our model for use presented in Figure 5.1. The idea behind power-aware content management is that, even if device *D1* runs out of battery power (dies), the user should be able to continue working on one of the other devices in the collection. That means that information initially stored on *D1* should be migrated to an alternate device if the user is likely to access that information after the device dies.

5.3.1 Implementation Goals and Components

There are three underlying goals that should be supported by an implementation of power-aware content management:

1. **Maintain data availability** - As long as one device in the collection remains live, the user should be able to access any piece of information he/she is interested in accessing. This goal can be accomplished by migrating crucial content away from dying devices and placing it on devices that are still live.

2. **Support ease of access** - The user should be able to pick up any device and access a desired piece of content as quickly and effortlessly as possible. This goal can be accomplished by making intelligent predictions and decisions about where content should be migrated and made available to the user.

3. **Limit energy wasted during migration** - The energy spent migrating content should not affect the overall use of the collection of devices. This goal can be accomplished by limiting the amount of content migrated, and by intelligently choosing where to migrate content such that it need not be migrated multiple times before it is accessed.

To support these goals, we describe the implementation of the architectural components described in Section 5.2:

Application Interface. The application interface supports ease of access by presenting a global view of the files available across a collection of devices. Each device shares metadata about the files it stores with the other devices in the collection. Using this metadata, each device can present the user with a complete view of the distributed file system.

Policies and Preferences. Policies and preferences support ease of access and limit the energy wasted during migration. The user can specify a policy that indicates which data he/she is likely to access as well as on which device each piece of data is most likely to be accessed. Using this information, migration can be done in a more efficient manner, with a greater probability of migrating the *best* data to the *best* location.

Resource State. Monitoring of the state of the battery life of each device in the collection supports all three goals of power-aware content management. To monitor the state of the battery on each device, we simply poll the operating system for the remaining energy supply. Additionally, we assume that the energy consumption characteristics of the devices can be configured, or observed. Using this information, we can migrate content from dying devices, hence supporting the goal of maintaining data availability. Also, this information can be used as a predictor; the user is most likely to access information on the devices with the greatest remaining battery supply. By helping to predict where the user is likely to access information, the resource state component supports ease of access as well as limits the energy wasted during migration.

Request Routing. The request routing component supports ease of access by determining whether or not information requested by the user is available on the device where it is being requested. If the user requests information

that is not on the local device, the request routing component automatically initiates an on-demand data fetch to retrieve the information from the remote device.

Data Migration. The data migration component maintains data availability by migrating content from dying devices; supports ease of access by attempting to use predictors in making migration decisions; and limits energy wasted during migration by making intelligent choices about which pieces of content to migrate, and where those pieces of content should be migrated. The component behaves according the the following algorithm:

```
poll the resource state component for the remaining energy level
if the level is below a threshold
  migrate subset of items on this device to 1 or more
    target devices in the collection
repeat
```

This algorithm is quite general and leaves a number of questions unanswered. The first question is how the threshold for migration is determined. The threshold is determined by taking into account the remaining energy on the device, the amount of content to be migrated (e.g., all content in a specified directory), the network throughput of the device, and the power consumption characteristics of the device. We assume that the power consumption and network characteristics can be configured, or can be dynamically determined by observing system behavior. When the amount of energy remaining is equal to the amount of energy it will take to migrate all of the required content, migration is initiated.

Migration presents two primary challenges: (1) determining which data to migrate from a dying device, and (2) choosing the destination device(s). The following two subsections address each of these challenges.

Choosing the Data to Migrate

Choosing which data to migrate from a dying device is challenging. Migrating too little content may not be helpful since the user may want to access

a piece of content that did not get migrated. Migrating too much content is wasteful of the resources, such as energy, required to transmit content. Because the answer to this question is not straightforward, we investigate three strategies for choosing which data to migrate:

- **Migrate All (MA)** - In a migrate all strategy, all data is migrated from the device losing power. While it may not be practical to migrate *everything* on a particular device, this strategy can be implemented by migrating all data that is located in a specified directory.
- **Migrate Most Recently Used (MRU)** - In a most recently used strategy, all data that has been accessed recently by the user will be migrated. This strategy assumes that a user is likely to access some specific subset of the available content over a given time period. For example, over the course of a few hours, the user might access only files related to one specific project.
- **Access Optimized (AO)** - In an access optimized scheme, we assume that we have some knowledge of which data the user is likely to access. Either the user can specify a list of content that is likely to be accessed, or we can implement some form of user profiling.

Choosing the Destination Device

The final challenge addresses the strategy for choosing the destination device(s). We can imagine a number of very simple schemes such as round-robin or simply choosing the device with the largest remaining power supply. However, such schemes may be ineffective, or wasteful. Using a round-robin scheme, content may be migrated from a dying device to another dying device. In this case, the content will only have to be migrated again, likely before it is even used. Choosing the device with the largest remaining power supply may also be ineffective if the target device has very high energy usage characteristics. Additionally, considerations such as available storage space or bandwidth

of the target device are also likely to have an effect on performance.

In this work, we investigate a **Power Aware (PA)** scheme for choosing the destination device. The amount of content migrated to a particular device is based on the remaining lifetime of that device with respect to the lifetimes of the other devices. We poll each live device d in the collection for the amount of time it can continue work before it reaches its threshold (rl_d). For each device d, the number of items migrated to d with respect to the total number of items *(ni)* is $(rl_d * ni)/\sum_{i=1}^{N} rl_i$. Similarly, we could consider the total size of the items instead of the number of items. However, for the purposes of our experiments, we assume that all files are roughly the same size. This formula ensures that the amount of content migrated to a given device is proportional to the remaining lifetime of the device with respect to the remaining lifetimes of the other devices in the collection. The goal of this scheme is to ensure that content is available on devices that are likely to be alive when the user tries to use them.

We could also imagine even more advanced schemes such as predicting the remaining energy supply *after* migration occurs, or predicting the device where the user is most likely to access the specific content. In the first case, implementing such a scheme would only be necessary if the ratio of energy consumed to remaining energy supply varied greatly across the collection of devices. The second option would avoid the case of migrating content to a device, and then having to fetch it, on-demand, from another device before it can be used. However, to implement such a scheme, we need to either explore user profiling techniques, or require the user to specify where content is likely to accessed.

5.4 Evaluating the Tradeoffs of Power-Aware Content Management

The goal of our evaluation is to explore the tradeoffs associated with the content migration schemes presented in the previous section, and to explore

the overall benefit of using migration. To fully understand the benefits and tradeoffs, it is necessary to explore the behavior of each scheme under a variety of conditions. This section presents the set up and results of a set of simulated experiments. By using a simulation to explore the behavior of migration, we have the freedom to modify both device characteristics and user usage patterns. This enables us to gain a more complete understanding of the benefits and tradeoffs of using power-aware content management schemes.

5.4.1 Simulation Overview

Our simulator models a set of personal devices and simulates a single user accessing content on those devices. An example would be a user who is traveling on an airplane. We suppose the user carries a collection of devices; perhaps a laptop, a PDA, a cell phone, and a digital watch. When the user boards the airplane, all devices are fully charged. For the first two hours of the flight, the user uses his/her laptop to access information stored there (e.g., a PowerPoint presentation, email, or an MPEG movie). When the battery on the laptop dies, the user continues to access the same information on the PDA device. When the PDA dies, the user switches to the cell phone, and eventually to the digital watch.

There are a number of assumptions inherent in this example scenario. First, all of the devices must have similar functionality. If a piece of content is migrated from one device to another, both devices should have the capability to access and/or modify that piece of content. For example, if an MP3 file is migrated from one device to another, both devices must have speakers and an MP3 player installed. This is certainly true of many devices today. It is not uncommon for a laptop, PDA, and cell phone to have overlapping functionality.

Next, we assume that a user uses one device at a time, and uses that device until the battery has been exhausted. In other words, the user does not switch from laptop, to cell phone, back to laptop, nor does the user listen to an MP3 on his/her watch while working on a PowerPoint document on his/her laptop.

Scenarios where the user accesses multiple devices simultaneously do exist, and would reduce the benefit of migration. However, our assumption of serial access is a reasonable assumption for many application scenarios such as the airplane example described above.

Finally, we assume that at any given time, one device is in use, and the remaining devices are idle (or have run out of power). In practice, an intelligent energy saving scheme that puts all devices into sleep mode when no device is in use is likely to be employed. We assume such a scheme is used, therefore we do not simulate intermittent periods when all devices are idle. We could additionally employ energy saving techniques to reduce energy consumption of the idle devices when one device is in use. However, for the purposes of this work, we assume that all unused devices must be on and idle such that any content not available on the device currently in use can be fetched on-demand. That is to say, if a user wants to listen to an MP3 stored on her PDA while she is currently using her laptop, the file will be fetched, on-demand, from the PDA.

The simulation proceeds as follows:

```
generate trace of items to be accessed
for device 1 to n
  while battery threshold not reached
    while trace contains a next item
      if item is on another device
        fetch on-demand
        record a fetch
      else if item is not available
        record total miss
        continue
      access item for specified time
  initiate migration
```

The trace of accesses is a list of <*item*, *time*> pairs. Each pair indicates that the the given *item* should be accessed for the specified *time*. The trace is finite and the total trace time is equal to the number of seconds that the device collection would be available in the ideal case. The battery threshold

is determined as described in Section 5.3. Similarly, migration is performed according to the algorithms described in Section 5.3.

5.4.2 Metrics

Our evaluation looks at three primary metrics:

- **Time Devices Remain Usable** - The primary metric we are interested in is the total time that a user can work. The goal of migration is to increase the total amount of time a collection of devices remains usable by migrating relevant items from devices with low battery supplies to devices with more plentiful battery supplies. Therefore, even if a user is interested in accessing an item that was originally stored on a device that has died, the user may be able to access the item elsewhere and hence continue to work.

- **Data Misses** - A secondary metric is the number of misses the user experiences. A miss occurs when the user picks up a device and tries to access an item not available there. There are three miss scenarios: (1) the item was originally stored on device *d2* and the user is trying to access the item on device *d1*; (2) the item i was migrated from device *d2* to device *d3*, but the user is trying to access the item on device *d1*; and (3) the item was originally stored on device *d1* and was not migrated before device *d1* died. In scenarios 1 and 2, the item will be fetched, on-demand, from the live device where it is currently stored. However, scenario 2 indicates that the migration algorithm has failed to predict the correct device and energy is wasted migrating the item twice. Scenario 3 indicates that the migration algorithm has failed completely and the user will have to move to the next item he/she is interested in accessing (e.g., the user cannot access his/her PowerPoint presentation, so he/she decides to read email instead).

- **Wasted Migrations** - The third metric is the amount of energy that is wasted by migrating items that are never accessed after migration. Migrating data requires additional energy that would not otherwise be consumed if a user were simply working on the device. Therefore, the energy spent migrating unnecessary items is likely to reduce the total amount of time a user can work.

5.4.3 Setup

Full Energy Supply (J)	Idle (W)	Active (W)	I/R (W)	A/R (W)	I/T (W)	A/T (W)
90000	3.93944	13.839	9.2006	14.0006	9.64615	14.44615

Table 5.1: Energy consumption characteristics of devices simulated.

The simulator models a set of large devices such as a laptop computers. Certainly, different devices have varying energy consumption characteristics. Moreover, smaller devices such as PDAs consume much less absolute energy. However, we have chosen the large device model for two reasons. First, we were able to build a more accurate large device model based on previously published energy consumption measurements. Also, while absolute energy consumption varies, most devices have similar lifetimes. Typically, devices consume energy at the same rate given the full energy supply, and energy consumption characteristics of the device. Farkas et al. confirm this notion with their measurements of the Itsy Pocket Computer [17]. Therefore, our results hold true for any device that has a lifetime similar to that of our large device.

The characteristics of the device we model are shown in Table 5.1. These characteristics are taken from measurements reported by Farkas et al. [17]. They measured power consumption of the IBM ThinkPad 560x running at 233MHz using 64Mbytes of memory. We use the idle, busy wait, and busy wait with LCD-enabled benchmarks. While different workloads may consume

different amounts of energy, differences are often slight and these measurements are reasonable for the granularity of our experiments.

Because Farkas et al. do not measure the power consumption of a network interface, we use the measurements taken by Feeney and Nilsson [18]. They measure the consumption of a 2.4GHz DSSS Lucent IEEE 802.11 WaveLAN PC Silver card (11Mbps) running on an IBM ThinkPad 560. To obtain the device characteristics shown in Table 5.1, we have added the measurements reported by Feeney and Nilsson (idle, receiving, and transmitting) to the measurements reported by Farkas et al. and Hill et al. This provides us with a reasonable model for our simulation. Finally, the full energy supply measurement is taken from numbers reported by Flinn and Satyanarayanan [19].

A device can be in one of six states:

- **Idle** - In the idle state, all components of the device are in idle mode including the display and network card.

- **Active** - In the active state, the processor and display are busy, and the network card is idle. This state models the user accessing content on the device without having any network interaction.

- **Inactive Receiving (I/R)** - In the inactive receiving state, the processor is busy, the display is idle, and the network card is in receiving mode. This state models the device receiving content migrated from another device. However, no content is being accessed on the receiving device.

- **Active Receiving (A/R)** - In the active receiving state, the processor and display are busy, and the network card is in receiving mode. This state models the device receiving the content migrated from another device while the user is accessing content on the receiving device.

- **Inactive Transmitting (I/T)** - In the inactive transmitting mode, the processor is busy, the display is idle, and the network card is in

transmitting mode. This state models migrating content from a device while the user is not accessing content on that device.

- **Active Transmitting (A/T)** - In the active transmitting state, the processor and display are both busy, and the network card is in transmitting mode. This state models migrating content from a device that the user is currently using to access content.

A device transitions into an *active* state if the user attempts to access content on the device. A device transitions into a *receiving* state if content is migrated to the device, or if the device fetches content from another device on-demand. Finally, a device transitions into a *transmitting* state if the device is migrating content to another device, or is responding to an on-demand fetch.

We compare six migration schemes based on the strategies described in Section 5.3. In the first scheme, **NONE**, no migration is employed. In the second scheme, power aware migrate all (**PAMA**), we use a power aware scheme for choosing the destination device, and we migrate all data from the current device. In the third scheme, power aware most recently used (**PAMRU**), we use a power aware scheme for choosing the destination device and we use a most recently used scheme for determining which data to migrate. In the most recently used strategy, we do not assume any history is kept from session to session and all data migrated is that which has been accessed since the simulation began. In the fourth scheme, power aware access optimized (**AO**), we look ahead in the trace of items to model user preference. In essence, we assume that we have perfect knowledge of the documents the user will access and migrate only those items that appear in the remainder of the trace. In the fifth scheme, ideal access optimized (**IAO**), we assume that all data is already in its ideal location such that no migration is necessary and a user can always continue working. In the final scheme, ideal power and access optimized (**IPAO**), we make the same assumptions as in the IAO scheme. However, we further assume that any device not in use can be safely turned off. Thus, this scheme wastes no power in idle mode.

Parameter	Normative Value	Range Tested
Storage	32MB	128KB - 1GB
Access Time	20 mintues	5 mintues - 40 mintues
Number of Items	128	16 - 1024
Bandwidth	11Mbps	106Kbps - 54Mbps
Idle Energy	3.9W	0W - 3.9W
Number of Devices	4	2 - 10
Trace Distribution	Zipf	Zipf, uniform

Table 5.2: Energy consumption characteristics of devices simulated.

Table 5.2 summarizes the parameters we examine in our experiments. Below, we summarize each parameter and its anticipated effects:

- **Storage** - Storage is the amount of disk space that is initially used on each device. As the amount of space used increases, more content is likely to be migrated, especially in the migrate all scheme. Migrating more content requires more energy, which, in turn, is likely to reduce the overall availability of the collection of devices.

- **Access Time** - Access time is the amount of time each particular item is accessed by the user. It is the *time* component of the trace. The shorter each access is, the better PAMRU is likely to perform. The logic follows from the fact that when access times are shorter, a longer history will be generated and more items ultimately migrated. However, the other schemes are unlikely to be greatly affected.

- **Number of Items** - Number of items is the total number of items initially stored on each device. The more items, the smaller each item since the storage remains fixed. Again, this parameter is most likely to affect the PAMRU scheme. When the number of items is larger, there is a higher probability that the migration scheme will fail to migrate an item that will be later requested by the user.

- **Bandwidth** - Bandwidth is the available bandwidth of each device in the collection. Devices with lower bandwidth will spend more time in transmit mode, migrating content. Thus, the devices will use more energy overall. By using more energy to migrate content, the collection of devices will be, ultimately, less available.

- **Idle Energy** - Idle energy is the power used by each device while in the idle state. It is clear that the less energy consumed in idle state, the longer the collection will remain available overall. However, the purpose of varying this parameter is to get an idea of how useful migration is, versus strategies that can reduce the absolute energy consumption of a device.

- **Number of Devices** - Number of devices is the total number of devices used in the experiment. In the IPAO scheme, the time devices remain usable should increase linearly. However, since the other schemes consume a great deal of energy in idle mode, they are unlikely to match the linear increase.

- **Trace Distribution** - The trace distribution is the distribution used for the *item*s selected for the trace. We compare a Zipf distribution to a uniform distribution. The Zipf distribution models many applications wherein the user is primarily interested in only a few of the items available on his/her devices. It is likely that migration will perform well for these kinds of applications because it should be easier to predict which pieces of content should be migrated. In contrast, a uniform distribution models the scenario wherein the user may be interested in any piece of content. Migration is less likely to perform well for these kinds of applications.

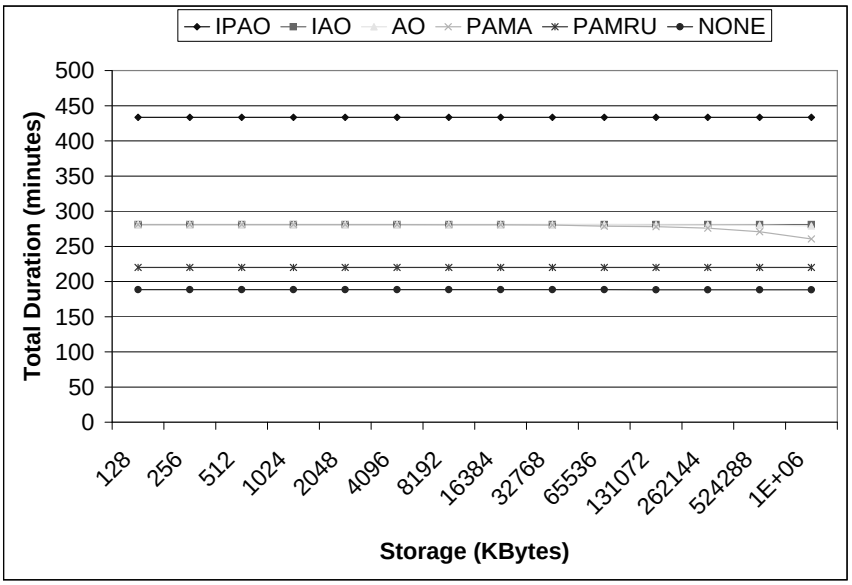

Figure 5.3: Usable duration for varied storage using Zipf access pattern.

5.4.4 Results

Figures 5.3 and 5.4 illustrate the total time the collection of devices remains usable as the total storage space initially used on each device varies. Primarily, these graphs illustrate the general relationship between each of the migration schemes, and demonstrate that migration is useful under a variety of conditions. More specifically, for the Zipf trace pattern, PAMA shows up to a nearly 50% improvement over NONE; well over a 1.5 hour total increase in the amount of time the collection of devices is usable. Additionally, the PAMA scheme performs very close to IAO, the ideal scenario, for most initial storage configurations. PAMA also performs very close to AO for most initial storage configurations. This suggests that we can achieve good results without implementing difficult user profiling techniques that would be necessary for AO. Finally, no migration scheme comes close to the results achievable with IPAO. This is because the power consumption of the devices during idle time is quite high. In order to compete with the IPAO scheme, the devices not in use would need to employ more intelligent, power saving techniques.

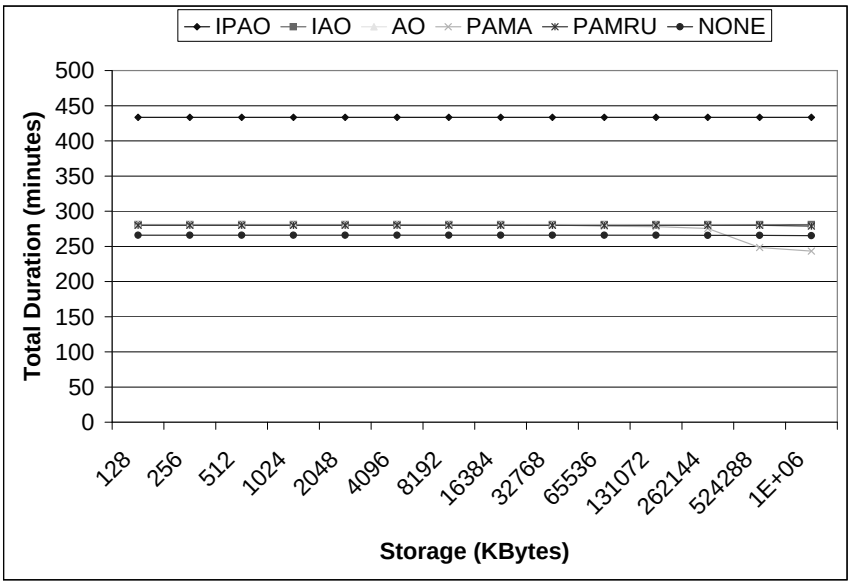

Figure 5.4: Usable duration for varied storage using uniform access pattern.

Unfortunately, PAMRU performs well below PAMA in most cases. While PAMRU does remain stable for all initial storage configurations, it only performs marginally better than NONE. The most recently used strategy alone is not sufficient to predict which pieces of content the user is most likely to access in the future. However, the benefit of the PAMRU strategy over PAMA is that less energy is consumed migrating content that is not going to be accessed in the future.

As the initial storage used on each device begins to exceed 256MBytes, the wastefulness of the PAMA scheme becomes more clear. The total usable duration begins to decline because the devices are wasting energy migrating content that will never be accessed. While this is an expected result, the decline is not as significant as we initially anticipated. Because the default device bandwidth is relatively fast, the energy wasted transferring content is minimal. In a lower bandwidth environment, this is less likely to be the case.

Figure 5.4 illustrates the same parameter, but for a uniform distribution of items accessed. In the uniform case, NONE performs much better and the benefit of migration is decreased. This is an expected result. In the Zipf

case, when a device that stores one or more pieces of popular content dies, the total usable time of the collection is dramatically decreased. In contrast, the uniform case models the situation where a user can make use of virtually any piece of content stored on any device. However, there are a large number of applications that do not follow the uniform model and, hence, will benefit from a content migration scheme.

An additional observation is that, much like in the Zipf case, PAMA starts to decline in performance when the initial storage used on each device exceeds 256MBytes. In this case, PAMA actually performs worse than NONE for large values of storage used. Again, this result is not surprising. The conclusion that can be drawn based on this set of experiments is that, for many applications, a migration scheme can be very beneficial. However, for the applications that follow the uniform access model, migration may actually be more costly than no migration at all.

Table 5.3 illustrates the number of items that are migrated and never accessed on the new device (Wasted Migrations) and the number of times a piece of content must be fetched on-demand (Fetches) or cannot be accessed because it was not migrated from a device that died (Total Misses). Migrations do not occur in the IPAO and IAO schemes, therefore results for those schemes are not presented. The results presented are for the Zipf trace distribution. Results for the uniform distribution as well as for the other parameters to be presented in this section were similar, and will not be presented.

First, the results confirm that PAMA is very wasteful of energy. This is not surprising since PAMA essentially implements a fully replicated file system across all devices in a collection. In some cases, this may be an acceptable solution. In fact, the analysis of Figures 5.3 and 5.4 indicates that full replication does not have a substantial impact on the overall availability of the collection. But, full replication not only impacts the energy spent transferring content, it also impacts the storage space used on the collection of devices. Replicating all data onto a watch may not be feasible. Therefore, the tradeoff

is to use a scheme, such as PAMRU, that incurs the penalty of increased Total Misses but reduces migrations. It is noteworthy to point out that the trace of requests is finite, but slightly exceeds the total amount of time the collection of devices will remain available. There are two impacts of this. First, this is why the AO scheme experiences a non-zero number of Wasted migrations. Also, since we assume that a user's list of tasks exceeds the amount of time she will actually have to complete them, even if a Total Miss is experienced, the user may be able to continue working. Making the list of tasks longer can increase the time work can be done in both the NONE and PAMRU schemes.

Storage (KB)	Scheme	Total Migrations	Wasted Migrations	Access Attempts	Fetches	Total Misses
128	AO	26	17	18	5	0
	PAMA	559	548	18	3	0
	PAMRU	11	9	24	2	11
	NONE	0	0	24	2	13
512	AO	26	17	18	5	0
	PAMA	561	550	18	3	0
	PAMRU	11	9	24	2	11
	NONE	0	0	24	2	13
2048	AO	26	17	18	5	0
	PAMA	561	550	18	3	0
	PAMRU	11	9	24	2	11
	NONE	0	0	24	2	13
8192	AO	26	17	18	5	0
	PAMA	561	550	18	3	0
	PAMRU	11	9	24	2	11
	NONE	0	0	24	2	13
32768	AO	26	17	18	5	0
	PAMA	561	550	18	3	0
	PAMRU	11	9	24	2	11
	NONE	0	0	24	2	13
131072	AO	26	17	18	5	0
	PAMA	560	551	16	2	0
	PAMRU	11	9	24	2	11
	NONE	0	0	24	2	13
524288	AO	26	17	18	5	0
	PAMA	560	550	17	2	0
	PAMRU	11	9	24	2	11
	NONE	0	0	24	2	13

Table 5.3: Misses and wasted migrations for varied storage using Zipf access pattern.

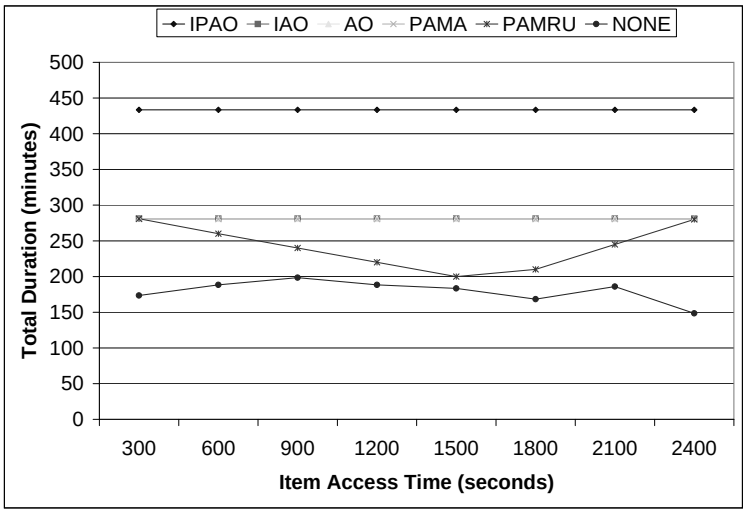

Figure 5.5: Usable duration for varied access time using Zipf access pattern.

Figure 5.5 illustrates the total time the collection of devices remains usable as the amount of time each item is accessed varies. The results discussed use the Zipf pattern for the distribution of accesses. A comparison of the results for the Zipf pattern versus the results for a uniform pattern yield the same observations made in the results for the varied initial storage used on each device. Therefore, the remainder of the results presented are for the Zipf access pattern only.

Not surprisingly, PAMA, AO, and IAO are unaffected by the change in access time. PAMA consistently performs well and shows up to an 88% improvement over NONE. Unfortunately, PAMRU and NONE are not as consistent. The lack of any reasonable trend indicates that the access time is not as important as the trace of items accessed during the experiment. While PAMRU sometimes performs well, NONE consistently performs poorly and seems to perform worse for longer access times. This result follows from the fact that longer access times mean that there are ultimately fewer items accessed. If one of the few items accessed is stored on a device that dies, the total time the device collection remains usable can be greatly reduced.

Figure 5.6 illustrates the total time the collection of devices remains usable

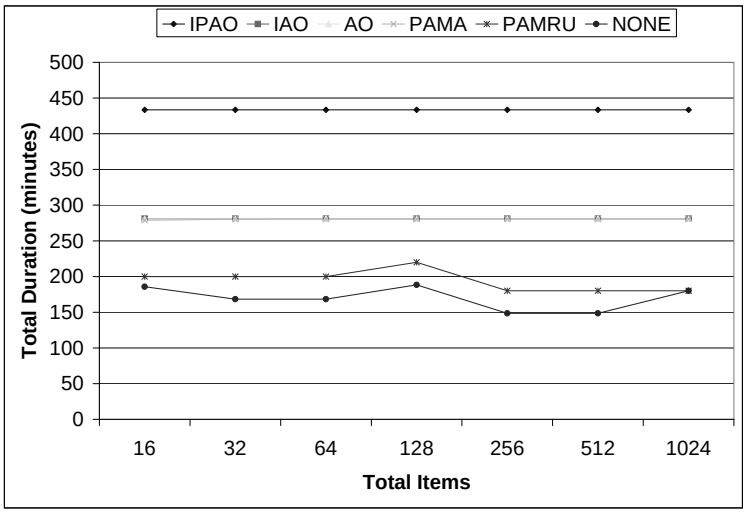

Figure 5.6: Usable duration for varied number of items using Zipf access pattern.

as the total number of items initially stored on each device grows. As expected, the results for PAMA, AO, and IAO are not affected by the change in number of items. However, as seen in the previous experiment, the results for PAMRU and NONE are very inconsistent. PAMRU should perform better for smaller values of total items because fewer total items means that each item will be larger. Thus, a single migration ensures that more of the total base of content remains available. However, the inconsistency in the experiment indicates that, with PAMRU, poor prediction of future content access is the cause of mediocre performance. Similarly, with NONE, poor performance can be attributed to the content access pattern.

Figure 5.7 illustrates the total time the collection of devices remains usable as the device bandwidth varies. Overall, the affect of lower bandwidth is similar to the affect of larger initial storage used. This is a logical result since it takes longer both to migrate more content or to migrate content over a lower bandwidth connection. The more time a device spends migrating, the more energy the device uses during migration, and the shorter the duration that the collection of devices remains usable.

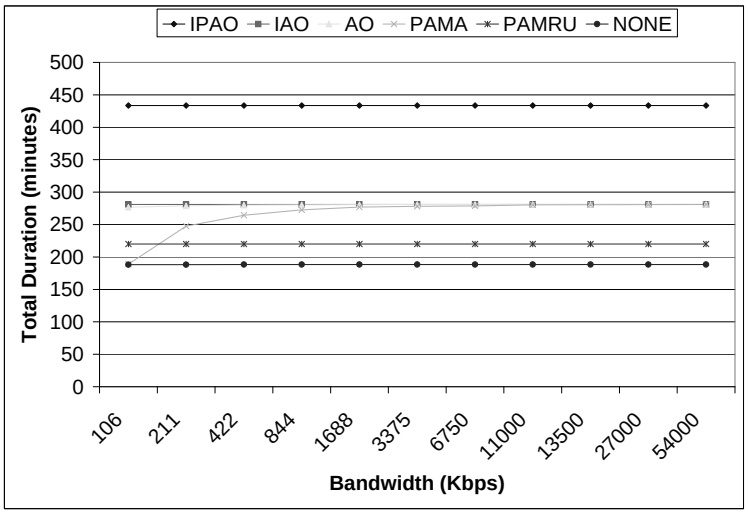

Figure 5.7: Usable duration for varied bandwidth using Zipf access pattern.

More specifically, PAMA starts to drop at bandwidths below 1.5Mbps while PAMRU remains unaffected. Again, this is the result of the wastefulness of the PAMA scheme. A slow connection between devices increases the transfer time and ultimately the power used. While it is possible that a lower bandwidth connection may actually require less energy during transfer, the results indicate that a migrate all scheme is not the best option for low bandwidth devices.

Figure 5.8 illustrates the total time the collection of devices remains usable as the idle power consumption varies. As the energy consumed in idle state is reduced, the IAO, AO, and PAMA schemes approach the performance of the IPAO scheme. While PAMA is consistently similar to IAO, PAMRU does not perform as well and is unaffected by the change in power consumption. This result again illustrates that the bottleneck with the PAMRU scheme is the trace of items accessed, and not the energy consumed during content migration.

Figure 5.9 illustrates the total time the collection of devices remains usable as the total number of devices grows. For IPAO, there is a linear increase in usable time as the number of devices gets larger. While PAMA does not approach linear growth, it does remain consistently similar to the IAO strategy.

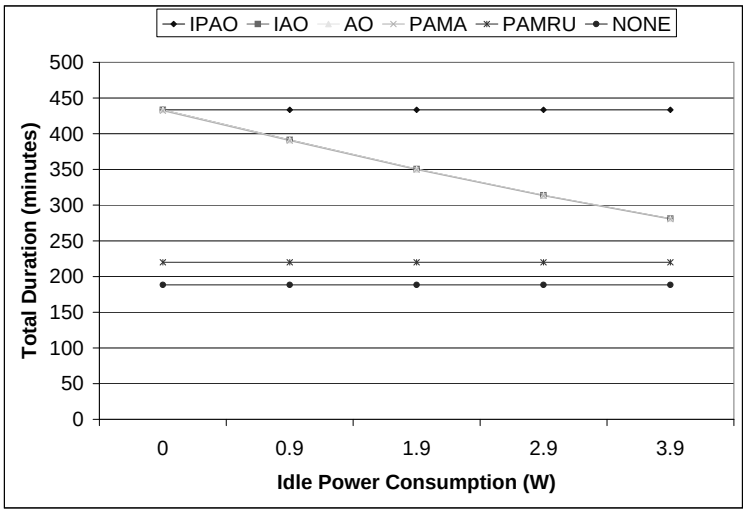

Figure 5.8: Usable duration for varied idle power consumption using Zipf access pattern.

Recall that IAO is the ideal result given that all devices remain on and idle throughout the course of the experiment. Unfortunately, PAMRU suffers the same fate as in previous experiments. It shows marginal improvement over NONE, and does not follow the same trend we see with PAMA.

5.4.5 Observations

Migration performs well in a variety of scenarios. In most cases, the cost in terms of energy consumed migrating content does not outweigh the benefit with respect to the total amount of time the collection of devices remains usable. An exceptional case is the uniform distribution for item access. For applications that offer flexibility in terms of which pieces of content can be used, migration may not be useful or necessary. However, for many common applications, the user focuses on only a few select pieces of content. The Zipf access pattern is a reasonable model for these kinds of applications.

Somewhat surprisingly, the PAMA scheme, which mimics a fully replicated file system, performs well in most cases. While PAMA is quite wasteful in terms of energy spent transferring content, the overall impact is minimal

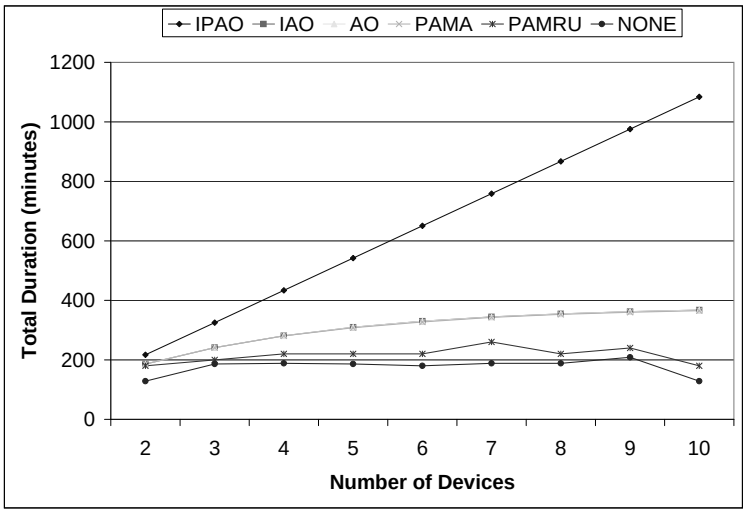

Figure 5.9: Usable duration for varied number of devices using Zipf access pattern.

in most cases. There may be additional factors, such as limited disk space, that prevent PAMA from being truly effective. However, from an energy perspective, migrating all content can be effective if the devices have reasonable available bandwidth and a reasonable amount of content.

Unfortunately, in the extreme cases when devices have low bandwidth, or a large base of initial content, it is clear that an intelligent form of user profiling should be employed to ensure extended device usability. Naive schemes such as PAMRU are not sufficient to increase the amount of time a collection of devices remains usable. While user profiling is a very challenging problem, ultimately a solution is likely to include some form of user profiling along with the use of user-specified preferences. The more information the user can provide about which content is likely to be accessed, the more effective migration can be.

Finally, while migration strategies aid in dealing with the challenge of power constraints, it is imperative to integrate strategies to reduce absolute power consumption, during idle periods or otherwise. Results of our experiments clearly show that reducing energy used during idle time greatly effects the amount of time a collection of devices can remain usable. Ultimately, a

resource-aware content management solution must manage content when resources are limited, but must also use resources in the most intelligent manner possible.

5.5 Discussion

The pervasive environment introduces a new dimension to consider when designing a content management scheme. Next-generation devices are incredibly resource-constrained in comparison to their desktop counterparts. Resources such as processing power, disk space, bandwidth, and battery lifetime are all relatively scarce. Interestingly, content management can be used as a solution to ease resource constraints by enabling devices to cooperate and share resources. However, a content management scheme must intelligently consider resource availability in order to aid in device cooperation and resource aggregation.

This work presents an architecture that supports resource-aware content management. The architecture allows a user's personal collection of devices to communicate and share resource state. Based on that state, intelligent content placement decisions can be made. More specifically, the evaluation focuses on the resource of power. Battery lifetime is typically the most constrained resource for a small device. The proposed technique provides support for content-driven applications by monitoring the battery levels on each device and ensuring that content remains available in the face of power loss.

Chapter 6

Using Push and Batching to Enable Efficient Content Location and Distribution

6.1 Motivation

Napster's pioneering efforts have spawned a number of academic and industrial projects aimed at developing efficient, peer-to-peer (P2P) applications. A primary goal of these applications is to support content exchange between users. In the peer environment, support for content exchange involves two phases. In the first phase, a user searches the network to find a remote peer that stores a particular piece of content that he/she is interested in downloading. In the second phase, the user requests download of the content from the remote user. If the remote user decides to service the request, that user is responsible for distributing content to the requester.

The primary use of these applications has, so far, been the sharing of MP3 music files between desktop computers connected via the wired Internet. However, the concept of P2P is expanding. One area that has recently received a great deal of attention is pervasive computing. The promise of the ubiquitous deployment of small devices has introduced a new, more constrained environment where P2P is not only a desired, but necessary model for content exchange as well as other applications. Technology such as Bluetooth

and even 802.11 allows devices to talk to other devices within a certain range, but may not support infrastructure-based, Internet connectivity. This kind of environment is inherently P2P, and as such, P2P architectures are necessary to enable advanced applications on small computing devices.

However, especially in small device environments, P2P content exchange is challenging. In a traditional peer network, participants may come and go by choice. In a small device environment, participants may come and go by choice, or may simply walk out of range of the other participants. In a traditional peer network, end-user computers may have limited resources such as bandwidth (e.g., a modem connection), disk space, and processing speed. However, compare the resources of a desktop computer to a PDA device, or even digital jewelry or clothing.

As peer networks expand and evolve, an efficient scheme for exchanging content is required. Much previous research has focused on the challenge of peer discovery and group management. This work focuses on the challenges of data location and reliable and efficient delivery. We have developed an architecture called *Pixie*. *Pixie* addresses the problem of data location by providing a browsable catalogue of popular content available across the network. Moreover, the catalogue caches the location of content making data location more resource efficient. Additionally, we address the challenge of efficient delivery of content by batching requests for content and servicing hundreds or thousands of requests simultaneously. From the client perspective, this greatly reduces the wait time experienced after issuing a request. From the server perspective, we greatly reduce the resources required at the serving peer including disk space, distribution time, and bandwidth.

6.2 The Jukebox Paradigm

The *Pixie* architecture represents the evolution of a traditional jukebox paradigm. A jukebox works by scheduling playout of requested songs. Users can browse the scheduled songs and choose to wait for a song already scheduled

for playout. Alternatively, a user can schedule a new song for playout if he/she is willing to wait until the next available time slot.

The Interactive Multimedia Jukebox (IMJ) [5] applied this paradigm to network-based content distribution by architecting a Video-on-Demand (VoD) application that supports near-on-demand functionality. The IMJ provides a scalable, network-based video streaming service by batching client requests for movies, and scheduling distribution of content. At distribution time, scheduled content is streamed to the end-user via multicast. The schedule of content is available to all users, and all users can take advantage of the same distributions. In this way, a larger user population can be supported.

6.2.1 The AIS Model

While the IMJ supports only a single service, distribution of video, the jukebox-style batching paradigm can be applied to a number of Internet-based applications. We identify the behavior of the IMJ as that of an Active Information System (AIS). An AIS supports scalable user interaction by using one-to-many, push-based delivery to distribute new information when it becomes available. In the AIS, new information can include anything from realtime questions posed in an online classroom to new bids in an auction application.

Figure 6.1 illustrates the architectural components of an AIS. The functions of an AIS are as follows:

Static Content Distribution. The AIS model assumes a stable back-end information source such as a database. Users begin interacting with the system by issuing requests for information from that source. In a VoD system like the IMJ, this is the list of available movies. Requests may be straightforward such as requesting all database information, or may be more sophisticated. For example, a user may request only a subset of available information (e.g. all PG rated movies). The basis of information delivery in the AIS model is the one-to-many delivery of content. The challenge is to find ways to aggregate the many user requests without having to constantly transmit information that is

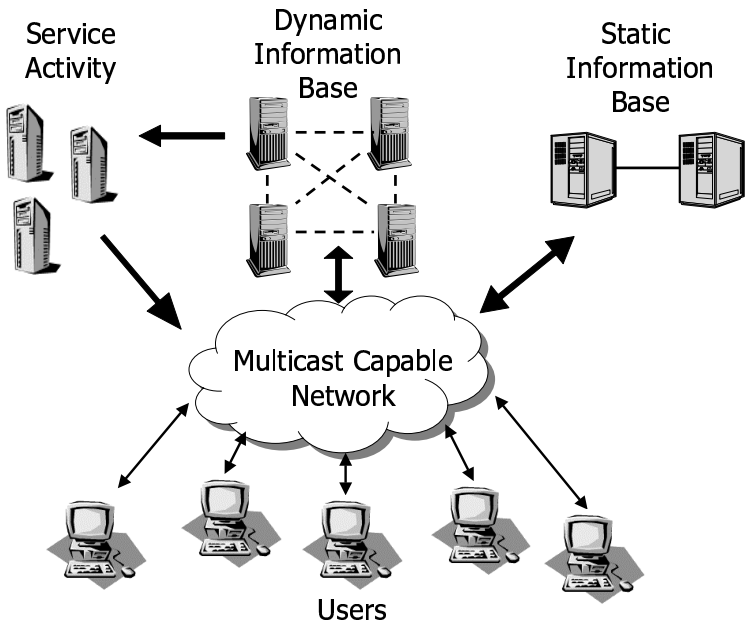

Figure 6.1: The architecture of a generic AIS.

not needed and without having to make users wait while other requests are batched. Given a large enough user base, this will not be a problem.

Input Processing. Input processing consists of accepting and processing requests from the user. In the IMJ, these requests may be to schedule playout of a particular video. Allowing the user to provide input to the application greatly increases the flexibility of the application itself. The goal of processing the request is to alter or add to the dynamic content distributed by the application. In most cases, the exact type of user requests and the processing required is application-specific. Ideally, the actions of one user should positively affect other users.

Dynamic Content Distribution. The AIS model is deemed "active" because user requests create dynamic changes in the application. In the IMJ, the dynamic component is the schedule of programs to be distributed. AIS-style applications are dependent upon user requests. When the state of the applica-

tion changes, the update must be propagated to all users. Unlike most traditional web-services, which expect users to check for new updates by re-loading, the AIS uses a push-based technique. The traditional pull-based approach is limiting in a number of ways. First, realtime applications suffer arbitrary delays based on how willing the user is to frequently check for updates. And second, additional load is placed on the system because users are constantly checking for updates. A large enough user population makes this kind of system wholly unscalable. In the AIS model, updates are sent to interested users of the system as *events*. This ensures that (1) updates are delivered efficiently without wasting network bandwidth and (2) users receive updates as soon as possible. Using multicast to deliver these events is an effective technique for achieving scalability.

Service Activity. The service provided by an AIS application may include anything from multimedia content distribution (as in the case of the IMJ) to the sale of an item in a realtime auction. This piece of the framework is intentionally left open such that we can support a variety of applications. However, as with the IMJ, the underlying goal is still scalability. Therefore, we would like to service as many users as possible by using a minimal allocation of resources. Again, in many applications, the use of multicast is appropriate to service all users with a single set of resources (e.g., one video stream). For some applications, like a chat room, the functionality can effectively be achieved using a completely distributed system. The real function of the AIS system is simply to coordinate users and build a community with a common interest.

6.2.2 Applying the AIS

The underlying goal of the AIS is to employ a batching paradigm to reduce the resource usage of resource-intensive applications. For a content-driven application in a peer network, especially a peer network of resource-constrained, pervasive computing devices, reducing resource usage is imperative. By batch-

ing download requests and distributing content to multiple peers in parallel, we can ease much of the burden placed on the peer acting as a server as well as the network. Additionally, by scheduling content to be distributed and distributing the schedule to all peers, we gain the advantage of having a local *hot list* catalogue. Users can consult the local schedule as a means to browse content available in the network.

Unfortunately, the current design of the AIS is targeted toward allowing application service providers to provide a service over the wired Internet. It relies on a centralized infrastructure to mediate communication and perform updates. This restriction makes the AIS model unsuitable for deployment in a peer-based network. In the remainder of this chapter, we describe and evaluate an architecture that applies the AIS model to peer networks. *Pixie* is a peer-based, jukebox-style architecture that supports efficient, scalable, content exchange in peer environments.

6.3 An Architecture for Request Aggregation and Batched Content Distribution in Peer Networks

Pixie applies the AIS model to reduce resource usage and improve data location and content delivery in peer networks. This section presents an overview of *Pixie* and discusses the *Pixie* architecture in more detail.

6.3.1 *Pixie* Overview

Pixie is an architecture to support one-to-many distribution of content in peer networks. The first goal of *Pixie* is to aggregate peer requests to download content and use intelligent, one-to-many content delivery (e.g., multicast) to enable a large number of peers to take advantage of the same distribution (see Figure 6.2). The second goal is to publish a schedule of content to be distributed to allow users to *browse* through the most popular subset of content available across the network. *Pixie* can be implemented on top of virtually any

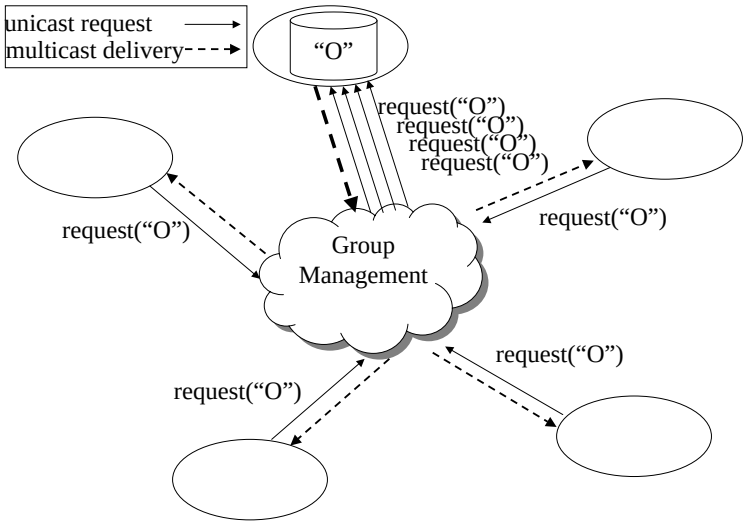

Figure 6.2: Overlapping requests are aggregated at the serving peer.

peer group management protocol. When a peer joins the network, it requests the *schedule*. The schedule contains information about content that will be distributed (e.g., Gone with the Wind), how the peer is to receive the content (i.e., the IP address of the multicast group), and when the distribution is scheduled to begin (e.g., 8pm GST). If a user is not interested in content already scheduled for distribution, the user may choose to search for and schedule new content. When a new distribution is scheduled, an *updateSchedule* message is *pushed* to all peers indicating the name of the content that will be distributed, how an interested peer can receive the content, and the scheduled distribution time. At distribution time, interested peers *tune in* to the distribution.

Using this model, peers are able to more rapidly and efficiently locate data of interest. Instead of requiring that the user perform an on-demand search for every piece of content, the schedule provides a new service by acting as a browsable hot list of available content within the network. Assuming that many users are interested in the same content, it is likely that a user will find the content he or he/she is interested in by looking at the schedule. Since the schedule is local, no on-demand search must be performed, thus easing the burden on the network.

By distributing content using one-to-many distribution, we provide additional scalability properties as well. Efficiency gains come from reducing the load on peers by aggregating requests and servicing multiple peers simultaneously. At the scheduled time, a sending peer distributes the information using one-to-many delivery. All interested peers simply *tune in* and receive the content. While network-layer multicast is the most efficient distribution mechanism, we also envision the use of application-layer proxies to reach peers that may not be multicast capable.

6.3.2 *Pixie* Architecture

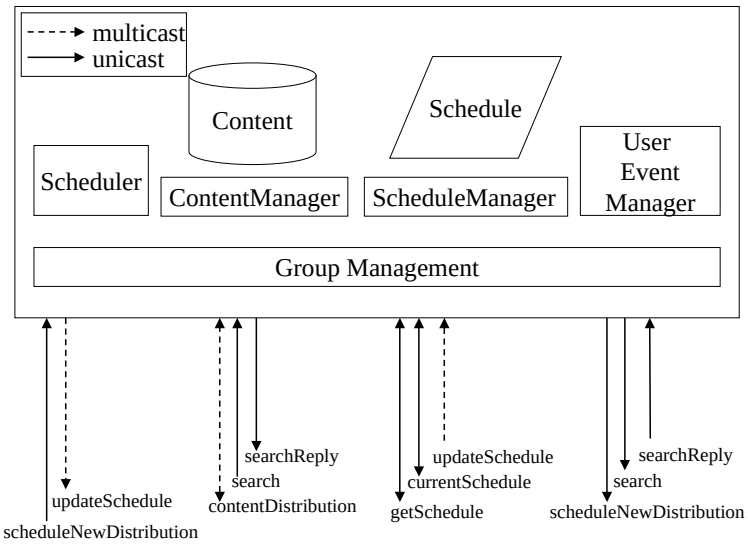

Figure 6.3: Architecture of a *Pixie* peer.

Figure 6.3 shows the general architecture of a *Pixie* peer. The *Pixie* components are implemented on top of a group management layer. We place no restrictions on the group management protocol. We envision anything from Napster-style centralized management to Gnutella-style distributed management to Chord-style distributed management. We discuss each component in more detail:

ScheduleManager. The ScheduleManager controls access to the schedule. The schedule contains information about which data are scheduled to be distributed, when distribution will begin, and where the data will be distributed. It is the equivalent of a TV guide that indicates which programs will be showing, at what time, and on which channel. Each peer retrieves a copy of the schedule when joining the network. Where the copy is found depends on the group management algorithm employed. In a Napster-style network, a getSchedule request will be routed to the centralized server. In a Gnutella-style network, a getSchedule request will be routed to a neighboring peer. We consider the schedule to be *best effort* in that we do not guarantee the peer will receive the latest version. However, if a peer receives a stale version and attempts to search for or schedule an already-scheduled piece of content, the peer serving the content will simply respond with an update indicating where and when the content is already scheduled. The ScheduleManager also receives and applies any updates to the schedule. Schedule updates contain relevant information about newly scheduled distributions (i.e., the content to be distributed, when the distribution will begin, and where the data will be distributed).

Scheduler. The Scheduler handles the scheduling of content distribution for a given peer. When the Scheduler receives a request for new content, it determines when the peer will have the resources available to fulfill the request. For example, if a peer can only support two simultaneous distributions and it is already distributing two streams, the new distribution must wait at least until one of the distributions has finished. The Scheduler may also apply more advanced scheduling algorithms such as delaying distribution in anticipation that more peers will be interested in the same content in the near future. Once the distribution has been scheduled, an *updateSchedule* message is generated and sent to all peers in the network. The most straightforward method of distributing the *updateSchedule* message is via multicast. However, a broadcast or gossiping scheme could be used propagate the message.

In a decentralized system, the Scheduler will exist on each peer and each peer will be responsible for scheduling distribution of its own content. However, a centralized implementation could also be employed. In a Napster-style system, a centralized authority would have information about each peer and could make scheduling decisions based upon that global information. This may be more efficient in terms of resource usage, however would require the presence of a centralized infrastructure.

ContentManager. The ContentManager controls access to the data stored on each peer. If a peer is not interested in scheduled content, it can search the network for other content. Search requests are routed through the network in a manner consistent with the underlying group management protocol. For example, using a Napster protocol, search requests would be routed to a centralized server while in a Gnutella protocol, requests would be routed to neighboring peers. When a peer receives a search request, it consults its content base and responds with information about content matching the search query. In a Gnutella-style network, the request would then be forwarded to neighboring peers.

The ContentManager is also responsible for distributing content. At the scheduled time, the ContentManager distributes the content, preferably using multicast. While network-layer multicast is the most efficient distribution mechanism, application-layer multicast distribution [26, 84] can be employed for peers without multicast connectivity. Finally, the ContentManager is responsible for receiving and storing content distributed by other peers.

UserEventManager. The UserEventManager processes events from the user and interacts with the user interface. It initiates searches for content specified by the user, requests new content be scheduled, and receives and displays search responses. This component is quite flexible and can be implemented to suit the preferred user interface.

6.4 Reliability and Fault Handling

The dynamic nature of typical peer networks like Gnutella makes reliable and fault tolerant peer-based applications challenging to implement. The problem is exacerbated in networks of pervasive computing devices. In these networks, unreliability may be caused by a user choosing to remove his/her device from the network (the typical case for wired peer networks), or may be caused by other factors such as the user walking out of range of the other users of the system. In this section, we address how *Pixie* handles faults and unreliability. We consider both reliability in terms of data delivery and fault handling in terms of failed serving peers.

6.4.1 Reliable Data Delivery

The first challenge we must address is to ensure that all data and control messages are reliably delivered to *Pixie* peers. Content must be delivered to all peers that are interested in receiving that content, and control messages (e.g., *updateSchedule* messages) must be delivered to every peer in the network. If a native multicast infrastructure is available, *Pixie* can use a basic reliable multicast distribution scheme for delivery of content and control messages. The primary advantage of using native multicast is the efficiency it provides. However, requiring that *Pixie* users have multicast connectivity is likely to be too restrictive. An alternative to reliable native multicast distribution is a reliable application-layer multicast scheme such as NICE [8]. Such a scheme is easier to deploy, and hence, can support a wider base of users.

Any straightforward reliable protocol run over either native or application-layer multicast requires receivers to join the distribution from the beginning. Moreover, if a serving peer fails, a new serving peer must start the distribution again from the beginning. To overcome this limitation, we propose the use of a digital fountain-style scheme [12]. Using a digital fountain scheme, the ContentManager distributes files that have been encoded using Tornado codes. The serving peer continuously distributes blocks of the encoded file until the

client peer has received a sufficient number of blocks to reconstruct the file. Since blocks may be received in any order, a client can join the distribution at any time and take advantage of the distribution in progress. Similarly, if N blocks are needed to reconstruct a file and a serving peer fails after a corresponding receiving peer has received N-X blocks, the receiving peer can join a new distribution and will only need to receive the remaining X blocks.

Using this scheme, a peer can potentially remain continuously occupied, distributing the same file. In the most extreme case, a serving peer distributing a file that requires N blocks to decode will receive a new request for the file after almost all blocks have been sent. In some cases, this behavior may be desirable. However, a peer that stores multiple pieces of popular content may need to perform some form of internal load balancing to ensure that it can service requests for multiple pieces of content.

6.4.2 Serving Peer Fault Handling

The most critical failure case occurs when a serving peer fails during or before its scheduled distribution. Failure can be the result of system failure, network failure, or a user can simply choose to take the peer offline, the so-called *freeriders* problem. We make the assumption that peers may go offline without any prior notification. Therefore, we must develop a strategy for rescheduling distributions that have not completed.

First, let us consider the basic (i.e., no failure) case when a user does not find an item of interest in the schedule and must search for a piece of content and schedule a distribution.

1. An interested peer searches for a piece of content.
2. Once the piece of content has been found, the requesting peer contacts the serving peer and requests a distribution be scheduled.
3. An *updateSchedule* message is propagated to all peers in the network.
4. At distribution time, the content is distributed to all interested peers.

In the event that the serving peer fails, there must be a strategy for detecting the failure and rescheduling the distribution. For the purposes of this work, we assume that a serving peer has failed if a receiving peer expects to receive content from the serving peer, but has not received data for some *timeout* period. The determination of this timeout period is dependent on many factors, including the underlying data distribution protocol. Moreover, more advanced schemes for detecting failures are also possible. However, more sophisticated schemes are beyond the scope of this work.

Once a fault is detected, the distribution must be rescheduled. To support our fault recovery algorithm, we must modify the basic search and schedule case as follows:

1. An interested peer searches for a piece of content.
2. Once the piece of content has been found, the requesting peer contacts the serving peer and requests a distribution be scheduled.
3. Along with the request, the requesting peer provides a list of all potential serving peers in the network that initially responded to the search request.
4. An *updateSchedule* message that includes the complete list of potential serving peers is propagated to all peers in the network.
5. At distribution time, the content is distributed to all interested peers.

Using the cached search results, the general fault recovery algorithm is as follows:

```
if the serving peer fails
  wait a random backoff period
  if the distribution has not resumed
    do
      select an alternate serving peer from the cached search
          results
    while the selected peer is not reachable
    request a new distribution from the selected peer on the
          same channel
    immediately, the new serving peer begins distribution to all
          peers waiting for the distribution
```

This strategy reschedules a failed distribution in an efficient manner. First, the random backoff period helps to avoid the case that multiple receiving peers simultaneously detect a fault and attempt to reschedule a distribution. The first peer to reach the end of the backoff period will reschedule the distribution and the remaining peers can take advantage of the rescheduled distribution. Moreover, using the same channel (e.g., multicast group address) for the rescheduled distribution avoids the overhead of sending out an additional *updateSchedule* message. Only those peers that are joined in the current distribution are affected. Finally, because the original search responses are cached in the schedule, there is no need to burden the network with a new search.

More advanced and efficient schemes for handling faults and ensuring reliability are also possible. For example, assuming the use of a digital fountain-style scheme, if multiple peers store the same content, they can be scheduled to distribute the same file simultaneously on the same channel. If neither serving peer fails, those receiving peers with a high bandwidth connection can potentially receive the file in half the time. Also, if one serving peer fails, another peer is already in the process of distributing the content. Moreover, if we used a more reliable group management protocol such as Chord, we could detect the failure of a peer and automatically reschedule a distribution at the infrastructure layer.

Using this strategy, the overhead and cost associated with unreliability are minimal. If a serving peer fails, the overhead incurred is one additional request to reschedule content. The cost associated with unreliability, either a serving peer failure, or the failure of an intermediate node in a distribution tree, is an increase in the amount of time it takes to download a piece of content. This increase in time is the result of the time to either reschedule a distribution, or reconfigure the distribution tree. However, using one-to-many distribution can greatly increase the number of peers that are ultimately serviced in a faulty network since multiple peers can be serviced simultaneously.

6.5 Evaluating Improved Data Location

In this section, we investigate the benefit of the schedule abstraction. Unlike other systems, *Pixie* allows users to browse an index of the most popular content available within the peer network. This property provides an enhanced user experience while incurring minimal overhead. This section focuses on the evaluation of the benefits of using Pixie. The overhead of using *Pixie* is explored in more detail in Section 6.6.

6.5.1 Metrics

We are interested in three metrics:

1. **Found** - Found describes the frequency that the user finds an item of interest in the schedule. This metric will allow us to conclude how useful the schedule is for users.

2. **Aggregated** - Aggregated describes the frequency that the user is interested in an item in the schedule *and* can take advantage of the scheduled distribution. This metric provides insight into how often multiple users are serviced by the same stream. We explore resource savings in more detail in Section 6.7.

3. **Schedule Size** - The size of the schedule will help us to determine the amount of repetition in the schedule and the manageability of the catalogue.

6.5.2 Setup

To evaluate these metrics, we have simulated the *schedule* portion of our architecture. When a request is made, we schedule the request according to the following algorithm:

```
if the requested item is scheduled
  record as FOUND
  if the distribution has not started
    record as AGGREGATED
  else if the distribution has started
    schedule on next available server according
      to shortest wait first
    send updateSchedule message to entire network
else
  send search message
  schedule at current time + 1 minute delay
  send updateSchedule message to entire network
```

For this set of experiments, we assume distribution is done through a basic reliable one-to-many distribution service such that users can only join the distribution from the beginning. This service can be either a network-layer multicast group or can be an application-layer distribution service. An item remains in the schedule from the time it is scheduled until it has been distributed.

This model does not entirely capture three cases. First, we do not capture the case when scheduling incurs an additional delay because a peer's resources are otherwise occupied. However, we claim that the model we use is, in fact, the most restrictive for the metrics we consider. Lower delay means that items remain in the schedule for a shorter period of time and are less likely to be *found*. Incurring an additional delay because a peer is distributing other content or is otherwise busy would only improve our results. Additionally, we assume that if it is possible to aggregate a request, aggregation occurs. If a client peer were to choose to schedule a new distribution rather than take advantage of an already-scheduled distribution, the aggregation and schedule size metrics would both be affected. However, the goal of this work is to promote request aggregation and investigate its benefits. Finally, we do not consider failures in the network. As described in Section 6.4, intelligent fault recovery schemes can minimize or eliminate the impact of faults on the metrics

we measure here.

We generate a trace of requests using a Zipf distribution [85]. Recent studies have shown this to be typical for current P2P systems[1]. For all experiments we use a catalogue of 400,000 items and run the experiment for a simulated period of 8 hours. We have also run simulations over a simulated period of 24 hours and observed similar results. To analyze the behavior of the system, we vary three main parameters:

1. **Load** - We look at the system behavior under different load conditions by varying the number of requests per second made across the network from 20-90. Values are taken from recent studies of the gnutella network [53, 65] that indicate that a single peer services or routes roughly 20 requests per second.

2. **Peer Characteristics** - We look at the behavior of the system based on different peer characteristics by varying the time it takes to distribute a single object. Table 6.1 details the values chosen. Small values for the distribution time can be the result of a fast connection or a small object. A large disparity between the min and max times is the result of highly varying peer characteristics. Each distribution time is chosen uniformly between the minimum and maximum times.

3. **Number of Servers** - We look at the behavior of the system as the number of serving peers available to serve each piece of content varies from 1 to 10. We assume that the number of available servers for each piece of content remains fixed through the experiment and that the same number of servers are available for every piece of content available in the network.

[1] http://www-2.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html

Min Time (sec)	Max Time (sec)	Description
1	500	Fast Connection High Variance
10	50	Fast Connection Low Variance
3800	4300	Mid Connection
10800	21600	Slow Connection High Variance
15120	16920	Slow Connection Low Variance
120	180	Typical of Current Usage

Table 6.1: Results of varying min/max distribution time.

6.5.3 Results

We present the results of four experiments. In the first experiment, we look at the *found* and *aggregated* metrics with respect to varying the load (number of requests per second) across the network. In the second experiment, we vary the number of servers and look at the *aggregated* metric. In the third experiment, we vary the peer characteristics in terms of the time to distribute a single item (the effect of either larger files or peers with slower connections) and again look at both the *found* and *aggregated* metrics. Finally, we look at the *schedule size* metric with respect to varied load, varied peer characteristics, and a varied number of servers. We follow with a discussion of the impact of these results.

Figures 6.4 and 6.5 illustrate how the number of found and aggregated items changes over time as the load (requests per second) varies. For this experiment, we fix the minimum and maximum distribution times at 1 and 500 seconds respectively. We observe that the greater the number of requests per second seen by the network, the greater the number of both *found* and *aggregated* items at each 1 minute interval. This is not surprising since a greater number of requests will mean that the schedule of distributions is

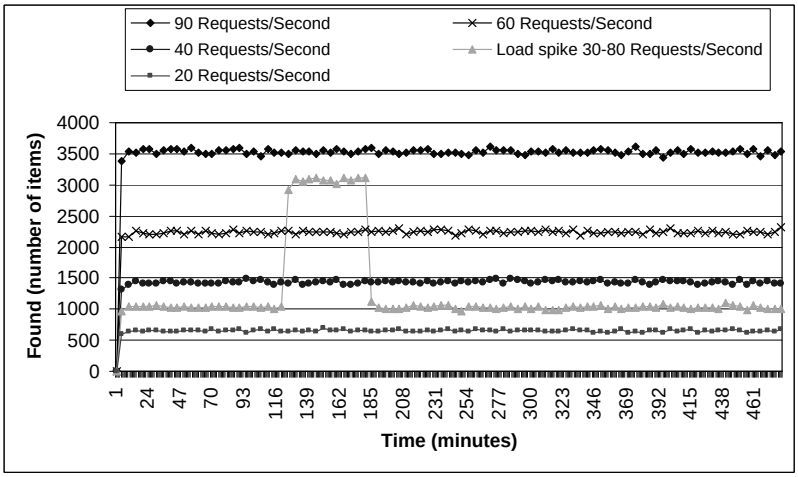

Figure 6.4: Number of items *found* over time for varied requests per second.

larger and there is a greater probability of overlap.

We also observe that, in all cases including the case when the load spikes from 30 to 80 requests per second from minute 120 to minute 180, the number of *found* and *aggregated* items stabilizes quickly and remains stable throughout the experiment. This property allows us to conclude that under varying load conditions, the system will remain stable.

Another interesting observation is that the *percentage* of requests that are found and/or aggregated remains relatively stable throughout the experiment. The percentage of found items ranges from 54.0% overall in the 20 requests per second case to 65.1% overall in the 90 requests per second case and the percentage of aggregated items ranges from 43.5% overall in the 20 requests per second case to 54.6% overall in the 90 requests per second case. Thus, we can extrapolate that even under varying load conditions, nearly the same percentage of requests will be found or aggregated overall.

Our final observation is that the difference between the number of *found* and *aggregated* items is relatively small. Thus, the case when an already-scheduled item must be scheduled again is relatively rare. Most requests for the same content can take advantage of an existing, scheduled distribution.

Figure 6.6 illustrates the number of items *aggregated* over time as the num-

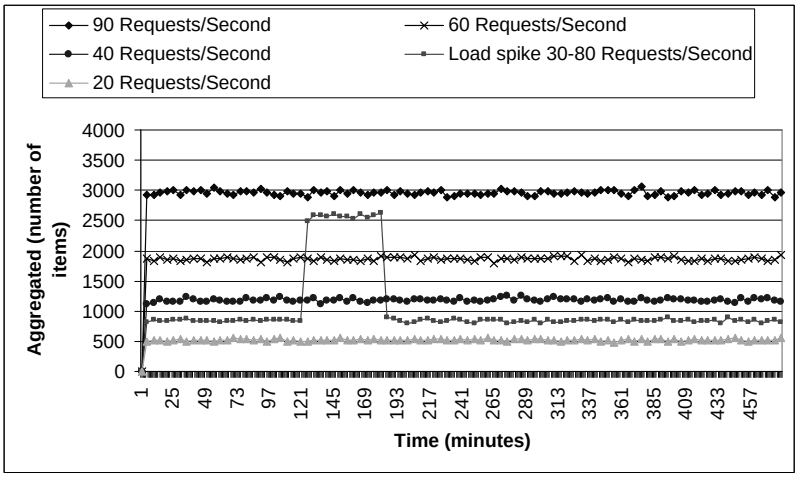

Figure 6.5: Number of items *aggregated* over time for varied requests per second.

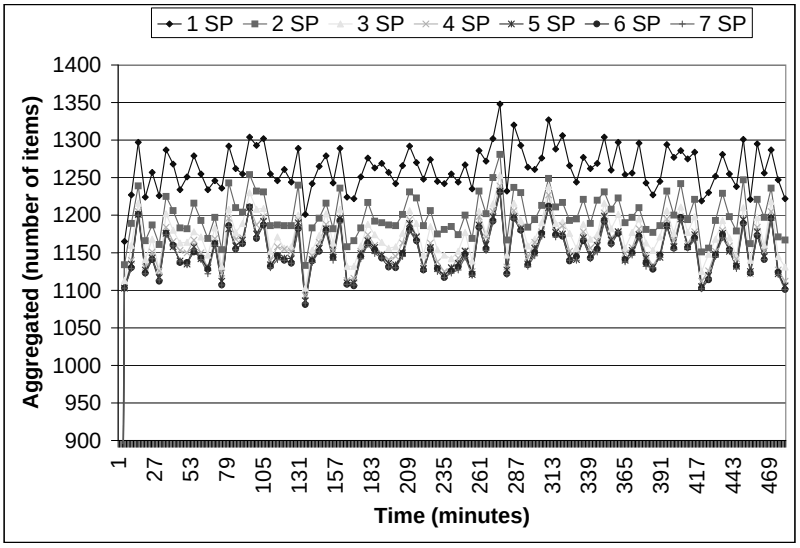

Figure 6.6: Number of items *aggregated* over time for varied serving peers.

ber of available serving peers (SP) varies. The *found* metric is almost unchanged for this experiment. Therefore, we omit the results. Our first observation of the figure is that the greater the number of serving peers available, the smaller the number of aggregated requests. In essence, with only one serving peer available, if a requesting peer cannot take advantage of an already scheduled distribution it must wait until that distribution is finished. However, by increasing the number of serving peers, there is an increase in the probability that a requesting peer can schedule a new distribution on an alternate, idle serving peer. This increases the parallelism and reduces aggregation. However, there is still a significant amount of aggregation across the network, and the level of aggregation reaches a minimum with only 6 serving peers available.

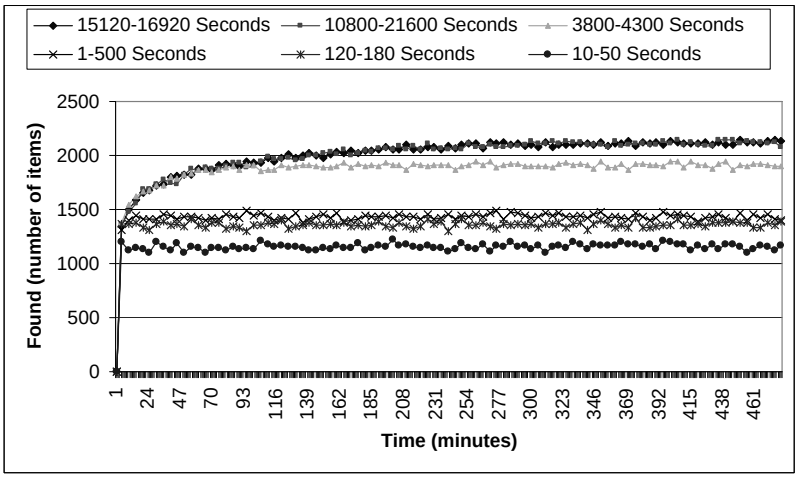

Figure 6.7: Number of items *found* over time for varied distribution times.

Figures 6.7 and 6.8 illustrate how the number of *found* and *aggregated* items changes over time for varied item distribution times. The item distribution time is the amount of time it takes to distribute a particular object. The greater the distribution time, the greater the number of *found* and *aggregated* items at each one minute time interval. The reason for this behavior is that items with longer distribution times will remain in the schedule longer. Hence, the schedule itself will be larger and the probability of finding an item

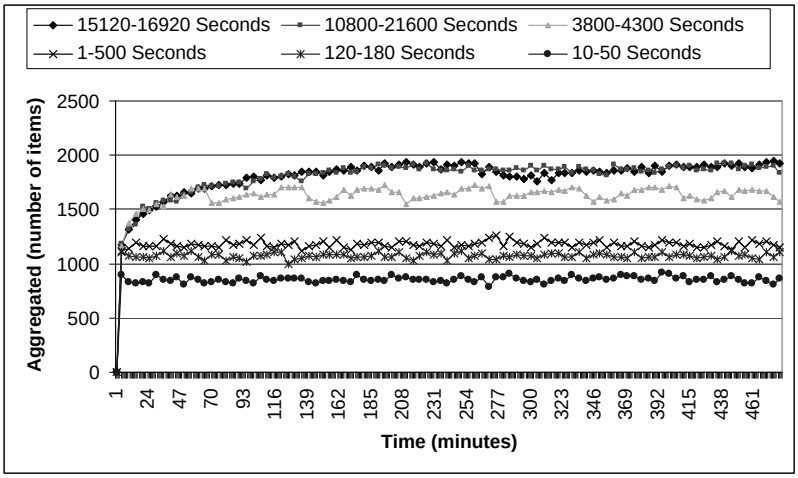

Figure 6.8: Number of items *aggregated* over time for varied distribution times.

in the schedule will be higher. Additionally, when items have longer distribution times, the system takes longer to stabilize. This is because no items are removed from the schedule until the initially scheduled items finish.

We also observe that faster distribution times result in fewer found and aggregated items overall. This is simply because when requests are processed faster, there is less opportunity to find a scheduled or executing distribution. Our results indicate that when downloads occur very quickly (10-50 seconds), the percentage of items found in the schedule is 48.1% and the percentage aggregated is 35.6%. This is still a substantial percentage and would still render our system useful.

Our final observation is that slower connections with low variance tend to be quite cyclic. This is largely because the low variance means that all requests initially scheduled are likely to finish at nearly the same time and new requests will be scheduled at that time. This behavior is less likely to occur in a system with varying load, or one in which the load gradually builds up to a stable point.

Figure 6.9 illustrates the total size of the schedule and number of distinct schedule items over time. We fix the number of requests per second at 40 and introduce a spike in load from 40 to 90 requests per second from minute 120

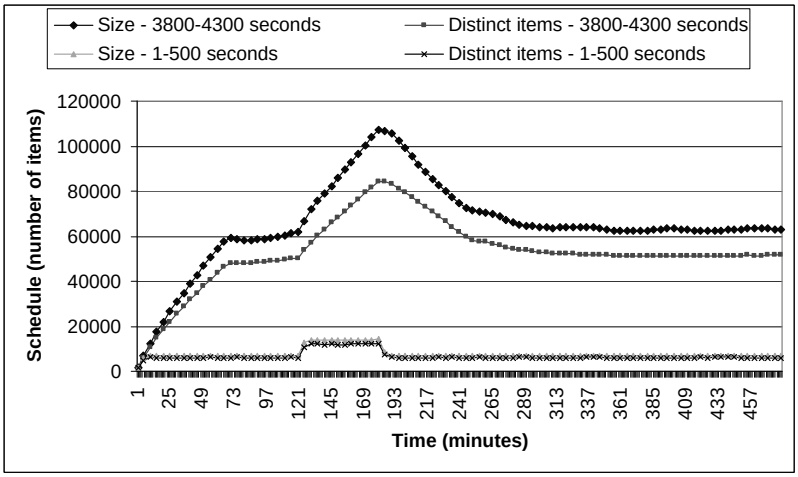

Figure 6.9: Total *size of schedule* and number of distinct items in schedule.

to minute 180. For the standard 1-500 second distribution time, the schedule size stabilizes quickly, recovers quickly from the load spike, and the size of the schedule is nearly identical to the number of distinct items. With distribution times from 3800-4300 seconds, we see that the schedule takes almost an hour to stabilize initially, does not completely stabilize during the load spike, and is large overall, over 60,000 items at its most stable point. The 6,000-7,000 item schedule for 1-500 second distributions is quite manageable. However, to deal with a larger schedule we might have to employ a solution such as caching only the *hottest* parts of the schedule and asking for other parts of it on demand.

Figure 6.10 illustrates the size of the schedule as the number of serving peers varies. A greater number of serving peers does lead to a slightly larger schedule. However, the total size of the schedule only increases by 200 to 300 items. Moreover, the maximum size is reached when only 7 serving peers are available. Therefore, increasing the number of serving peers has a very manageable impact on the schedule size metric.

6.5.4 Summary

Pixie introduces a new model for supporting content exchange. From a

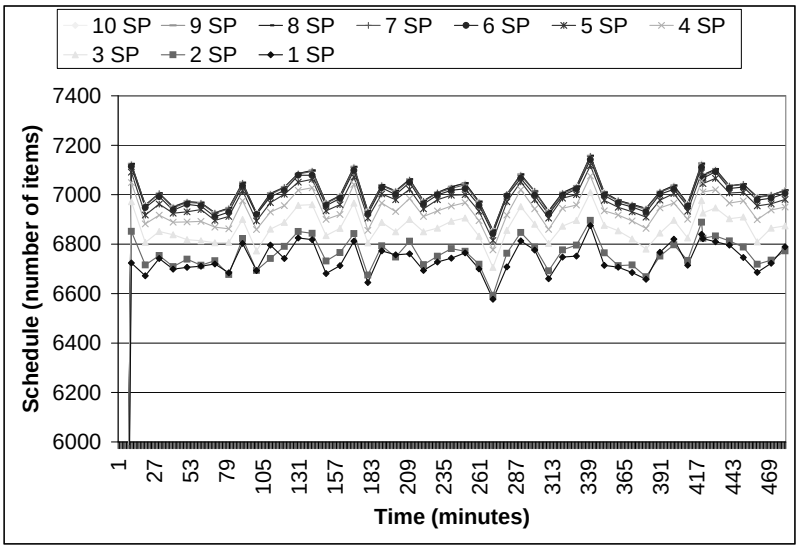

Figure 6.10: Total *size of schedule* over time for varied serving peers.

user's perspective, *Pixie* means that information exchange is no longer simply pull-based. Interesting information is actually *pushed* to the end user generating a browsable, local catalogue of the most popular content available in the network.

Our experiments and analysis demonstrate that the benefit of the schedule is affected by a number of factors. The more heavily loaded the network, the more absolute benefit the schedule provides. However, the relative benefit remains reasonably stable. Moreover, the object delivery time has a significant effect on the performance of *Pixie* search. The longer it takes to deliver an object, the more likely it is that a user can find an item of interest in the schedule. This same benefit could also be achieved by implementing a *timeout* period that indicates how long an item should remain in the schedule (possibly longer than the object delivery time). However, the tradeoff in either case is the size of the schedule. Finally, the most significant factor contributing to the metrics evaluated in this section is the Zipf request distribution. We compared the performance using a Zipf request distribution to the performance using a uniform request distribution and discovered that the *found* metric drops from

about 60% to about 3% and the *aggregated* metric drops from about 50% to about 1.25%. While *Pixie* may not be suitable for applications where users have uniformly distributed interests, the overlap in user interest observed in peer content exchange networks makes a push-based data location strategy useful.

6.6 Evaluating the Overhead of Schedule Maintenance

The previous section demonstrates the benefit of using *Pixie* for data location in peer networks. The schedule abstraction provides a number of benefits including reduction of the number of searches performed in the network. However, *Pixie* also incurs the overhead of *updateSchedule* messages which must be delivered to all peers in order to maintain the distributed schedule. This section examines the overhead of sending *updateSchedule* messages and the tradeoff between search and update messages in a variety of networks.

6.6.1 Metrics

To evaluate the benefit versus overhead tradeoff of *Pixie*, we are interested in two metrics:

1. **Number of Search Messages Processed** - Number of Search Messages Processed indicates how many search messages are processed at a single peer in the network. This metrics provides a quantification of the search overhead incurred by *Pixie* versus the search overhead incurred using a straightforward search scheme.

2. ***updateSchedule* Message Overhead** - *updateSchedule* Message Overhead quantifies the overhead of distributing *updateSchedule* messages throughout the network. This metric provides an idea of the cost associated with using the *Pixie* scheme.

6.6.2 Setup

This set of experiments is conducted using the same setup used for the experiments in the previous section. When a request is made, it is processed according to the same algorithm:

```
if the requested item is scheduled
  record as FOUND
  if the distribution has started
    schedule on next available server according
      to shortest wait first
    send updateSchedule message to entire network
else
  send search message
  schedule at current time + 1 minute delay
  send updateSchedule message to entire network
```

As in the previous set of experiments, we use a Zipf distribution [85] to model user behavior. Unless otherwise noted, we assume a network size of 15,000 peers, a catalogue of 400,000 items, object distribution time between 1 and 500 seconds, and the experiment is run for a simulated period of 8 hours. We vary three parameters:

1. **Load** - As in the previous experiment, we look at the system behavior under different load conditions by varying the number of requests per second made across the network from 20-90.

2. **Number of Servers** - Also, as in the previous experiment, we look at the behavior of the system as the number of serving peers available to serve each piece of content varies. We consider 1 to 7 serving peers.

3. **Network Size** - We also look at the system behavior as the network size (i.e., number of participating peers) varies. We look at a small network of 500 nodes and a large network of 50,000 nodes.

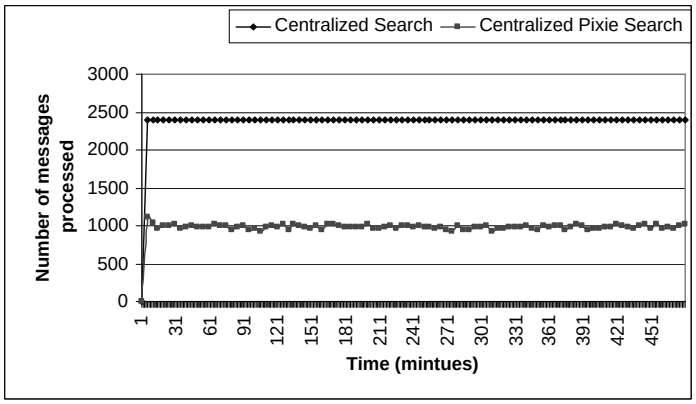

Figure 6.11: Number of search messages processed in a centralized scheme.

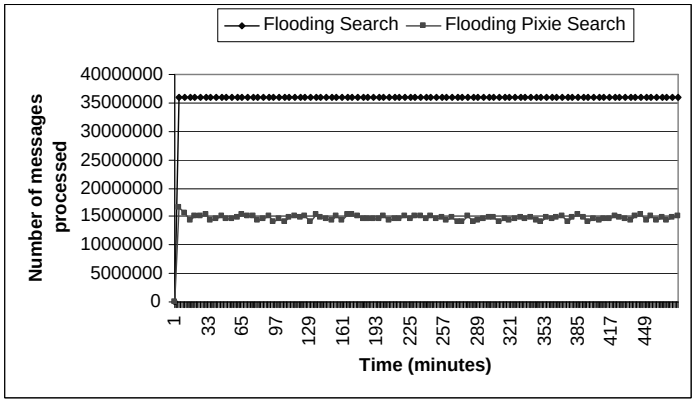

Figure 6.12: Number of search messages processed in a flooding scheme.

6.6.3 Results

To quantify the search savings in *Pixie*, Figures 6.11, 6.12, and 6.13, illustrate the number of search messages that are processed in *Pixie* versus three standard search schemes. In each case, we assume that *Pixie* runs over the corresponding group management scheme we compare against. For example, in Figure 6.11, we assume that *Pixie* runs on top of a centralized group management protocol, and compare against a straightforward centralized protocol such as Napster. We assume a network size of 15,000 nodes which is consistent with recent studies of the Gnutella network [65].

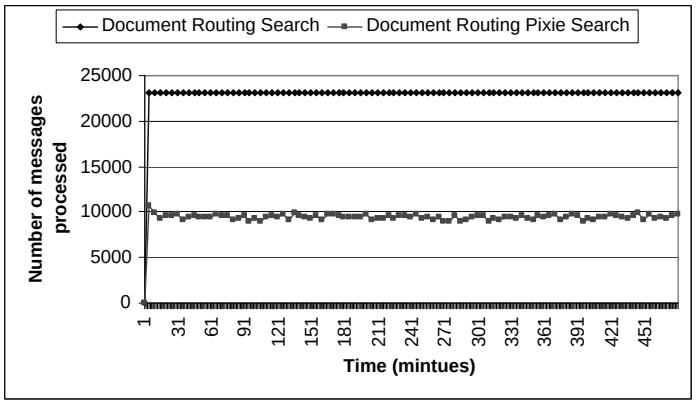

Figure 6.13: Number of search messages processed in a document routing scheme.

First, we compare *Pixie* to a Napster-style, **centralized** search scheme. In the centralized case, we assume that each search requires that one search message is sent and processed by the centralized entity. Next, we compare *Pixie* against a Gnutella-style, **flooding** search scheme where each search requires that a message be flood throughout the network and processed by every peer. Therefore, the cost of a single search is equal to the number of peers in the network. Finally, we compare *Pixie* against a **document routing** search scheme such as Chord. In this scheme, we assume that each search message is processed by $log\ N$ peers where N is the total number of peers in the network.

Our first observation is that *Pixie* reduces the total number of search messages processed by over half. In a centralized network, the savings is about 1,400 messages per minute. However, in a document routing network the savings is roughly 14,000 messages per minute, and in a flooding scheme, the savings is over 21,000,000 messages per minute. The disparity can be explained by the fact that each search scheme requires a different number of peers process each search message. Given our network size of 15,000 nodes, using *Pixie* instead of a flooding scheme saves approximately 24 messages per second per peer.

We also observe that the shapes of the curves for the three classes of search

scheme are nearly identical. This is the result of using the same trace data for each experiment. However, as previously observed, the scale of the results varies for each scheme. A flooding scheme generates significantly more traffic overall than a centralized scheme. Therefore, *Pixie* is more beneficial in the flooding case.

Our final observation is that all schemes are very stable. The stability of the centralized, flooding, and document routing schemes can be explained by the fact that we assume that the network is stable, and we assume that the same number of messages are processed for each search. In reality, factors such as network instability, user behavior, and network configuration could affect the stability of these schemes. However, we also observe that *Pixie* is nearly as stable as the schemes we compare against. Therefore, in a stable network, *Pixie* is likely to be well behaved and consistently perform well.

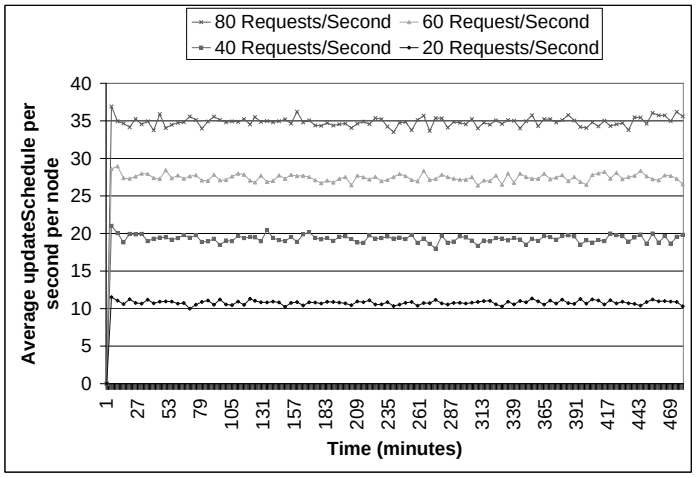

Figure 6.14: Number of *updateSchedule* messages processed at each node per minute.

While *Pixie* dramatically reduces the number of search messages processed in a P2P network, *Pixie* does incur the cost of additional *updateSchedule* messages that are flooded throughout the network for every scheduled distribution. In Figure 6.14, we examine the number of *updateSchedule* messages processed *at each node* in the network. We vary the load from 20 requests per second

to 80 requests per second and look at the average number of *updateSchedule* messages per second processed at each node for every one minute time interval.

Our first observation is that the more heavily loaded the network, the more overhead incurred. This follows from the fact that a more heavily loaded network will have more scheduled distributions and hence more updates. However, we notice that even in a heavily loaded network supporting 80 requests per second, fewer than half the requests result in an *updateSchedule* message. Moreover, in a moderately loaded network of 40 requests per second, each node sees fewer than 20 updates per second. We believe this to be a reasonable tradeoff for the search savings and enhanced user experience gained by using the *Pixie* scheduling scheme.

We further observe that the number of *updateSchedule* messages sent remains stable throughout the 480 minute run of the experiment. Therefore, we can conclude that in a stable network, *Pixie* will remain stable as well.

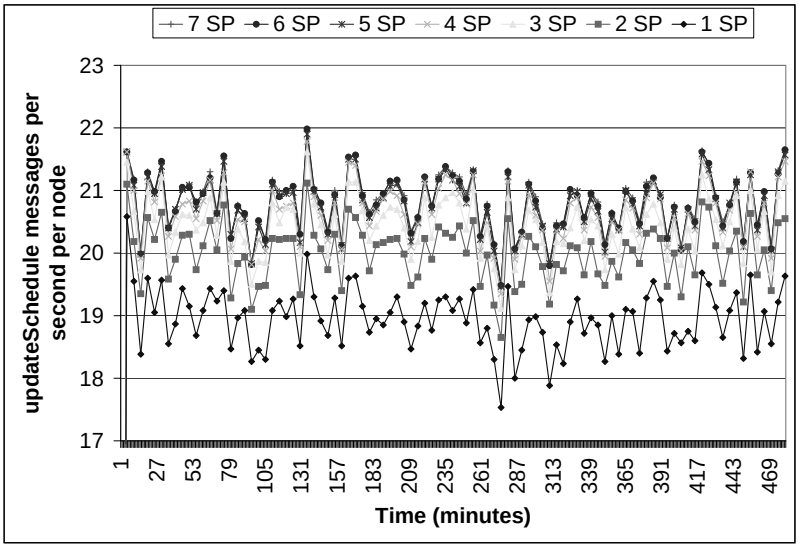

Figure 6.15: Number of *updateSchedule* messages processed at each node per minute.

In Figure 6.15, we fix the load at 40 requests per second and illustrate how the number of *updateSchedule* messages varies as the number of serving peers increases. As the number of serving peers increases the number of updates sent

and processed in the network increases as well. This is because more peers are available to distribute content, thus more distributions are scheduled and must be advertised. However, the average number of *updateSchedule* messages processed by each peer still remains reasonably low, less than 22 per second. Finally, the number of *updateSchedule* messages does not increase after the number of servers reaches 6.

Our final set of experiments further examines the tradeoff between the search savings gained by using *Pixie* and the overhead required by the *updateSchedule* messages. We present the number of search messages processed in a straightforward flooding search (FS) and the total number of search messages plus the number of *updateSchedule* messages (overhead) processed in a *Pixie* flooding search scheme (P+O). We vary the load from 20 requests per second to 90 requests per second and look at the total number of messages processed network wide during each one minute time interval. In Figure 6.16 we assume a small network of 5,000 nodes and in Figure 6.17 we assume a large network of 50,000 nodes. We omit the results of the same experiment run on a medium-sized network of 15,000 nodes for clarity of presentation. However, the results were as expected.

Again, we use the same trace data for each experiment and thus observe the same phenomenon we observed in Figures 6.11, 6.12, and 6.13. Our curves are consistent, however, the larger the network, the more absolute savings. We see about a ten fold increase in savings from roughly 1,200,000 messages per minute to roughly 12,000,000 messages per minute from the 5,000 node network to the 50,000 node network.

We further observe that the more heavily loaded the network, the greater the disparity between *Pixie* and the flooding search scheme. Therefore, the more heavily loaded the network, the more benefit we gain by using *Pixie*. Even in a lightly loaded network, *Pixie* requires roughly an equivalent number of total messages to the flooding scheme. Further, we contend that *updateSchedule* messages require less processing and therefore, even in a lightly

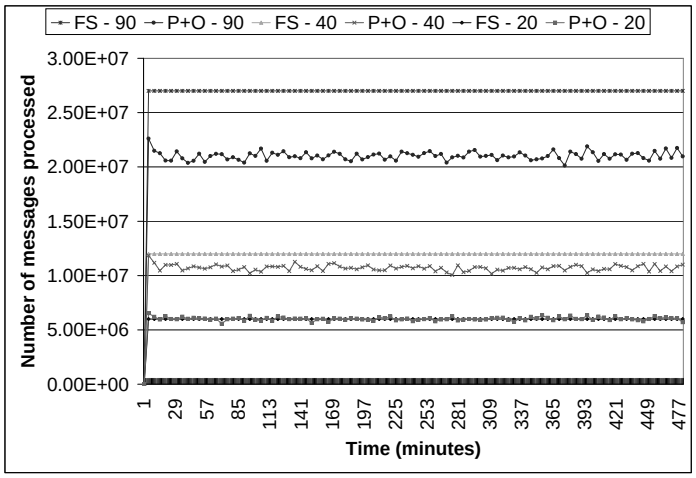

Figure 6.16: Search and search plus overhead in a small network.

loaded network, we benefit from using *Pixie*.

Figure 6.18 compares the 40 requests per second results illustrated in Figure 6.17 to the results of P+O as the number of servers increases. The number of *Pixie* search and overhead messages increases as the number of servers increases. However, the value of P+O reaches a maximum with 6 servers and remains well below the number of messages processed in the flooding search. Therefore, even with a large number of available servers for each piece of content, *Pixie* reduces the overhead of performing a flooding search.

6.6.4 Summary

Not surprisingly, there is a tradeoff between the search savings in *Pixie*, and the overhead incurred by the *updateSchedule* messages. *Pixie* reduces approximately 60% of searching overhead, but about 50% of requests incur the *updateSchedule* cost. Compared to a centralized network like Napster, the search and update overhead of *Pixie* is significant. However, *Pixie* outperforms a flooding search network like Gnutella. Moreover, compared to the cost of delivering content in a peer network, the cost of maintaining the schedule is quite manageable. Therefore, we conclude that *Pixie* search provides a new

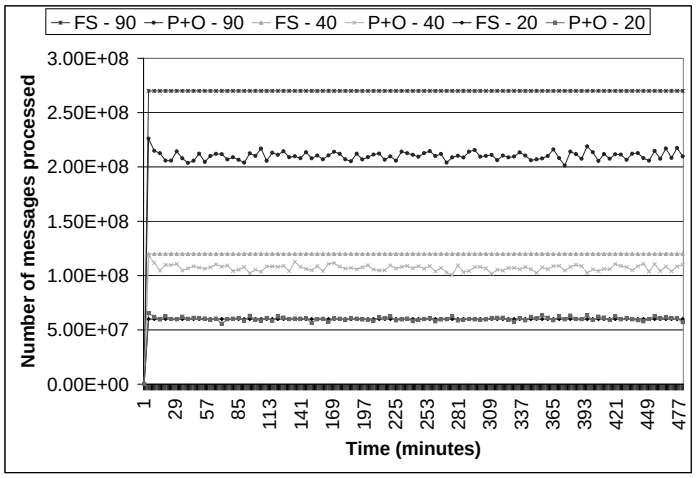

Figure 6.17: Search and search plus overhead in a large network.

and effective user service and incurs a manageable cost.

6.7 Evaluating Improved Resource Usage using Simulation

Recall that *Pixie* has two unique properties. The previous two sections have examined the benefit of using the *Pixie* model for improved data location in peer networks. This section evaluates the second property of *Pixie*; the use of aggregation and one-to-many data distribution protocols to improve the performance of content delivery. We assume the existence of an efficient one-to-many content delivery solution such as native or application-layer multicast. In this section, we examine an aggregation algorithm designed to batch overlapping requests at the serving peer such that they can be serviced simultaneously.

6.7.1 Metrics

We are interested in evaluating two primary metrics that demonstrate the tradeoffs of request aggregation:

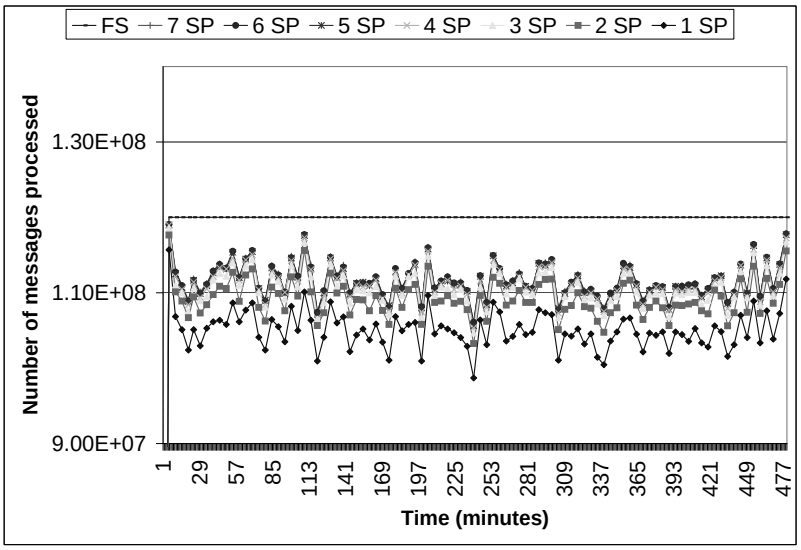

Figure 6.18: Search and search plus overhead as the number of serving peers varies.

1. **Wait Time** - In order to evaluate the benefit aggregation provides to the client, or requesting peer, we look at wait time. Wait time describes the amount of time the client must wait from the time it requests a piece of content until the distribution begins.

2. **Number Serviced per Distribution** - To evaluate the benefit aggregation provides to the serving peer, we look at the number of clients satisfied with each distribution. Using this metric, we can extrapolate on the resource savings of using an aggregation scheme.

6.7.2 Setup

To evaluate these metrics, we have simulated a set of peers that store the same piece of content. If a request is made for the piece of content, the schedule is consulted. If the content is scheduled on one of the peers in the set, but the distribution has not begun, the request is aggregated and will be serviced by the already-scheduled distribution. If the request is for content that is not scheduled or all scheduled distributions are already in progress,

the peer schedules a new distribution of the requested content according to a shortest wait first algorithm. The requesting peer selects the serving peer with the shortest wait time and schedules a new distribution of the content. We compare three scheduling schemes with respect to our target metrics.

We generate our traces using the parameters outlined the previous section. Unless otherwise noted, experiments are run for 480 minutes, item distribution time is between 1 and 500 seconds, 40 requests per second are made across the entire network, and the serving peer stores one piece of moderately popular content. Of the 40 requests per second made across the network, only those requests for the content stored on the serving peers will be processed. All scheduling is done first come first served with respect to the requests for content. We discuss each of the scheduling schemes evaluated in more detail:

- **FCFS** - This is the base case, *first come, first served*, no aggregation-no delay scheme. Distribution is one-to-one as is the case is current P2P systems and requests are serviced as soon as they are received.

- **AGG-<DELAY>** - This is an *aggregation-delay* scheme. Multicast distributions of requested content are scheduled with delay <DELAY>, specified in minutes.

- **DF-<DELAY>-<MAXDIST>** - This is a *digital fountain - delay - maximum distribution time* scheme. Digital fountain style distributions [12], as described in Section 6.4, are scheduled with delay <DELAY> as in the previous scheme. In addition, since the digital fountain scheme can cause starvation if requests for the same content continue to arrive, <MAXDIST> is a variable that specifies the maximum number of times a single distribution can be extended.

In addition to varying the scheduling scheme, we vary the following characteristics to evaluate system behavior:

1. **Load** - As in the previous sections, we look at behavior under varied load. We look at loads of 40 and 90 requests per second across the

system. Additionally, we look at the behavior of the system during a spike in load.

2. **Peer Characteristics** - Also, as in the previous sections we look at the system behavior based on peer characteristics with regard to the distribution time for each item. In addition, we vary the type of content stored on the serving peer between popular (many requests made for the content) and unpopular (few requests made for the content).

3. **Number of Servers** - Again, as in the previous sections, we vary the number of servers that store a particular piece of content. We consider values from 1 to 18.

6.7.3 Results

We present the results of five experiments. In the first experiment we look at our target metrics in the average case. The second experiment evaluates variations in load and peer characteristics by varying the number of requests per second made across the network as well as the popularity of the content stored on the serving peer. In the third experiment we take a slightly different look at the wait time metric while varying the load across the network. In the forth experiment, we look at both metrics with respect to the peer characteristics by increasing the time required for distributing a single piece of content. Finally, in the last experiment we look at both metrics as the number of serving peers varies.

Figure 6.19 plots the number of requests that experience each given wait time throughout the 480 minute experiment. In the FCFS case, many requests are made, queued, but not serviced within the 480 minute window. This is a common occurrence and illustrates the instability of the system using a FCFS scheme. Unserviced requests will remain in the queue of waiting requests at the end of the 480 minutes and we do not report on them here. We have truncated the FCFS data for presentation, but what happens in the FCFS case is that most of the serviced requests are issued in the first few timesteps.

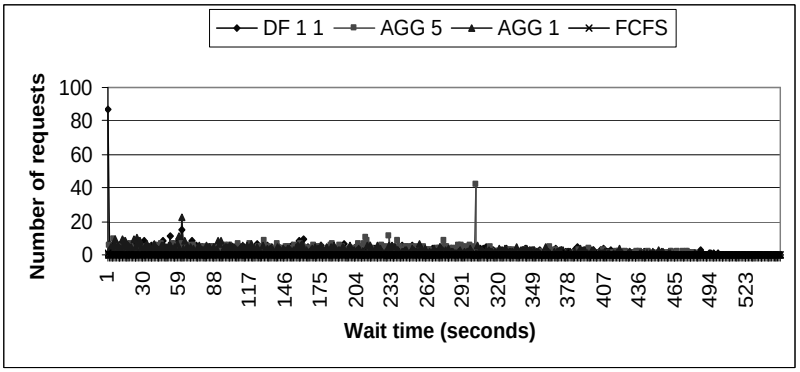

Figure 6.19: Number of requests experiencing each wait time at 40 requests per second.

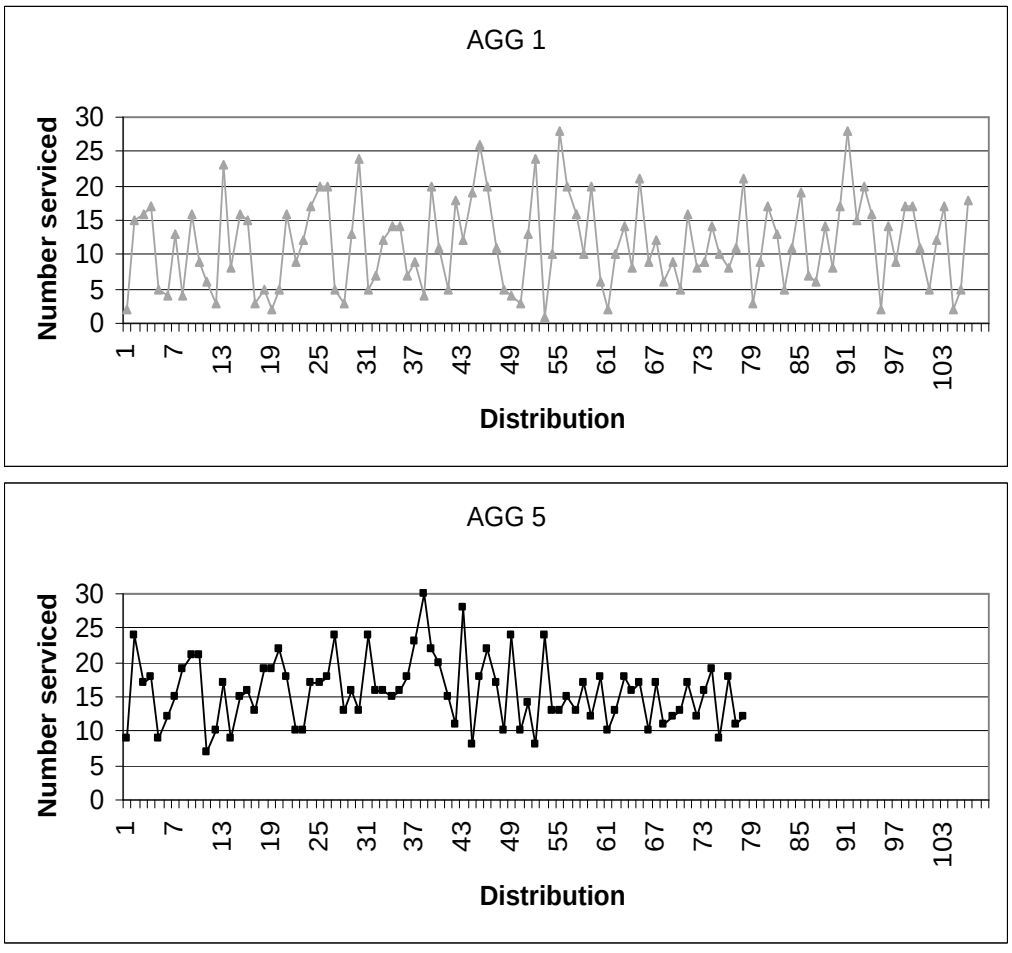

Figure 6.20: Number of requests serviced with each distribution at 40 requests per second.

Because they are serviced sequentially, each request waits from the beginning of the experiment until the time it is serviced and the time waited increases linearly for each serviced request. In fact, the final request serviced waits for 25,487 seconds.

While the FCFS wait times increase linearly, in all aggregation schemes compared, the wait time remains relatively constant. Using aggregation, no request ever waits for greater than 500 seconds because, in the worst case, a request will be issued just as a distribution is starting, hence the request will have to wait the duration of the distribution. This worst case would be affected if the serving peer stored more than one piece of content. In the worst case, a peer storing N pieces of content would schedule them sequentially, *1, 2,...,N*. If a request for *1* was issued right after the distribution began, the request would have to wait $\sum_{i=1}^{N} distribution\_time_i$ seconds until *1* was scheduled again. A peer could potentially distribute multiple pieces of content simultaneously, but the time to complete each distribution is still restricted by the peer's outgoing bandwidth. Additionally, we suggest that by enabling users to browse a schedule of content, requests are likely to be influenced by content already scheduled.

Another observation of Figure 6.19 is that the spikes at wait times 0, 61, and 301 indicate that the largest number of requests wait for the amount of time specified by the delay of the aggregation scheme. This is because the system is somewhat lightly loaded and relatively few requests arrive between the time that a distribution is scheduled and when it begins. While a longer delay implies a longer average wait time, the tradeoff is that a longer delay scheme utilizes fewer resources at the serving peer.

Finally, the difference between straightforward aggregation and a digital fountain-style aggregation scheme is minimal. This is primarily because we delay any new distributions by 1 minute and because we restrict the number of times the distribution can be extended to 1. In fact, since our serving peer in this experiment stores only one piece of data, in an unrestricted digital

fountain scenario we would achieve 0 wait time for all requests. If a peer was stable, likely to remain available, and stored only 1 or a few pieces of popular content, using an unrestricted digital fountain scheme would be the best choice.

Figure 6.20 illustrates the number of peers serviced for each distribution scheduled during the 480 minute run using the same parameters used for the experiment shown in Figure 6.19. We omit the results of the DF 1 1 scheme for presentation since the results were similar to the AGG 1 scheme. We also omit the results of FCFS because it always services one request.

We notice that the aggregation schemes manage to service up to 30 requests per distribution. We also notice that the AGG 5 scheme never services less than 7 requests per stream while the AGG 1 scheme often services fewer. Because AGG 5 consistently services more requests than the lower delay scheme, fewer distributions are required. This is simply because more requests are aggregated prior to the beginning of a given distribution. What this implies is that there is a tradeoff between the resources used at the serving peer and the wait time experienced by the client. By incurring an average wait time penalty of 33 seconds with the AGG 5 scheme, we gain roughly a 25% resource savings at the serving peer. Our final observation is that by using aggregation we gain an advantage in terms of disk space required across the network. For the FCFS scheme to achieve the same performance of the AGG 5 scheme, content would have to be replicated up to 30 times throughout the network.

Figures 6.21 and 6.22 illustrate the number of requests experiencing each wait time and the number of requests serviced per distribution respectively for 40 and 90 requests per second and for the 40 requests per second case when more popular content is stored at the serving peer. For the aggregation scheme, increasing the number of requests per second or the popularity of the content has a similar effect. Varying these parameters does not increase the worst case wait time experienced for each request, but rather simply increases the number of requests that experience each given wait time. Based on our

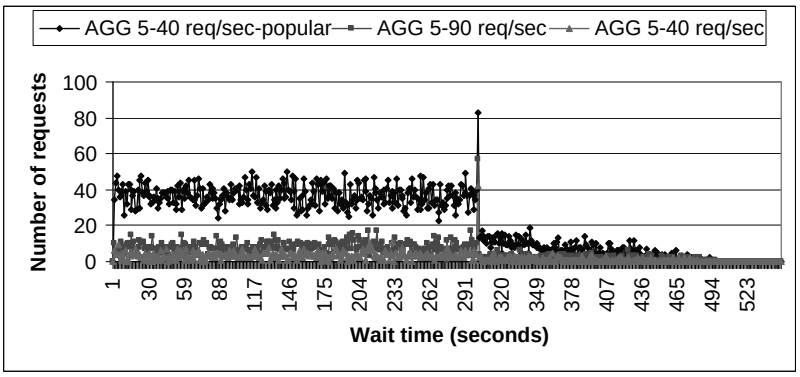

Figure 6.21: Comparison of the number of requests experiencing each wait time at 40 and 90 requests per second.

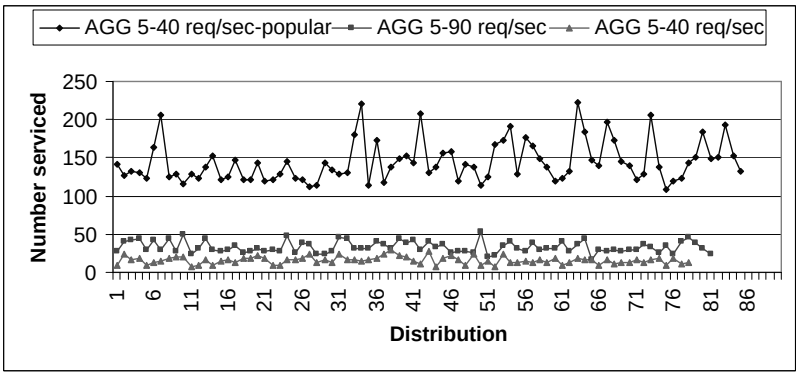

Figure 6.22: Comparison of the number of requests serviced with each distribution at 40 and 90 requests per second.

evaluation technique, the results of the FCFS case would not change greatly between the 40 and 90 request per second cases, or when the content becomes more popular, because all requests serviced would be issued near the beginning of the experiment. What would change is that the queue of waiting requests would be much larger, though we do not evaluate this metric here.

Additionally, as more requests are issued we can see a clearer pattern. There is an even distribution of wait time from 0 to the amount of delay used in the aggregation scheme. This is largely because requests arrive at a constant rate and are queued until distribution begins. The few requests that experience a wait time greater than the delay are those that arrive while a distribution is in progress and must the remainder of the current distribution.

In Figure 6.22 we observe that the number of requests serviced per distribution becomes much larger when the load is heavier. Again, increasing the number of requests per second and increasing the popularity of the stored content has a similar effect. Over 12,200 requests are satisfied using and AGG 5 dwarfing the less than 120 requests that can be satisfied in a FCFS case.

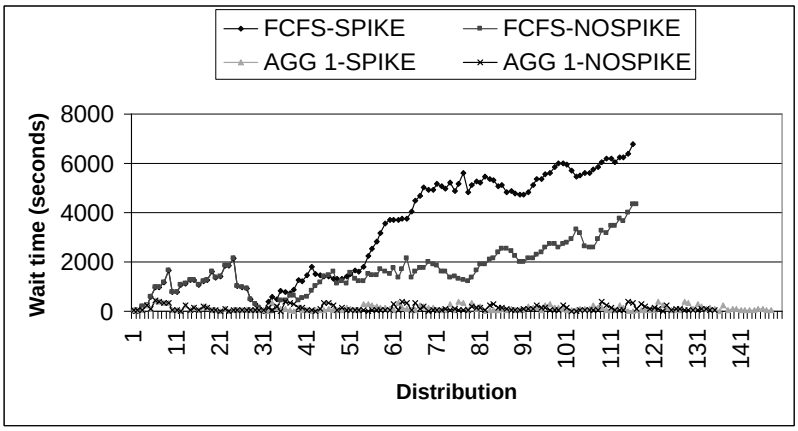

Figure 6.23: Wait time for each request serviced with load surge.

In Figure 6.23 we store an unpopular piece of content on the serving peer and demonstrate a spike in load from 40 to 90 requests per second. The figure illustrates the wait time experienced under these conditions. While

wait time with the FCFS scheme continues to grow, even without the load spike, wait time using the aggregation scheme remains stable over all requests. Such behavior is especially important when a new, popular piece of content is introduced into the peer network. In the best case FCFS scheme, distributing a single piece of content throughout the entire network would be logarithmic with respect to the number of peers. Using aggregation, the same distribution occurs in constant time.

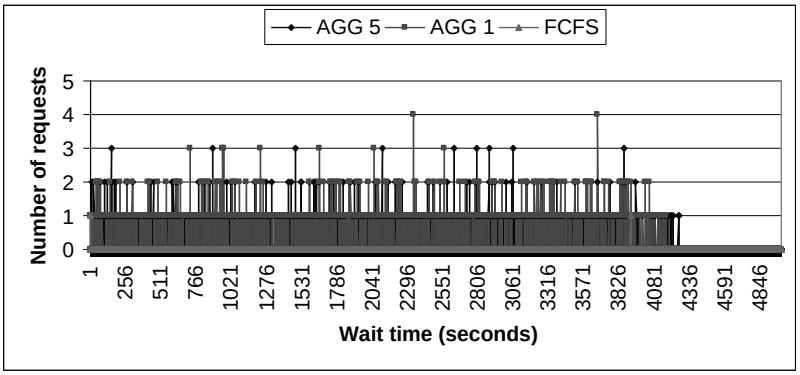

Figure 6.24: Number of requests experiencing each wait time for content with 3800-4300 second distribution time.

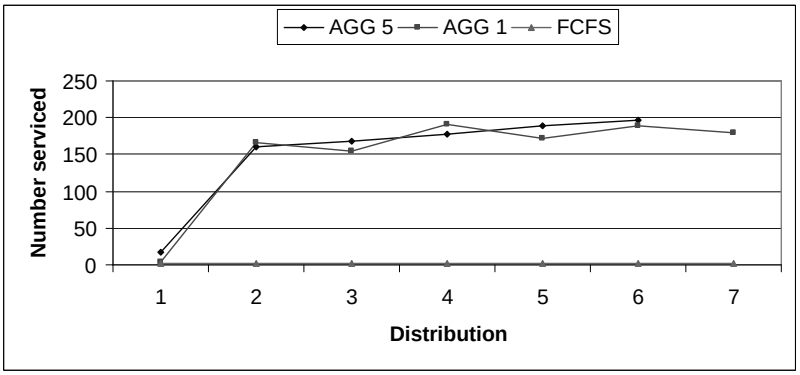

Figure 6.25: Number of requests serviced with each distribution for content with 3800-4300 second distribution time.

Figures 6.24 and 6.25 illustrate the effect of extending the amount of time a single distribution takes to complete. This would be the result of distributing

larger files and/or using a slower connection. What we notice is that, overall, wait time for the aggregation schemes increases to be, at most, nearly the length of the distribution. This is simply because requests made when a distribution is already in progress must wait until the distribution finishes. With FCFS, the behavior is nearly the same as in previous cases with the maximum wait being nearly the length of the experiment (480 minutes). Looking at Figure 6.25, we notice that in the FCFS case, only a total of 7 requests are serviced while over 150 requests are serviced with each distribution in the other schemes. The disk space savings *Pixie* achieves becomes clearer in this case. Assuming files of 5MBytes, a conservative estimate for for content such as video, we can achieve a savings of nearly 1GByte. In a wireless and/or small device environment where bandwidth and disk space are scarce and distribution can be lengthy, an aggregation scheme provides a clear benefit over a straightforward FCFS technique.

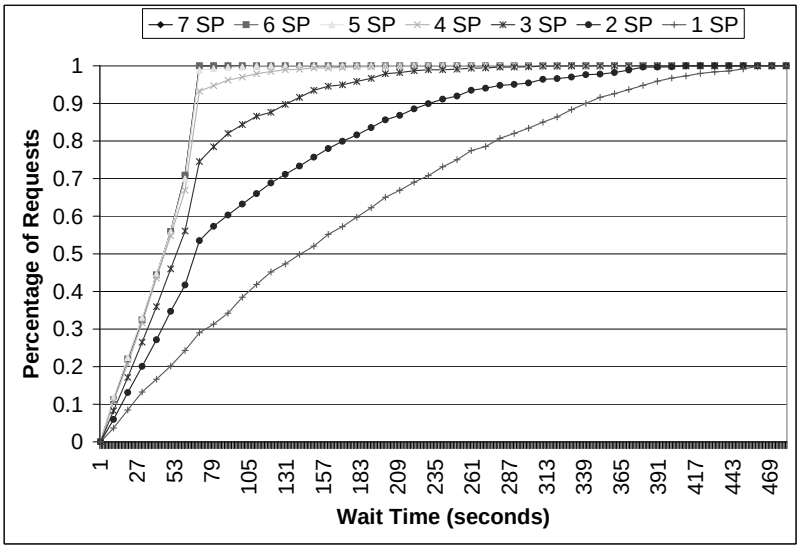

Figure 6.26: Cumulative distribution of wait times using AGG 1.

Figures 6.26, 6.27, and 6.28 illustrate the reduction in client peer wait time as the number of serving peers available to serve the requested item increases. Figure 6.26 illustrates the cumulative distribution of wait time incurred using

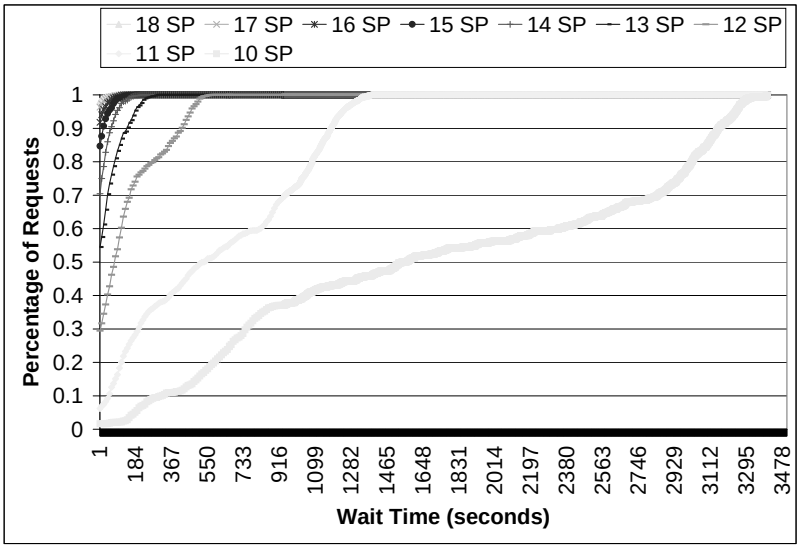

Figure 6.27: Cumulative distribution of wait times using FCFS.

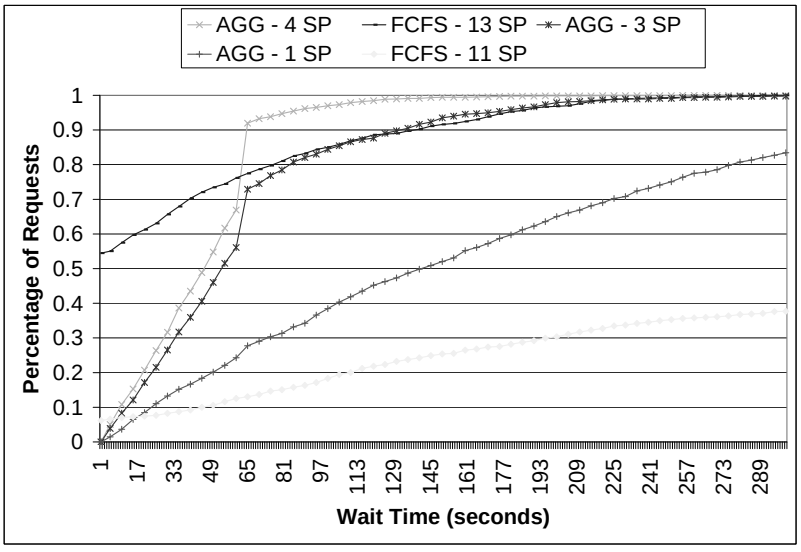

Figure 6.28: Cumulative distribution of wait times using AGG 1 and FCFS schemes.

the AGG 1 scheduling scheme. Our first observation is that as the number of serving peers increases, the cumulative wait time decreases. This is not surprising since a requesting peer is more likely to find a non-occupied serving peer to schedule a distribution rather than waiting for a currently occupied peer to finish a distribution in progress. Additionally, the cumulative wait time reaches a minimum when six peers are available to serve the same piece of content. This indicates that no more than six parallel requests are unable to be aggregated at any given time. However, this result would change if a peer could take advantage of an already-scheduled scheme, but chose to schedule a new distribution instead.

We also observe that, even in the best case, some requests still experience up to 60 seconds of wait time. This is the result of the 60 second scheduling delay used in the aggregation algorithm. The requesting peers that initiate the scheduling of a new distribution will always wait for at least 60 seconds.

In contrast, Figure 6.27 illustrates the cumulative distribution of wait time incurred using a FCFS scheduling scheme. In the FCFS case, the wait time will theoretically go to zero when the maximum amount of parallelism is achieved. But, even with 12 available peers, the cumulative wait time is greater than the cumulative wait time for one peer using the AGG 1 scheme.

To further compare the FCFS and AGG 1 schemes, Figure 6.28 illustrates selected results from the two previous figures. AGG 1 with 1 server performs considerably better than FCFS with 11 servers. However, AGG 1 with 4 servers achieves nearly the minimal wait time for the AGG 1 scheme, and is outperformed by FCFS with 13 servers about 75% of the time. To achieve a wait time comparable to AGG 1 in any case, the FCFS case requires at least 13 peers.

Recall, the tradeoff with respect to wait time is the level of aggregation achieved at the serving peer. Figure 6.29 illustrates the cumulative distribution of the number of requests serviced with each distribution (level of aggregation) as the number of servers increases. Similar to the wait time metric, with 6

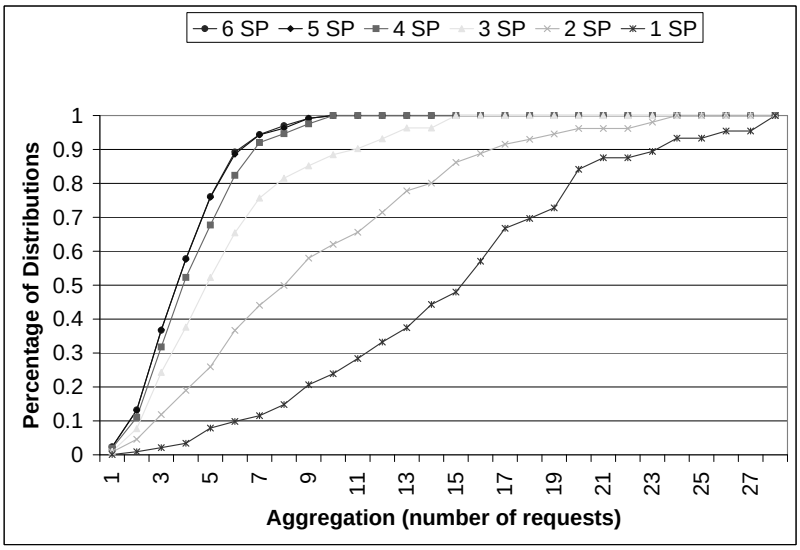

Figure 6.29: Cumulative distribution of aggregation per distribution.

serving peers the level of aggregation is minimized as well. Again, this indicates that the maximum level of parallelism is achieved with 6 available serving peers. In the case of 6 serving peers, about 50% of distributions aggregate fewer than 4 requesting clients. While a smaller level of aggregation can still be useful in terms of reducing the resources used at the serving peer, clearly aggregation is most beneficial when only a small number of serving peers are available.

Finally, Figure 6.30 compares the total number of requests serviced by the AGG 1 and FCFS schemes at the end of the 8 hour simulation. While the AGG 1 scheme always services all requests, the FCFS scheme requires at least 12 serving peers to match the performance. Especially in a network where peers frequently join and leave, aggregating requests and batching content delivery can have a positive impact on overall performance.

6.7.4 Summary

The aggregation algorithm employed by *Pixie* is highly effective. The level of batching achieved is significant and can yield a large resource savings in a

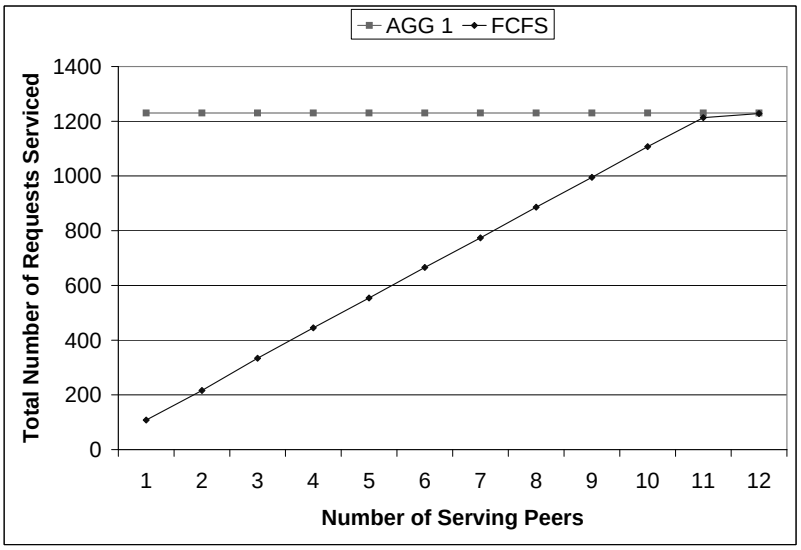

Figure 6.30: Total number of requests serviced as the number of serving peers varies.

peer content exchange network. Moreover, the tradeoff with respect to client peer wait time is minimal. As previously discussed in the chapter, load and object delivery time play a crucial role in the level of batching *Pixie* is able to achieve. However, the most significant factor is still the distribution of user requests. *Pixie* is effective primarily because of the significant overlap in user interest seen in peer networks. As we have demonstrated, by leveraging that property, we can provide new and more efficient peer content exchange services.

6.8 Discussion

Supporting content exchange for content-driven applications, such as P2P file sharing, in next-generation environments will be greatly more challenging than supporting the same applications in the traditional wired Internet. In contrast to desktop computers, devices such as PDAs have limited resources, such as bandwidth. Moreover, users of pervasive computing devices can be mobile making user participation highly transient. These properties make

traditional protocols for searching and downloading inappropriate. Protocols to support content exchange in next-generation environments must reduce resource usage while accounting for the dynamic nature of the environment.

Pixie provides a more efficient solution for search and content delivery in peer networks. The *Pixie* search protocol can reduce the bandwidth and processing resources necessary to locate content while incurring minimal and manageable overhead. Additionally, by batching requests and using one-to-many distribution, *Pixie* provides more efficient and faster delivery of content. As content-driven applications evolve to include exchange of content between mobile users carrying small, resource-constrained devices, *Pixie* can be used to solve many of the challenges inherent in the pervasive computing environment.

Chapter 7

Concluding Remarks

The underlying goal of this dissertation has been to develop and evaluate techniques that can be used to support current content-driven applications, such as web surfing and peer file sharing, in the next-generation computing environment where computing devices may have severe resource limitations. We have introduced three novel techniques to meet this goal:

Non-traditional Content Access. The MakerFactory realizes a model for automatically generating multimodal interface components from a schema definition. The feasibility of this model is demonstrated by the implementation of AXL. Using AXL, we have been able to automatically generate usable speech-based interfaces that provide access to XML content. The MakerFactory model and AXL implementation demonstrate that we can provide access to information, such as web content, using devices with constrained display capabilities.

Resource-Aware Content Management. Our architecture for resource-aware content management proposes a new model for integrating the dimension of resource availability into a content management scheme. Focusing on battery lifetime as a primary constrained resource, our experimental evaluation has shown that the lifetime of a collection of devices can be greatly extended by monitoring energy levels and migrating content accordingly. These results demonstrate that it is feasible to use a collection of devices in a cooperative

manner to overcome the resource constraints of a single device and prolong content-based services.

Efficient Content Location and Distribution. *Pixie* introduces an architecture for providing new and more resource-efficient services in peer networks. *Pixie* integrates existing communication and content delivery techniques which, we have demonstrated, can reduce searching overhead and improve serving peer resource usage while minimizing client wait time. Using the *Pixie* model, content exchange is more viable in large-scale environments where resources are scarce.

As next-generation computing environments become more commonplace, current applications will evolve and new applications will emerge to take advantage of new functionality. Factors such as increased user mobility, decreased device size, and ubiquitous deployment will enable new applications such as large-scale environmental sensing. Similarly, existing applications such as digital classrooms, web access, and peer file sharing will change to leverage available functionality. However, a fundamental challenge that we must address in order to achieve this vision and enable these kinds of applications is to overcome the inherent resource constraints of the future computing environment. This dissertation proposes three techniques to address the resource constraints that deter the fundamental functionality that these kinds of applications require; content access, management, and distribution. By using these techniques, we can support a wide range of advanced, content-driven applications in the next-generation computing environment.

Bibliography

[1] G. Abowd. Classroom 2000: An experiment with the instrumentation of a living educational environment. *IBM Systems Journal*, 38(4), 1999.

[2] M. Abrams, C. Phanouriou, A. Batongbacal, S. Williams, and J. Shuster. UIML: an appliance-independent XML user interface language. In *Proceedings of the eighth international conference on the World Wide Web*, Toronto, Canada, May 1999.

[3] E. Adar and B. Huberman. Free riding on gnutella. *First Monday*, 5(10), October 2000.

[4] M. Albers. Auditory cues for browsing, surfing, and navigating. In *Proceedings of ICAD 1996*, pages 85–90, Santa Fe, NM, USA, 1996.

[5] K. Almeroth and M. Ammar. The interactive multimedia jukebox (imj): A new paradigm for the on-demand delivery of audio/video. In *WWW7*, Brisbane, Australia, April 1998.

[6] Y. Amir, A. Peterson, and D. Shaw. Seamlessly selecting the best copy from internet-wide replicated web servers. In *International Symposium on Distributed Computing*, Andros, Greece, September 1998.

[7] B. Arons. Hyperspeech: Navigating in speech-only hypermedia. In *Proceedings of Third Annual ACM Conference on Hypertext*, pages 133–146, San Antonio, TX, USA, December 1991.

[8] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Sigcomm*, Pittsburgh, PA, August 2002.

[9] S. Barrass. *Auditory Information Design*. PhD thesis, Australian National University, 1997.

[10] E. Bergman and E. Johnson. Towards accessible human-computer interaction. *Advances in Human-Computer Interaction*, 5, 1995.

[11] S. Brewster. *Providing a Structured Method for Integrating Non-Speech Audio into Human-Computer Interfaces*. PhD thesis, University of York, York, Great Britain, August 1994.

[12] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Sigcomm*, pages 56–67, Vancouver, British Columbia, September 1998.

[13] A. Chankhunthod, P. Danzing, C. Neerdaels, M. Schwartz, and K. Worrell. A hierarchical internet object cache. In *Usenix Annual Technical Conference*, San Diego, CA, USA, January 1996.

[14] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, USA, July 2000.

[15] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *SOSP 2001*, Banff, Canada, October 2001.

[16] C. Ellis. The case for higher-level power management. In *HotOS*, Rio Rico, AZ, USA, March 1999.

[17] K. Farkas, J. Flinn, G. Back, D. Grunwald, and J. Anderson. Quantifying the energy consumption of a pocket computer and a java virtual machine. In *Sigmetrics 00*, pages 252–263, Santa Clara, CA, USA, June 2000.

[18] L. Feeney and M Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *INFOCOM 01*, pages 1548–1557, Anchorage, Alaska, USA, April 2001.

[19] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *SOSP 99*, pages 48–63, Kiawah Island, SC, USA, December 1999.

[20] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gauthier. Cluster-based scalable network services. In *SOSP 97*, Saint-Malo, France, October 1997.

[21] E. Glinert, R. Kline, G. Ormsby, and G.B. Wise. Unwindows: Bringing multimedia computing to users with disabilities. In *Proceedings of IISF/ACMJ International Symposium on Computers as Our Better Partners*, pages 34–42, Tokyo, March 1994.

[22] S. Goel, M. Singh, D. Xu, and B. Li. Efficient peer-to-peer data dissemination in mobile ad-hoc networks. In *ICPPW*, Vancouver, B.C., Canada, August 2002.

[23] S. Gribble, M. Welsh, E. Brewer, and D. Culler. The multispace: An evolutionary platform for infrastructural services. In *Usenix*, Monterey, CA, USA, January 1998.

[24] P. Havinga. *Mobile Multimedia Systems*. PhD dissertation, University of Twente, Department of Computer Science, February 2000.

[25] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *MobiCOM 99*, pages 174–185, Seattle, WA, USA, August 1999.

[26] D. Helder and S. Jamin. End-host multicast communication using switch-tree protocols. In *Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems*, Berlin, Germany, May 2002.

[27] T. Hodes and R. Katz. A document-based framework for internet application control. In *Proceeding of the Second USENIX Symposium on Internet Technologies and Systems*, Boulder, CO, USA, October 1999.

[28] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MobiCOM 2000*, pages 56–67, Boston, MA, USA, August 2000.

[29] F. James. Lessons from developing audio html interfaces. In *Proceedings of ASSETS 1998*, pages 15–17, Marina Del Rey, CA, USA, April 1998.

[30] F. James. *Representing Structured Information in Audio Interfaces: A Framework for Selecting Audio Marking Techniques to Represent Document Structures.* PhD thesis, Stanford University, Palo Alto, CA, USA, June 1998.

[31] M. Kempa and V. Linnemann. On XML objects. In *Programming Language Technologies for XML*, Pittsburgh, PA, October 2002.

[32] G. Krasner and S. Pope. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, 1988.

[33] R. Kravets, C. Carter, and L. Magalhaes. A cooperative approach to user mobility. *ACM Computer Communications Review*, 31, 2001.

[34] R. Kravets and P. Krishnan. Power management techniques for mobile communication. In *MobiCOM 98*, pages 157–168, Denver, Colorado, USA, September 1998.

[35] M. Krell and D. Cubranic. V-lynx: Bringing the world wide web to sight impaired users. In *Proceedings of ASSETS 1996*, pages 23–26, Vancouver, British Columbia, April 1996.

[36] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and

B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *ASPLOS*, Cambridge, MA, USA, November 2000.

[37] L. Lamport. *Latex: A Document Preparation System.* Addison-Wesley, Reading, Mass., 1986.

[38] K. Luyten and K. Coninx. An xml-based runtime user interface description language for mobile computing devices. In *International Workshop on Design, Specification, and Verification (DSV-IS 2001)*, Glasgow, Scotland, UK, June 2001.

[39] S. Maes and T.V. Raman. A single authoring programming model for the next web. In *Scientific Emphasis Proceedings*, June 2001.

[40] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, Hewlett Packard Laboratories, 2002.

[41] N. Mitrovic and E. Mena. Adaptive user interface for mobile devices. In *9th International Workshop on Design, Specification, and Verification (DSV-IS 2002)*, Rostock, Germany, June 2002.

[42] S. Morley, H. Petrie, A.M¿ O'Neill, and P. McNally. Auditory navigation in hyperspace: Design and evaluation of a non-visual hypermedia system for blind users. In *Proceedings of ASSETS 1998*, pages 100–107, Marina Del Rey, CA, USA, April 1998.

[43] A. Mueller, T. Mundt, and W. Lindner. Using xml to semi-automatically derive user interfaces. In *Second International Workshop on User Interfaces to Data Intensive Systems (UIDIS'01)*, Zurich, Switzerland, May 2001.

[44] M. Neary, S. Brydon, P. Kmiec, S. Rollins, and P. Cappello. Javelin++: Scalability issues in global computing. In *Java Grande*, pages 171–180, San Francisco, CA, USA, June 2000.

[45] M. Neary, S. Brydon, P. Kmiec, S. Rollins, and P. Cappello. Javelin++: Scalability issues in global computing. *Concurrency: Practice and Experience*, 12:727–753, 2000.

[46] T. Oogane and C. Asakawa. An interactive method for accessing tables in html. In *Proceedings of ASSETS 1998*, pages 126–128, Marina Del Rey, CA, USA, April 1998.

[47] A. Oram, editor. *Peer to Peer Harnessing the Power of Disruptive Technologies*. O'Reilly and Associates, first edition, 2001.

[48] M. Papadopouli and H. Schulzrine. Network connection sharing in an ad hoc wireless network among collaborative hosts. In *NOSSDAV 99*, Bell Labs, New Jersey, USA, June 1999.

[49] M. Papadopouli and H. Schulzrinne. Seven degress of separation in mobile ad hoc networks. In *Globecom*, San Francisco, CA, USA, November 2000.

[50] S. Portigal. Auralization of document structure. Master's thesis, University of Guelph, Canada, 1994.

[51] T.V. Raman. *Audio System for Technical Readings*. PhD thesis, Cornell University, Itica, NY, USA, May 1994.

[52] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Sigcomm 2001*, San Diego, CA, USA, August 2001.

[53] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal, Special Issue on Peer-to-Peer Networking*, 6(1), 2002.

[54] S. Rollins. Audio XmL: Aural interaction with xml documents. Master's thesis, University of California at Santa Barbara, Santa Barbara, CA, USA, 2000.

[55] S. Rollins and K. Almeroth. Deploying an infrastructure for technologically enhanced learning. In *ED MEDIA*, Denver, CO, USA, June 2002.

[56] S. Rollins and K. Almeroth. Pixie: A jukebox architecture to support efficient peer content exchange. In *ACM Multimedia*, Juan-Les-Pins, France, December 2002.

[57] S. Rollins, K. Almeroth, D. Milojicic, and K. Nagaraja. Power-aware data management for small devices. In *Workshop on Wireless Mobile Multimedia (WoWMoM 2002)*, Atlanta, GA, USA, September 2002.

[58] S. Rollins, R. Chalmers, J. Blanquer, and K. Almeroth. The active information system (ais): A model for developing scalable web services. In *Internet Multimedia Systems and Applications*, Kauai, Hawaii, USA, August 2002.

[59] S. Rollins and N. Sundaresan. AVoN calling: AXL for voice-enabled web navigation. In *The Ninth International World Wide Web Conference*, Amsterdam, The Netherlands, May 2000.

[60] S. Rollins and N. Sundaresan. A framework for creating multi-modal interfaces for xml documents. In *The International Conference on Multimedia and Expo*, New York, NY, USA, July 2000.

[61] M. Roussopoulos, P. Maniatis, E. Swierk, K. Lai, G. Appenzeller, and M. Baker. Person-level routing in the mobile people architecture. In *USITS*, Boulder, Colorado, USA, October 1999.

[62] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Middeware*, Heidelberg, Germany, November 2001.

[63] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent, peer-to-peer storage utility. In *SOSP 2001*, Canada, November 2001.

[64] A. Rudenko, P. Reiher, G. Popek, and G. Kuenning. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2(1):19–26, 1998.

[65] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *MMCN*, San Jose, CA, USA, January 2002.

[66] M. Satyanarayanan, J. Kistler, L. Mummert, M. Ebling, P. Kumar, and Q. Lu. Experience with disconnected operation in a mobile computing environment. In *USENIX Symposium on Mobile and Location-Independent Computing*, Cambridge, MA, USA, August 1993.

[67] M. Satyanrayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, August 2001.

[68] S. Sheu, K. Hua, and T. Hu. Virtual batching: A new scheduling technique for video-on-demand servers. In *DASFAA*, pages 481–490, Melbourne, Australia, April 1997.

[69] B. Shneiderman, M. Alavi, K Norman, and E. Borkowski. Windows of opportunity in the electronic classroom. *Communications of the ACM*, 38(11):19–24, November 1995.

[70] T. Starner. The role of speech input in wearable computing. *IEEE Pervasive Computing*, 1(3):89–93, 2002.

[71] M. Stemm and R. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications*, August 1997.

[72] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrisnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Sigcomm 2001*, San Diego, CA, USA, August 2001.

[73] E. Swierk, E. Kiciman, N. Willams, T. Fukushima, H. Yoshida, V. Laviano, and M. Baker. The roma personal metadata service. In *IEEE Workshop on Mobile Computing Systems and Applications*, December 2000.

[74] D. Tsichritzis. Reengineering the university. *Communications of the ACM*, 42(6):93–100, June 1999.

[75] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3), September 1991.

[76] C.M. Wilson. Listen: A data sonification toolkit. Master's thesis, University of California at Santa Cruz, Santa Cruz, CA, USA, 1996.

[77] D. Wu, A. Swan, and L. Rowe. An Internet MBone broadcast management system. In *Proceedings of Multimedia Computing and Networking 1999*, San Jose, CA, USA, January 1999.

[78] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava. On peer-to-peer media streaming. In *ICDCS*, Vienna, Austria, July 2002.

[79] T. Ye, H. Jacobson, and R. Katz. Mobile awareness in a wide area network of info-stations. In *MobiCOM 1998*, pages 109–120, Denver, Colorado, USA, September 1998.

[80] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. Using smart clients to build scalable servics. In *Usenix Annual Technical Conference*, Anaheim, CA, USA, January 1997.

[81] M. Zajicek and C. Powell. Building a conceptual model of the world wide web for visually impaired users. In *Proceedings of Ergonomics Society 1997 Annual Conference*, 1997.

[82] M. Zajicek, C. Powell, and C. Reeves. A web navigation tool for the blind. In *Proceedings of ASSETS 1998*, Marina Del Rey, CA, USA, April 1998.

[83] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[84] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV*, Port Jefferson, NY, USA, June 2001.

[85] G. Zipf. *Human Behavior and the Principle of Least Effort.* Addison-Wesley, Reading, MA, 1949.