comp 2

THE APECS PORTAL: A COLLABORATIVE NEUROSCIENCE REPOSITORY

by

VICTOR HANSON-SMITH

A THESIS

Presented to the Department of Computer
and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Master of Science

December 2007

[illegible] 72SH

THE APECS PORTAL: A COLLABORATIVE NEUROSCIENCE REPOSITORY

by

VICTOR HANSON-SMITH

A THESIS

Presented to the Department of Computer
and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Master of Science

December 2007

"The APECS Portal: A Collaborative Neuroscience Repository," a thesis prepared by Victor Hanson-Smith in partial fulfillment of the requirements for the Master of Science degree in the Department of Computer and Information Science. This thesis has been approved and accepted by:

Allen D. Malony

Dr. Allen D. Malony, Chair of the Examining Committee

Oct. 15, 2007

Date

Committee in charge: Dr. Allen D. Malony, Chair
Dr. John Conery
Dr. Dejing Dou

Accepted by:

[illegible]

Dean of the Graduate School

An Abstract of the Thesis of

Victor Hanson-Smith for the degree of Master of Science

in the Department of Computer and Information Science

to be taken December 2007

Title: THE APECS PORTAL: A COLLABORATIVE NEUROSCIENCE REPOSITORY

Approved: Allen D. Malony

Dr. Allen D. Malony, Chair

Electroencephalography (EEG) is a non-invasive method for measuring neural activity across a wide range of clinical and basic research contexts, including the detecting of seizures, studies of sleep patterns, and research on the cognitive and neural bases of language recognition. EEG research faces a number of computational challenges, relating to the abundance of experimental data and a paucity of organizational tools. To address these issues, we create a data storage and management system. This system includes a collaborative neuroscience repository called the APECS Portal. This thesis responds to the question: how do we design a collaborative repository for the digital artifacts required for EEG/ERP analysis?

CURRICULUM VITAE

NAME OF AUTHOR: Victor Hanson-Smith

PLACE OF BIRTH: Long Beach, CA

DATE OF BIRTH: June 8, 1981

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon
Seattle University

DEGREES AWARDED:

Master of Science in Computer and Information Science,
2007, University of Oregon

Bachelor of Science in Computer Science,
2003, Seattle University

AREAS OF SPECIAL INTEREST:

Collaborative Software   Web Portals   Computational Science

PROFESSIONAL EXPERIENCE:

Graduate Research Fellow, Department of Computer and Information Science, University of Oregon, 2007 - present

Teaching Assistant, Department of Computer and Information Science, University of Oregon, 2007 - present

Graduate Research Fellow, Neuroinformatics Centers, University of Oregon, 2005 - 2007

Software Engineer, Authora Inc, Seattle, WA, 2003 - 2005

Web Designer, Seattle University, Seattle, WA, 2002 - 2003

PUBLICATIONS:

Victor Hanson-Smith, Daya Wimalasuriya, and Andrew Fortier. Nutri-Stat: Tracking Young Child Nutrition. In *CHI 2006 Extended Abstracts on Human Factors in Computing Systems. Conference on Human Factors in Computing Systems*, ACM, 2006.

ACKNOWLEDGMENTS

This project was made possible with advising and mentorship from Allen Malony, who challenged me to explore an exciting and foreign field of study. I am also grateful to Gwen Frishkoff, for patiently teaching me about the neuroscience analysis pipeline. Additional thanks goes to Robert Frank, Robert Bell, and other researchers at the Neuroinformatics Center for support with countless engineering and design concerns.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

NEUROSCIENCE BACKGROUND

Electroencephalography (EEG) is a measurement of electrical brain activity. Non-invasive EEG electrodes are attached to the scalp to measure the electro-neural response to visual, auditory, olfactory, or other sensory stimulus. EEG is experimentally useful for a wide-range of clinical and research applications (for reviews see [18] [22] and [28]).

When EEG is used to measure the neural response to stimuli, scientists are looking for event-related potentials (ERPs). An ERP is the electrophysiological response to experimental stimulus, measured as a difference in electrical potential between electrodes. However, before ERPs can be detected in EEGs, the source data must undergo a multi-stage transformation.

An EEG headnet contains a mesh of electrodes. The head-net captures "raw continuous" EEGs, which are typically contaminated with electrical activity from eye blinks and other biological artifacts. In the first step of the transformation pipeline, independent component analysis (ICA) algorithms can be used to "clean" biological artifacts from EEG waveforms. The Automated Protocol for Evaluation of Electromagnetic Component Separation (APECS) [20] is a Matlab toolbox which implements several ICA algorithms, including Infomax, FastICA, and SOBI. APECS is based on Joe Dien's ICA Toolbox [5].

In the second step of the analysis pipeline, "cleaned continuous" data is segmented into smaller pieces. The segments which correspond to possible ERPs can be kept, while the uninteresting segments can be discarded.

Up to this point in the pipeline, the signal-to-noise ratio is too low to typically detect ERPs. In other words, concurrent brain activity usually conceals the desired ERP. This problem is solved in the third step of the pipeline. Tens or hundreds of cleaned-segmented EEGs are averaged together to elicit a better signal-to-noise ratio. When many EEGs are averaged together, ERPs stand-out clearly.

After ERPs are collected, correctly labeling the results is a challenge. ERPs can be described by several attributes, including polarity and onset delay. For example, the P100 is a type of ERP with a positive potential-difference which appears approximately 100 milliseconds after the onset of the experimental stimulus. The N400 is another ERP type with a negative potential-difference which appears about 400 milliseconds after the stimulus. ERP labeling is problematic because classification rules vary among research groups. Recent work presents techniques for automatically label some types of ERP data [17, 29].

In some cases, ERPs might be passed through a final processing stage to remove bad channels, to perform a baseline correction, or to perform a re-referencing montage. The product of this pipeline is a collection of segments, each containing experimentally useful ERPs. Figure 1.1 and 1.2 illustrate the analysis pipeline.

EEG data is difficult to work with for three reasons. First, there does not exist a universal EEG analysis pipeline. In some cases, the APECS cleaning might be ignored, or an alternative processing algorithm might be applied. In other cases, the continuous EEG might not be segmented at all. The specific steps for transforming raw data into experimentally useful ERPs depends on the precision of the recording equipment, the timing of the experimental stimuli, and many other factors. The flexibility of the EEG analysis pipeline has ramifications for information modeling, which is discussed in Chapter Three.

Second, the physical storage requirements for an EEG-based study can be significant. A brief example illustrates why EEG files are large. Suppose each EEG electrode records time samples as four-byte floating-point values. If we employ 256 electrodes, then our electrode mesh generates 1,024 bytes per sample (which equals one megabyte). If we record at 60 Hertz (which is 60 samples-per-second), then we generate 60 MB of EEG content per second. At these rates, a five-minute recording session generates 17.58 gigabytes of data. However, this estimate is conservative. We can explore the upper-bound of storage requirements if we use higher-resolution recording equipment. Specifically, the Geodesic Sensor Net [3] is capable of recording at 1,000 Hz. 256 electrodes recording at 1,000 Hz generates 292.97 gigabytes in five minutes. Our conservative example generates hundreds of gigabytes of data with a dozen human subjects. However, in our upper-bound example, a dozen human subjects generate several terabytes of raw data.

The third challenge is the complex inter-relationship between EEG data. As researchers modify and transform EEGs, they generate cleaned data, filtered data, and segmented data. Researchers also need supplementary digital artifacts, such as blink templates, sphering matrices, and job logs. This panoplay of content can be logically arranged into a hierarchical "provenance tree." By tracking provenance, scientists can determine the history of transformation applied to an artifacts, and which from which source the artifact originated.

Although EEG-studies generate a complex abundance of experimental data, there exists a paucity of neuroscience-specific organizational tools. In response to these challenges, this work aims to create a data storage and management system.

1.1 Claims about Current Practice

Our investigation reveals shortcomings with current EEG/ERP practice. Here we enumerate these claims.

File Storage

The researchers in our study store EEG/ERP files on local workstations. Unfortunately, distributed local storage systems make it difficult to track the provenance of a file throughout the EEG analysis pipeline.

Interface Limitations

Users currently interface with the APECS tools through Matlab. Although Matlab provides a robust analysis environment, it does not natively support job-batching and job-queueing.

Job Definitions

APECS job definitions are stored as Matlab files. These job files collude the algorithmic parameters with the input-file specifications. Simply put, a job "definition" and a job "instance" are not clearly separated.

Waveform Meta-data

EEG/ERP files contain waveforms, which can be described by their sampling-rate, channel count, segment count, and many other parameters. The meta-data about a waveform is stored in its file header. Searching for a specific waveform can be laborious because several files typically must be opened and inspected.

Ad-hoc Sharing

Under current practice, researchers share EEG/ERP-related files in an ad-hoc fashion. The team of researchers in our study shared files without using version-control software or a centralized repository. Although ad-hoc sharing can be convenient, it leads to consistency problems across workstations.

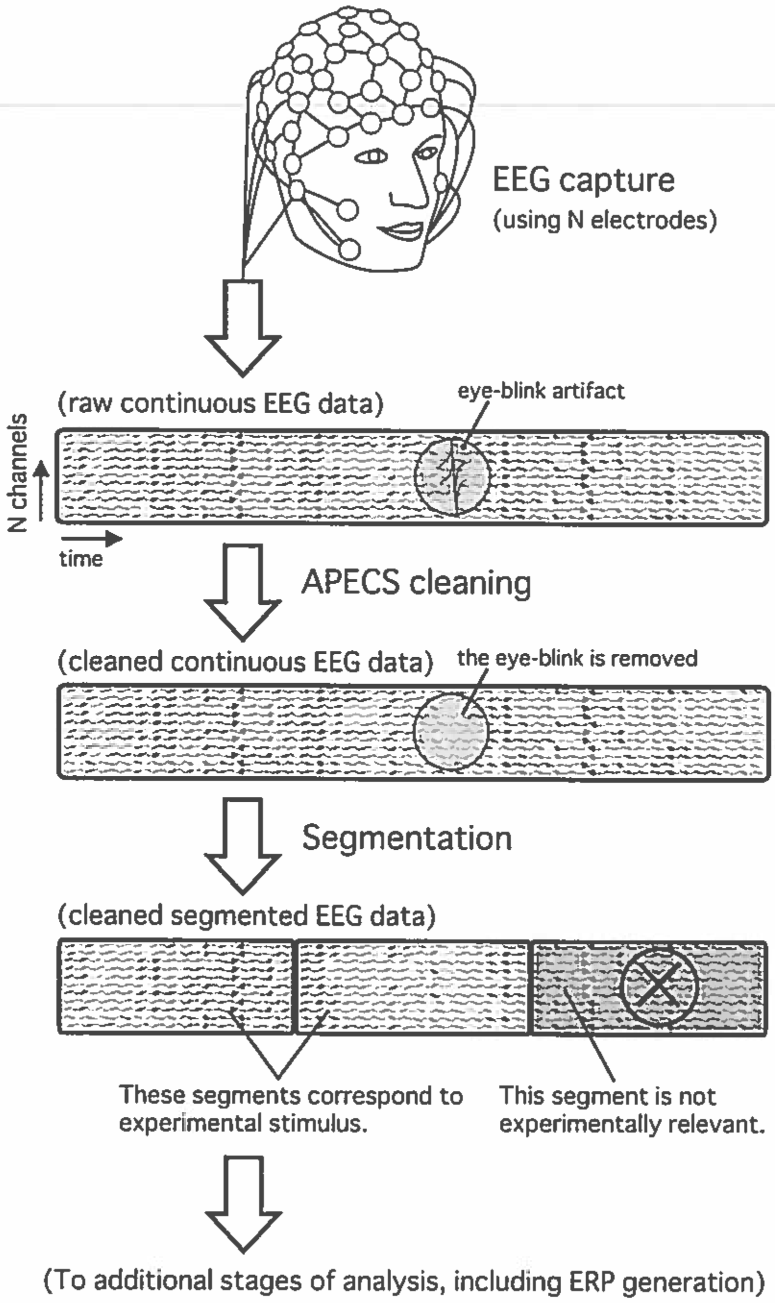

EEG capture
(using N electrodes)
(raw continuous EEG data)
eye-blink artifact
N channels
time
APECS cleaning
(cleaned continuous EEG data)
the eye-blink is removed
Segmentation
(cleaned segmented EEG data)
These segments correspond to experimental stimulus.
This segment is not experimentally relevant.
(To additional stages of analysis, including ERP generation)

FIGURE 1.1. The EEG analysis pipeline begins with captured data. In this illustration, N electrodes capture N individual channels of EEG data. The raw continuous data is "cleaned," using the APECS algorithms and then segmented for downstream assay.

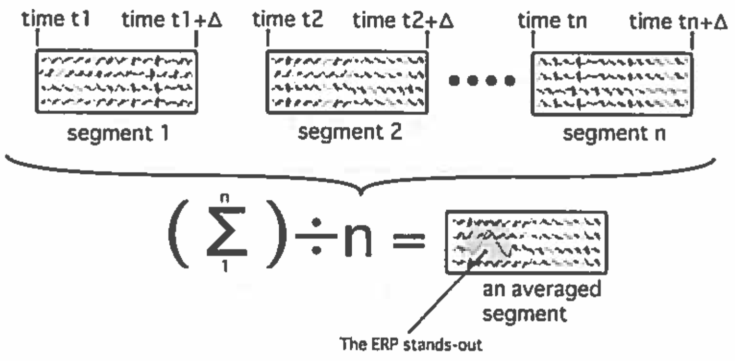


FIGURE 1.2. The EEG analysis pipeline continues with statistical averaging. EEG segments can be averaged to isolate experimentally useful event-related potentials (ERPs). In this figure, segments 1 through n were recorded while a single human subject was presented the same stimuli n times. ERPs can also be generated by averaging EEG data from several human subjects reacting to the same stimuli

CHAPTER II

METHODOLOGY

How do we design a storage and management system for the digital artifacts required in EEG/ERP analysis? This chapter discusses the concept of a collaboratory, and a methodology for designing collaboratories.

2.1 Collaboratories

The word "collaboratory" is used in many contexts, ranging from simple web applications to sophisticated virtual meeting rooms. For our discussion, we are concerned with collaboratories as a means to provide cooperative access to a large and dynamic dataset. This type of collaboratory has been successfully employed in related domains. For example, The Biomedical Informatics Research Network (BIRN) includes a collaboratory for studies of magnetic resonance imaging (MRI), electron microscopy, and other bioimaging technology [13][23]. The Zebrafish Information Network (ZFIN) is another example of a collaboratory [31]. ZFIN is built around the zebrafish model organism and provides tools for searching and analyzing genetic sequences. The Collaboratory for Multi-Scale Chemical Science (CMCS) is a collaboratory for chemistry research [11][24]. The CMCS supports several scales of analysis, from nano-scale quantum chemistry to deci-scale flow simulations.

2.2 Scenario-Based Design

The scenario-based design (SBD) methodology is a useful tool for articulating specific requirements for many instances of user-centric software, including a collaborative repository. SDB is a design framework which emphasizes the documentation of user activities. In this way, SBD is a means to accurately abstract, categorize, and capture technical knowledge about unfamiliar domains. This chapter briefly describes the SBD paradigm, and chapter three applies the methodology to the problem of EEG/ERP analysis.

In the SBD parlance, a system is composed of artifacts and human stakeholders. "Scenarios" are stories about users and their activities within a system. SBD suggests three design steps: activity design, information design, and interaction design. Figure 2.1 illustrates the relationship between these steps and the larger design process.

In the first step, we enumerate the tasks which should be supported by our system. At this step, the system is considered without concern for any implementation details. By postponing a discussion about implementation, the designer is free to abstractly generalize the system.

During activity design, we consider the tradeoff between the flexibility of a general-purpose system versus the specificity of a specialized system. At this step, we also bound the problem space by defining activity-based relationships between actors and artifacts. At the end of the activity design, we articulate "activity scenarios" which define the tasks to be supported by the emergent system.

In the second step, we elaborate upon the earlier activity scenarios by articulating which pieces of information are needed to accomplish each task. Whereas activity design is concerned with defining the activities in the system, information design is concerned with defining the information within those activities. The result of the information design step is a coherent set of "mock-up" prototypes. During this stage, we considered many design tradeoffs and aesthetic principles. For example, Gestalt principles of visual organization present guidelines for the proximity, similarity, and symmetry of objects in a visual field. The coherent use of colors, position, shape, and size establishes expectations which gives each user an intuitive ease [30].

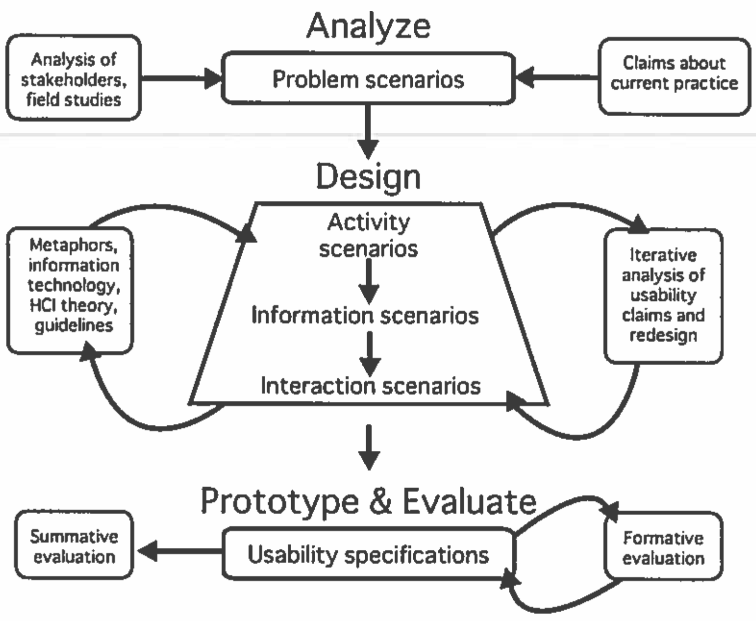

Analyze
Analysis of stakeholders, field studies
Problem scenarios
Claims about current practice
Design
Metaphors, information technology, HCI theory, guidelines
Activity scenarios
Information scenarios
Interaction scenarios
Iterative analysis of usability claims and redesign
Prototype & Evaluate
Summative evaluation
Usability specifications
Formative evaluation

FIGURE 2.1. An overview of the scenario-based methodology, reproduced from Rosson and Carrol [30]

In the third design step, interaction design, we define the mechanisms for retrieving and manipulating information. Simply put, interaction design is about ensuring that users can perform appropriate actions at appropriate times. At this step, we decompose the earlier activity scenarios into action sequences. Transforming action sequences into a functional web-based prototype raises interesting design challenges. In chapter four, we address these challenges. In chapter five, we discuss the current version of the APECS portal.

CHAPTER III

DESIGNING THE PORTAL

This chapter explains the design process behind the APECS Portal. Impatient readers can skip to chapter four, in which we discuss our design result.

3.1 Requirements Analysis

I began my investigation by answering the following questions: Who will use the system? What objects (virtually) exist in the system? How do the users and objects interact?

3.1.1 Stakeholder Profile

Many different types of users employ the EEG/ERP analysis pipeline. Although no archetypical user exists, we can describe the actors in our system according to a continuum of scientific needs. At one end of the spectrum, users experimentally capture raw EEG. These users need a storage system for large volumes of digital content. At the other end of the spectrum, users analyze pre-recorded EEG/ERP content and need a management system. Depending on the research environment, a user might focus on one aspect of the analysis pipeline, or operate at both ends of the spectrum.

3.1.2 Relationships Among Stakeholders

The relationships among stakeholders can be defined with respect to the EEG analysis pipeline.

Inter-stage collaboration occurs between actors focused on separate analysis stages. For example, User A might focus on capturing waveform content and User B might focus on cleaning and processing the content generated by User A.

Intra-stage collaboration occurs between actors focused on the the same step in the analysis pipeline. For example, User A and User B might cooperatively analyze the results of cleaned data. Or, User A and User B might collaboratively capture and store waveforms.

Despite these classifications, the user relationships are typically blurred. For example, User A and User B might collaborate between stages in one context, but collaborate on the same stage in another context.

3.1.3 Artifacts

My investigation into the EEG/ERP domain reveals several artifacts which are necessary for waveform analysis. The following section enumerates these artifacts and describes their role within the portal.

EEG/ERP files contain electrical waveform content. An EEG file is recorded outside the system, using NetStation (or other EEG capturing technology). Once a waveform is committed to the repository, it can be processed, segmented, and/or averaged. EEGs which have been averaged can contain experimentally useful ERPs. EEG/ERP files exist in a diversity of forms, but generally can be described according to three factors: processing status, segmentation status, and averaging state. The types of processing which can be applied to waveform data include ICA separation, bad-channel removal, and baseline correction. Unprocessed waveform data is considered "raw." The segmentation process divides a waveform into smaller units. The number and length of the resultant segments can vary. Individual segments can be discarded, which means that segmented EEG/ERPs might not contain all the original waveform content as the original unsegmented file.

All EEG/ERP files share the following set of attributes: file size, number of segments, number of channels, number of samples, number of events, last modified date, upload date, and a timestamp indicating when the file was committed to the repository.

Blink templates contain a single-column matrix of electrical potential values. Blink templates correspond to the apex of a blink event, and can be used to "clean" eye-blinks from EEG data. Although blink templates are derived from waveform data, blink templates lack a temporal component.

All blink templates share the following set of attributes: file size, number of samples, last modified date, upload date, and a timestamp indicating when the file was committed to the repository.

Sphering matrices are used to produce whitened data. Sphering matrices are produced by decomposing the covariance matrix into eigenvectors and eigenvalues. A sphering matrix is multiplied by "centered" waveform data to generate whitened data. **APECS output logs** are brief text files with relevant messages produced during the APECS execution. **Excel spreadsheets** are used to store information about human subjects and EEG analysis results.

3.1.4 Hierarchical Task Analysis

My preliminary investigation of the EEG/ERP domain revealed six high-level tasks which should be supported by the repository. These tasks include: creating a system account, adding content to the repository, searching the repository, cleaning EEG/ERP with APECS, organizing/grouping files, and sharing content with other users. Figures 3.1, 3.2, 3.3, 3.4, 3.5, and 3.6 illustrate a hierarchical task analysis for the six high-level tasks. Each analysis diagram describes a task and its dependent subtasks. It is important to notice that the hierarchical task analysis decouples the task description from the task implementation. By separating the description from the implementation, we can gather system requirements without being constrained by the technology limitations.

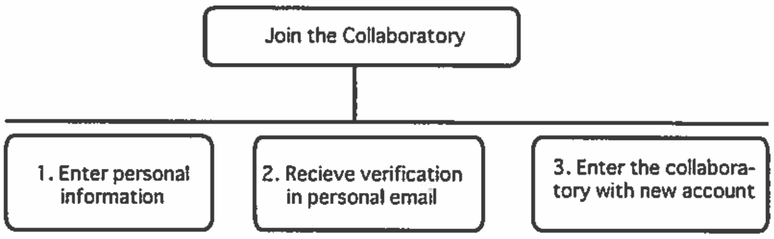

Join the Collaboratory
1. Enter personal information
2. Recieve verification in personal email
3. Enter the collaboratory with new account

FIGURE 3.1. An HTA for account creation: Creating an account is the simplest task in our system. Users create an account by entering the APECS portal, submitting information about themselves, and then checking their personal mail system for instructions about their new account.

3.1.5 Summary of Themes

This section discusses several themes which were revealed during our investigation of EEG/ERP analysis.

APECS Parameter Reuse / Job-Batching

An APECS operation is defined by multiple parameters which control ICA protocol, blink activity, data whitening, and other algorithmic behavior. Users invest large amounts of time configuring these parameters. Therefore, it is critical to reuse APECS definitions in support of job-batching.

Storage Constraints

EEG/ERP files can be very large and impose significant physical storage demands. Depositing files in a shared location can overcome these storage constraints. However, centralized storage systems can lead to alternative limitations relating to network latency.

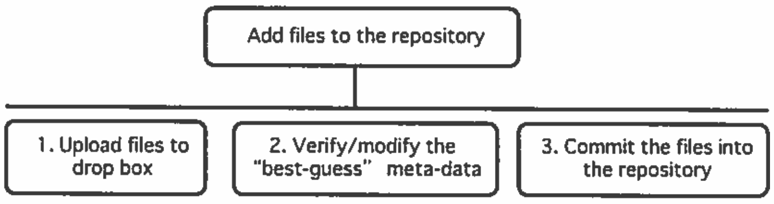

Add files to the repository
1. Upload files to drop box
2. Verify/modify the "best-guess" meta-data
3. Commit the files into the repository

FIGURE 3.2. An HTA for adding files: Users add files to the repository by using our drop-box model. Each user has a personal drop-box, and files can be uploaded into that box. New files are not officially part of the repository until they are "committed." Users commit files by first navigating to their personal drop-box. Second, the user verifies and/or modifies the system's "best-guess" about the meta-data in each uploaded file. Finally, the user commits the file to the repository.

Provenance

Each step in the EEG/ERP analysis pipeline generates intermediate files. Capturing these artifacts can be beneficial for analyzing results and repeating experiments. The challenge is to hierarchically organize these artifacts with regard to provenance.

Organizational Context

The requirements for file organization are diverse. Objects need to be clustered as "projects," but also need to be organized hierarchically. The need exists for a flexible organizational infrastructure.

Data Sharing

Allowing multiple users to read and/or modify an object is critical to the collaborative nature of our system. Shared artifacts require an information model which constrains object ownership.

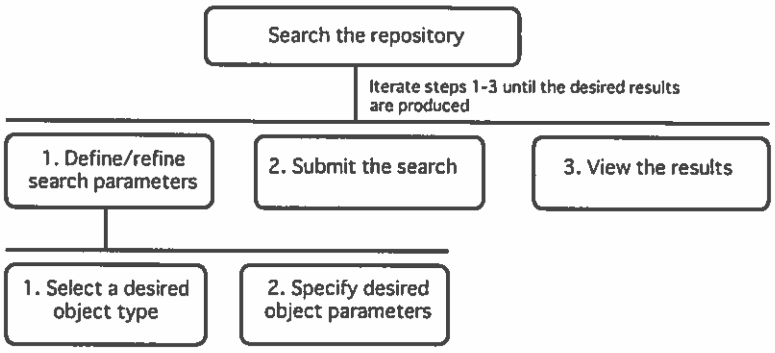

Search the repository
Iterate steps 1-3 until the desired results are produced
1. Define/refine search parameters
2. Submit the search
3. View the results
1. Select a desired object type
2. Specify desired object parameters

FIGURE 3.3. An HTA for searching the repository: Users search the repository by first defining search parameters. Specifically, users specify the desired types of objects over which the search will occur and specify the properties of those desired objects. Second, the user submits the search parameters. The system processes the query, and finally the user can view the results.

3.1.6 Problem Scenarios

In the following section, we describe the current practice for five primary tasks in our system. In the descriptions, we introduce two hypothetical users: Fred and Susan. Fred is mostly concerned with the *development* of EEG analysis methods, whereas Susan is primarily concerned with the *application* of EEG analysis.

Susan stores an EEG file

Susan wants to store a new waveform file. The file was exported from NetStation, and contains experimentally useful EEG content. Susan wants to store the file so that she can later process the file and analyze its contents. Susan does not use a shared storage system with her colleagues, so she imports the EEG file onto her laptop. The operating system on Susan's laptop organizes files according to a hierarchical directory-based structure. This means that Susan can store the EEG file in only one

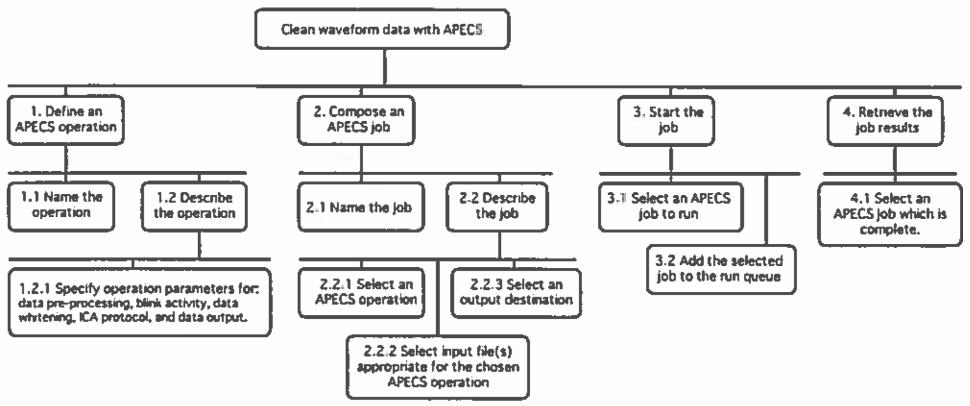

Clean waveform data with APECS
1. Define an APECS operation
2. Compose an APECS job
3. Start the job
4. Retrieve the job results
1.1 Name the operation
1.2 Describe the operation
2.1 Name the job
2.2 Describe the job
3.1 Select an APECS job to run
4.1 Select an APECS job which is complete.
3.2 Add the selected job to the run queue
1.2.1 Specify operation parameters for: data pre-processing, blink activity, data whitening, ICA protocol, and data output.
2.2.1 Select an APECS operation
2.2.3 Select an output destination
2.2.2 Select input file(s) appropriate for the chosen APECS operation

FIGURE 3.4. An HTA for cleaning waveforms: Users process waveform data with APECS by first defining an APECS operation. Each operation is defined by a unique name and a set of algorithmic parameters. Next, users compose an APECS job. Each job combines specific input files with an abstract APECS operation. Finally, users start the jobs and retrieve the processed results.

directory (unless she make copies or soft links). She must choose the file location carefully.

Susan considers the many different ways to categorize her new EEG file. On one hand, she wants to organize it with respect to provenance. On the other hand, she wants to store the file in a directory with EEGs from the same project. A third approach is to keep EEGs in a folder separate from ERPs. Ultimately, she decides to group the EEG with files from the same project and not worry about provenance.

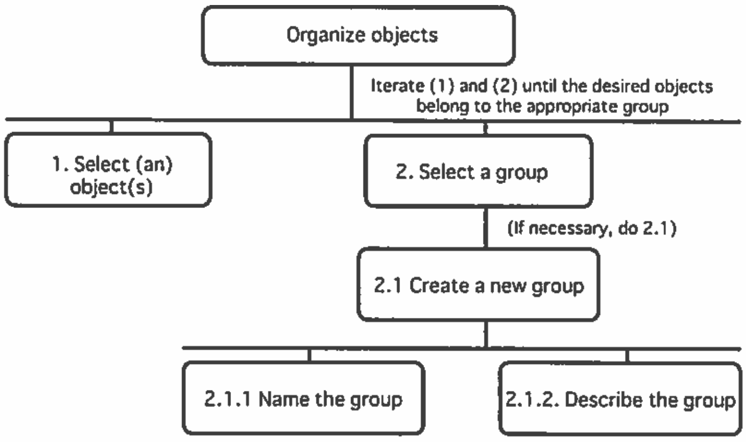


FIGURE 3.5. An HTA for organizing objects: Users organize objects in the repository by first selecting a set of objects to be grouped. Users add the selected files to an existing group, or create a new group.

Susan searches for a waveform file

Several weeks ago, Susan stored a particular EEG file on her laptop and now she wants to retrieve the waveform. Susan remembers that the EEG was "raw" and contained 128 data channels. Susan also knows that she stores her files in directories according to projects. Unfortunately, Susan cannot remember in which project she grouped her desired file.

Susan opens the search tool in her laptop's operating system. The search tool is not aware of waveform meta-data in EEG files. Susan's search capabilities are limited. She sets the "last-modified" parameter to search for files modified within the last month. Susan submits the query and receives dozen of results. Susan scrolls through the results. Some of the files have descriptive extensions, of Susan's own invention. For example, ".bl" means the file was processed for baseline correction.

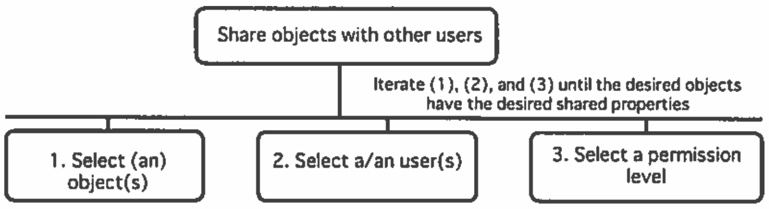

Share objects with other users
Iterate (1), (2), and (3) until the desired objects have the desired shared properties
1. Select (an) object(s)
2. Select a/an user(s)
3. Select a permission level

FIGURE 3.6. An HTA for sharing data: Data-sharing is almost a trivial task. Users share objects with other users by first selecting object(s) to share, then selecting a target user and permission levels.

As Susan parses the results, she finds a few files which might be candidates for her search. To determine which file is correct, Susan needs to perform a linear search over the waveform meta-data embedded in the candidates's header. Susan opens each file and reads the meta-data. Eventually, she finds her desired file.

Fred removes eye-blinks from an EEG file

Fred has a raw EEG file and he wants to remove eye-blink artifacts from the embedded waveform. Fred also has a blink template, which corresponds to the apex of the blink event.

Fred opens Matlab on his workstation. He loads the APECS driver file, which initiates a multi-part questionnaire. Fred follows the instructions and specifies information about data pre-processing, blink activity, data whitening, and data output. He also specifies the input file (the target EEG) and selects an independent component analysis (ICA) protocol.

When Fred finishes the questionnaire, the APECS job begins and useful output streams down his workstation's terminal. Fred waits several minutes for the job to complete and then retrieves the results.

Fred shares files with Susan: Version One

Fred has an ERP file he wants to share with Susan. The file is small, so he sends it to Susan as an email attachment. In the email body, Fred reminds Susan which project this ERP belongs to.

Later, Susan receives Fred's email. She downloads the attachment and saves it with other files which are related to this project. Susan has her own file-naming scheme, so she renames the ERP file. Susan and Fred now share a file with two different names.

Fred shares files with Susan: Version Two

Fred has a set of EEG files he wants to share with Susan. The entire package of files is three gigabytes large. Fred gives Susan access to his file-server, and Susan begins downloading the files to her local workstation. The download progress bar indicates that the download will take over two days, so Fred and Susan consider other options. Ultimately, Fred writes the files to a DVD and mails the disc to Susan via express mail.

Susan organizes a set of waveform files

Susan recently received a large set of files from Fred. Now, she wants to incorporate these files onto her local workstation. As mentioned in a previous problem scenario, Susan's operating system organizes files according to a hierarchical directory-based structure. Susan wants to organize the new files according to several topologies, including provenance and file-type. To represent this additional information, Susan creates short text files which she strategically sprinkles throughout the new directories.

3.1.7 Claims Analysis

Our system requirements can be distilled into a set of claims about the current practice of EEG/ERP analysis. Each claim describes a system feature which has notable effects on each user's experience.

Files are stored on local workstations.

- + each user can build organizational system that makes sense to him or her
- - but provenance is difficult (or impossible) to track
- - and data sharing is difficult to automate

Users interface with APECS directly through Matlab.

- + APECS is easy to deploy
- - but the interface is limited to a command-line

APECS job definitions are stored in Matlab files. The definitions mix the algorithmic parameters with the specifications for input files.

- + provides a quick way to define APECS jobs
- - but thwarts job-batching

Meta-data about a waveform is stored in its file header.

- + is convenient for analyzing a small set of waveforms
- - but searching for files is laborious

Files are shared in an ad-hoc fasion.

- + is convenient
- - but leads to consistency problems

3.2 Activity Design

During the requirement analysis step, we identified five primary tasks which should be supported by our system. In this section, we redefine those scenarios by presupposing the existence of a collaborative repository. Our result is five "activity scenarios," which are stories about people interacting with a system.

3.2.1 Useful Analogies

Precisely describing the tasks in our system is paramount, but selecting the appropriate language can require an intuitive leap. One approach is to use analogies to articulate important aspects of our system's activities. In this section, we explore analogies.

Adding content to the repository like . . .

- shelving books (content needs to be archivally stored in some organized fashion)
- passing a doorman (all objects are inspected before they enter the repository)
- bringing an elephant home (the repository can store a large amount of volume, but maybe not ALL content belongs in the repository. This is a question of self-editing and self-censorship).
- hanging an ornament on a tree (all the content is organized in context of provenance)

Searching the repository is like. . .

- finding your car keys (sometimes the search is intended to find a single important object)
- collecting rocks (sometimes the search is intended find many similar objects)
- staring at clouds (sometimes search is performed so as to detect patterns)
- following a creek (all the data can be searched in context of the "upstream" and "downstream" content in the provenance chain)

Processing waveform data is like. . .

- editing digital photographs (the final product is produced through a combination of several smaller steps)
- polishing precious stones (the desired part of the waveform can be surrounding by non-useful waveform data. "Carving" and "polishing" is required).
- slicing carrot sticks (a waveform can be sliced into even-length segments)
- auditioning musicians in an orchestra (some channels might need to be removed or corrected)

Sharing objects with other users is like. . .

- sharing a secret with all your friends (users place trust in the system and other users)
- releasing open-source software (someone else might work on your project)
- working on a crossword puzzle with a friend (objects are typically shared in the context solving some larger scientific problem).

Grouping objects is like. . .

- taking a poll (the group might reveal surprising aspects of the data)
- labeling test tubes (most data needs to be labeled in the context of a larger experiment)

3.2.2 Activity Scenarios

In the following section, we improve upon our previous problem scenarios by presupposing the existence of a file storage and management system. Here we articulate new activity scenarios:

Susan stores an EEG file

Susan wants to store a waveform file which was exported from NetStation. Susan plans to process the file, then analyze its contents. She starts by uploading the file to the shared repository. Next, Susan opens the repository's interface and finds the new file in her "dropbox." The repository system automatically determined that the file contained EEG content. Furthermore, the system extracted waveform meta-data from the EEG file header, including the number of channels, the number of samples, and the number of segments. Susan views the automatically-extracted meta-data and verifies its correctness. Finally, Susan "commits" the EEG file into the repository.

Susan searches for a waveform file

Several weeks ago, Susan stored a particular EEG file in the collaborative repository. Now, she wants to retrieve this waveform file. Susan remembers that the EEG was "raw" and contained 128-chanels. Susan opens the repository search engine, which is aware of all the repository files and their associated meta-data. She defines parameters to search for all EEG files which are "raw," which contain 128-channels, and were committed to the repository within the last month. The search returns a short list of files. Susan is able to search through the meta-data about all these files in a continuous and intuitive fashion.

Fred removes eye-blinks from an EEG files

Fred has a raw EEG file and he wants to remove eye-blink artifacts from the embedded waveform. Fred also has a blink template, which corresponds to the apex of the blink event.

Fred opens collaboratory interface. He looks at a list of existing APECS operations and decides to create a new APECS operation instead of re-using previous operations. Fred provides works through a multi-part questionnaire, specifying information about data pre-processing, blink activity, data whitening, and data output behavior. Fred saves this configuration of parameters. Next, Fred composes an APECS job around his new APECS operation. He specifies the input files to be his EEG file and blink template. He indicates that the APECS output should be automatically placed in a new group.

Fred starts the job, and the system puts the job into an execution queue. Fred leaves the collaboratory, but his job remains in the persistent queue. Later, he returns to the collaboratory and finds that his job is completed. Furthermore, the results were bundled into a new group and automatically archived.

Fred shares a waveform file with Susan

Fred has a file in the repository, which he wants to share with Susan. Fred opens the repository interface and searches for the file. He opens the sharing tool, and selects Susan as the share-recipient. Fred gives Susan "full-access" to the file, which allows her to both read the file and modify its meta-data. However, Fred could instead give Susan "restricted" access where she could only read the file.

Later, Susan enters the repository. Her history log reports that Fred shared a file with her. Susan searches and finds the file. The file is already inside the repository, so Susan does not need to re-label or re-commit the file.

Susan organizes a set of waveform files

Susan recently committed a large set of files to the repository. When the files were previously in her "dropbox," Susan labeled the files according to their source and subject. At the same time, the repository automatically ascertained meta-data about any waveform content. Now that the files are inside the repository, Susan wants to group the files by projects.

Susan enters the repository and searches for the new files. She iteratively selects sets of files and adds group labels to those sets. In many cases, a file belongs to several sets.

3.2.3 Claims Analysis

Our activity scenarios point to a set of claims about a system for EEG/ERP collaboratiion. Here, we enumerate and analyze these claims:

The search engine is aware of waveform meta-data.

- + users can search for EEG/ERP files without needing to open individual file headers.

Files must be committed from the dropbox into the repository.

- + forces users to inspect and verify waveform meta-data.
- + ensures that provenancial information is specified early in the analysis
- - but could be frustrating if a user does not want to inspect many files, and if that user is certain the waveform meta-data is correct.

Users can arbitrarily group objects.

- + Users can create very personalized organizational systems
- + The concept of a "group" can be used in many contexts: projects, job results, groups of groups, and future situations not yet forseen.
- - but can be confusing if the user interface is not extensible

APECS operations are defined separately from APECS jobs.

- + APECS operations can be reused
- + APECS operations can be shared
- + batches of jobs can be constructed from a single operation

APECS jobs are executed within a job queue.

- + naturally supports job-batching
- - but might be frustrating if users want to prioritize jobs

3.3 Information Design

In the first part of the SBD process, we articulated the shortcomings of current practices for EEG/ERP analysis. In the second part, we began to describe improved practices. In this section, we elaborate on our proposed activity scenarios and enumerate the specific information which is needed to accomplish each task.

3.3.1 Useful Analogies

The biggest challenge at this step is to describe how information looks within the system. In the second part of the SBD process, we used analogies to describe activities. In the section, we use analogies again, but as means to understand the possibilities of information design.
Content in the repository looks like. . .

- tracks/timelines (waveform content has a temporal aspect)
- hair comb (blink templates are used to filter eye-blinks)

A search interface is like. . .

- a clerk (the search interface will take requests and retrieve objects)
- a census (the search engine can be used to determine how many objects share a set of attributes.)

Shared content is like. . .

- a library book (requires a pattern for judiciously sharing data)
- family-style meals (simultaneous access, which sometimes causes collisions)
- warmth from a fireplace (simultaneous access, without collisions)

Groups of objects are like. . .

- a family (everyone is different, but related)
- a folder of documents (the group is a container)

3.3.2 Information Design Scenarios

Susan stores an EEG file

Susan wants to store a waveform file which was exported from NetStation. Susan plans to process the file, then analyze its contents. She starts by uploading the file to the shared repository. Next, Susan opens the repository's interface and navigates to her dropbox. She finds a list of recently uploaded files, along with her new file. The list contains file names, file types, and timestamp when the file was uploaded.

The repository system automatically determined that the file contained EEG content. Furthermore, the system extracted waveform meta-data from the EEG file header, including the number of channels, the number of samples, and the number of segments. Susan wants to verify this meta-data. She selects the file and watches the meta-data appears within context of the original file list. Susan can view the meta-data without navigating to a separate page.

Susan thinks the meta-data is correct, so she "commits" the EEG file into the repository. When she does this, the file disappears from her dropbox and reappears in a list of "recently committed files." The files in the "recently committed files" are displayed in a similar fashion to the dropbox files, except each file has an additional parameter "committed on."

Susan searches for a waveform file

Several weeks ago, Susan stored a particular EEG file in the repository. Now, she wants to retrieve this waveform file. Susan remembers that the EEG was "raw" and contained 128-chanels. Susan enters the repository and navigates to the search engine. On one side of the display are the tools for constructing a search query. On the other side of the display is a list of search results. When she first enters the repository search engine, the results list is empty.

Susan begins constructing her search query. First, she selects that she wants to search for waveform files. Next, she limits the search. The search tools allow for an unlimited number of search parameters to be specified. Susan adds a parameter to limit the search to waveform files with 128 channels, and another parameter to limit the search whose "committed on" date is within the last month.

Susan submits the search query, and results soon appear on the other side of the display. The list is short enough so that Susan quickly finds the file for which she was looking.

Fred removes eye-blinks from a file

Fred has a raw EEG file and he wants to remove eye-blink artifacts from the embedded waveform. Fred also has a blink template, which corresponds to the apex of the blink event. The blink template was generated in an application outside the collaboratory, by taking the single EEG sample corresponding to the apex of the blink spike.

In order to clean the EEG file, Fred needs to compose an APECS job. He creates a new job, names the job, and describes the job. Before Fred can connect input files to the job, he must select an APECS operation. On the opposite side of the screen, Fred sees a browser for APECS operations. He suspects that one of the previous operations can be reused for this job, so he begins browsing through the list. Fred can select each operation and view the parameters about data whitening, bad-channel removal, and other specifications.

Although Fred does not find any operation that can be wholly reused, he does find a close candidate. Fred would prefer not to entirely rebuild an APECS operation, so he clones the candidate operation and sets about making adjustments to the clone. The cloned operation appears in the operation browser. Fred selects the newly cloned version and modifies a small number of parameters in the operation questionnaire. Fred saves this configuration of parameters.

Fred returns to the APECS job editor, and his incomplete job definition. He selects his newly cloned operation as the job's operation. He then specifies his blink template file and his EEG file as inputs for the job. Finally, Fred specifies that the APECS output should be automatically placed in a new group.

Fred saves the job definition and starts the job. The system puts the job into an execution queue. In the job browser, the queue number appears beside the new job. Fred sees that his job will not be performed immediately, so he leaves the collaboratory, (but his job remains in the queue). Later, Fred returns to the collaboratory and finds that his job is completed. In the job browser, the status field says, "completed" along with the date of completion. Fred uses the search engine to browse objects by group, and finds the results from the job bundled and archived into a new group.

Fred shares a waveform file with Susan

Fred has a file in the repository which he wants to share with Susan. Fred enters the repository and searches for his file. Once he finds the file, he selects the sharing tool.

In the sharing tool, Fred needs to specify sharing behavior for his file. He gives Susan "full-access" to the file, which allows her to both read the file and modify its meta-data. However, Fred could instead give Susan "restricted" access where she could only read the file. After Fred closes the sharing tool, the shared file appears with a "+" next to its name in the file browser.

Later, Susan enters the repository. Her history log reports that Fred shared a file with her. The log indicates the filename, the filetype, and the timestamp when Fred shared the file. Susan uses the repository's search engine to finds the file.

Susan organizes a set of waveform files

Susan recently committed a large set of files to the repository. When the files were previously in her "dropbox," Susan labeled the files according to their source and subject. At the same time, the repository automatically ascertained meta-data about any waveform content. Now that the files are inside the repository, Susan wants to group the files by projects.

Susan enters the repository and searches for the new files. She iteratively selects sets of opens the grouping tool. The grouping tool looks like a paperclip. The tool prompts Susan to select a destination group, or specify the name and description of a new group.

3.3.3 Claims Analysis

Our information design scenarios point to a set of claims about the information required to accomplish tasks within the EEG/ERP repository. Here, we enumerate and analyze these claims:
Objects in the repository are displayed in lists.

- + Users are familiar with visually parsing lists
- - but displaying objects in a tree hierarchy might be more appropriate

The search engine allows for an unlimited number of search queries to be specified

- + Users can construct very-specific searches

APECS operations can be cloned and reused.

- + is useful when a user wants to create a new operation which is similar to an existing operation
- - but might lead to an explosion of forked operation variants

APECS job results are automatically bundled into groups.

- + is useful for automatically retrieving results
- - but might not provide enough flexibility if the results need to be automatically split into several groups

Users see "history logs" of actions.

- + is useful for giving context to actions
- - but might be more useful if the system includes a means to "undo" actions.

3.4 Interaction Design

3.4.1 Interaction Scenarios

In the interaction scenarios, we begin to consider how our original scenarios can be turned into state-machines. An exhaustive explanation of each interaction scenario would make this thesis of ungainly length. Instead, we include preliminary sketches which illustrate user interactions.

Susan stores an EEG file

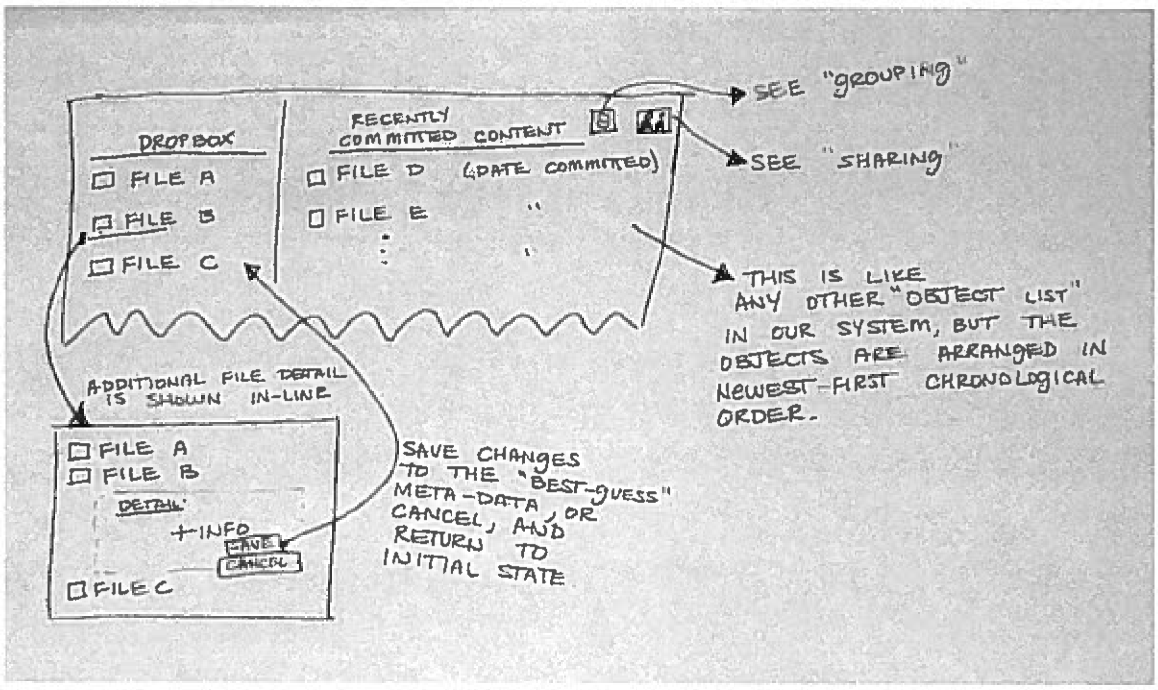

SEE "GROUPING"
SEE "SHARING"
DROPBOX
FILE A
FILE B
FILE C
RECENTLY COMMITTED CONTENT
FILE D (DATE COMMITTED)
FILE E
THIS IS LIKE ANY OTHER "OBJECT LIST" IN OUR SYSTEM, BUT THE OBJECTS ARE ARRANGED IN NEWEST-FIRST CHRONOLOGICAL ORDER.
ADDITIONAL FILE DETAIL IS SHOWN IN-LINE
DETAIL
+INFO
SAVE
CANCEL
FILE C
SAVE CHANGES TO THE "BEST-GUESS" META-DATA, OR CANCEL, AND RETURN TO INITIAL STATE

FIGURE 3.7. An early sketch of the upload interaction

Susan wants to store a waveform file which was exported from NetStation. Susan plans to process the file, then analyze its contents. She starts by uploading the file to the shared repository. Next, Susan opens the repository's interface and navigates to her "dropbox." She finds a list of recently uploaded files, along with her new file. The list contains file names, file types, and the date and time of when the file was uploaded.

The repository system automatically determined that the file contained EEG content. Furthermore, the system extracted waveform meta-data from the EEG file header, including the number of channels, the number of samples, and the number of segments. Susan wants to verify this meta-data. She selects the file and watches the meta-data appear in view. The meta-data appears within context of the original file list, without forcing her to navigate to a separate page.

Susan thinks the meta-data is correct, so she "commits" the EEG file into the repository. When she does this, the file disappears from her dropbox and reappears in a list of "recently committed files." The files in the "recently committed files" are displayed in a similar fashion to the dropbox files, except each file has an additional parameter "committed on."

Susan searches for a waveform file

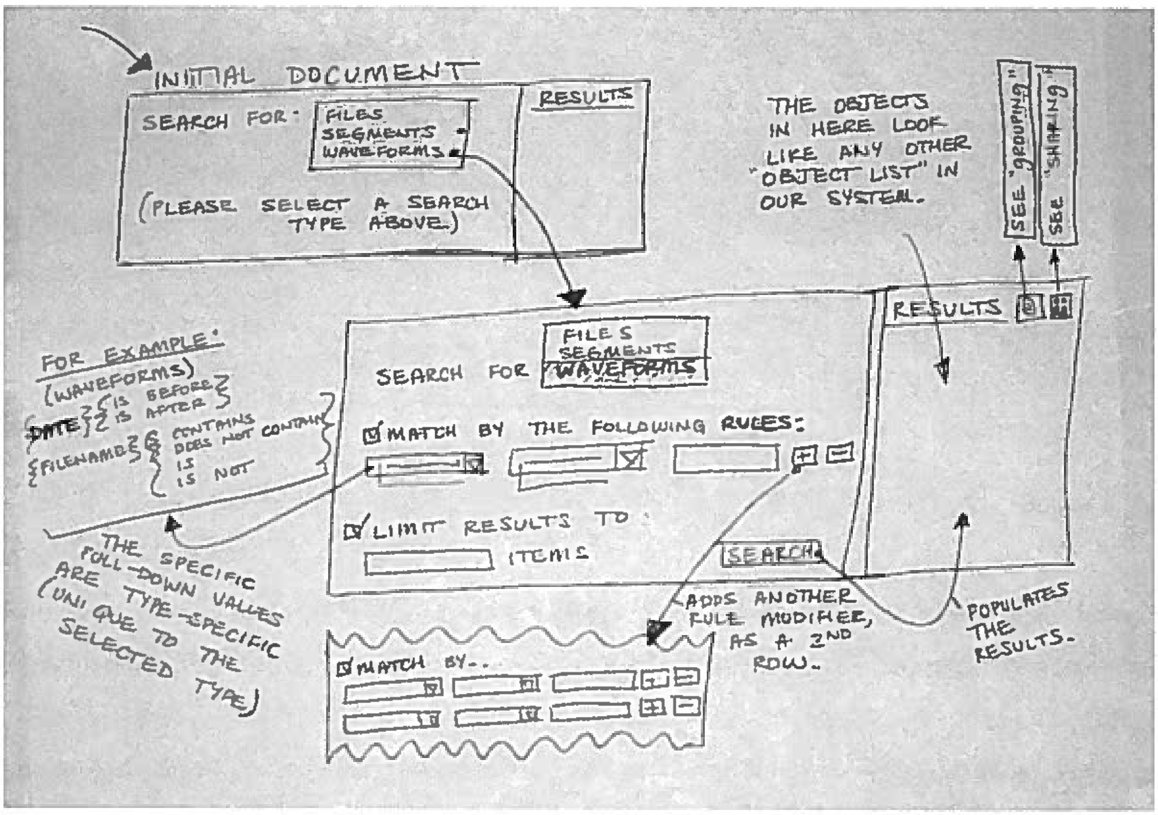

INITIAL DOCUMENT
SEARCH FOR:
FILES
SEGMENTS
WAVEFORMS
(PLEASE SELECT A SEARCH TYPE ABOVE.)
RESULTS
THE OBJECTS IN HERE LOOK LIKE ANY OTHER "OBJECT LIST" IN OUR SYSTEM.
SEE "grouping"
SEE "SHIFTING"
RESULTS
FOR EXAMPLE:
(WAVEFORMS)
DATE IS BEFORE / IS AFTER
FILENAME CONTAINS / DOES NOT CONTAIN / IS / IS NOT
SEARCH FOR
FILES
SEGMENTS
WAVEFORMS
MATCH BY THE FOLLOWING RULES:
LIMIT RESULTS TO
ITEMS
SEARCH
THE SPECIFIC PULL-DOWN VALUES ARE TYPE-SPECIFIC (UNIQUE TO THE SELECTED TYPE)
ADDS ANOTHER RULE MODIFIER, AS A 2ND ROW.
POPULATES THE RESULTS.
MATCH BY..

FIGURE 3.8. An early sketch of the search interaction

Several weeks ago, Susan stored a particular EEG file in the repository. Now, she wants to retrieve this waveform file. Susan remembers that the EEG was "raw" and contained 128-chanels. Susan enters the repository and navigates to the search engine. She selects "search for waveforms," upon which additional tools appear. The first appended tool allows Susan to find waveforms which match a set of rules. For example, she can search exclusively for waveforms whose recorded-date is before last week. The tool includes three fields: the parameter, the predicate, and the value. In this example, the parameter is *recorded-date*, the predicate is *is before*, and the value is a timestamp seven days ago. The tool also includes buttons to add and subtract rules. An arbitrary number of rules can be specified. The second appended tool allows Susan to limit the result count. Susan uses all these tools to construct a search query.

Susan presses the "search" button, waits for the network to communicate and the server to think, and the sees a list of results appear on the right-side of the portal. We intentionally leave ambiguous the order in which the results should be listed. Options include: ascending/descending by date, in alphabetical order, or grouped by parent file.

If Susan's desired waveform appears on the right, then Susan's task is complete. If the file does not appear, then Susan returns to the left-side of the portal interface and modifies her search criteria.

Fred removes eye-blinks from a file

Fred has a raw EEG file and he wants to remove eye-blink artifacts from the embedded waveform. Fred also has a blink template, which corresponds to the apex of the blink event. The blink template was generated in an application outside the collaboratory, by taking the single EEG sample corresponding to the apex of the blink spike.

Fred enters the collaboratory and navigates to the APECS tools. The initial page has two large columns, labeled "Operations" and "Jobs." The operations side contains tools for constructing APECS operations, while the jobs side contains tools to compose and manage APECS jobs. In the parlance of the APECS portal, an

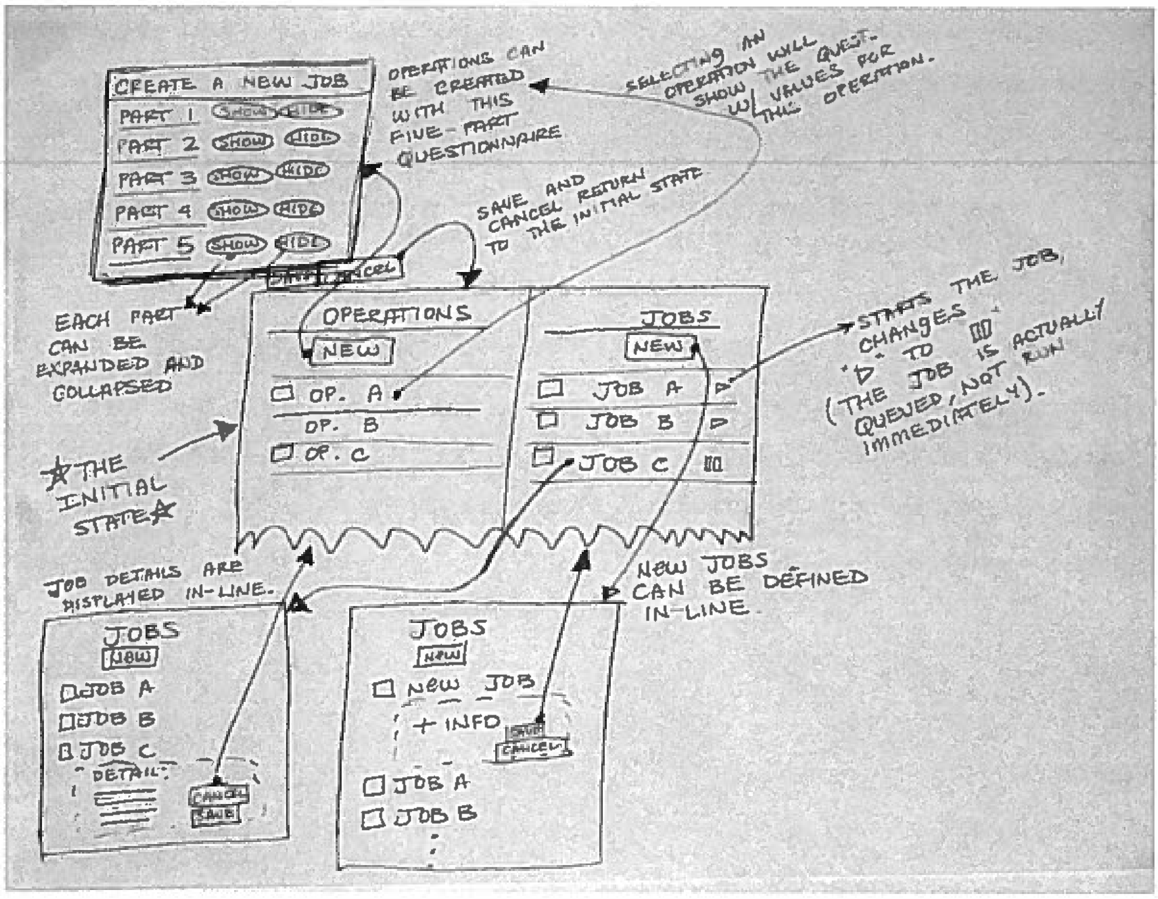

CREATE A NEW JOB
PART 1 SHOW HIDE
PART 2 SHOW HIDE
PART 3 SHOW HIDE
PART 4 SHOW HIDE
PART 5 SHOW HIDE
SAVE CANCEL
OPERATIONS CAN BE CREATED WITH THIS FIVE-PART QUESTIONNAIRE
SELECTING AN OPERATION WILL SHOW THE QUEST. W/ VALUES FOR THIS OPERATION.
SAVE AND CANCEL RETURN TO THE INITIAL STATE
EACH PART CAN BE EXPANDED AND COLLAPSED
OPERATIONS
NEW
OP. A
OP. B
OP. C
JOBS
NEW
JOB A
JOB B
JOB C
STARTS THE JOB, CHANGES ▷ TO ▥ (THE JOB IS ACTUALLY QUEUED, NOT RUN IMMEDIATELY).
★THE INITIAL STATE★
JOB DETAILS ARE DISPLAYED IN-LINE.
NEW JOBS CAN BE DEFINED IN-LINE.
JOBS
NEW
JOB A
JOB B
JOB C
DETAIL
CANCEL
SAVE
JOBS
NEW
NEW JOB
+ INFO
SAVE
CANCEL
JOB A
JOB B

FIGURE 3.9. An early sketch of the APECS eyeblink cleaning interaction

operation is a set of algorithmic parameters which define how a particular waveform file should be processed with respect to data-whitening, bad channel removal, and ICA protocol. A "job" is a user-specified relationship between a set of input files, an APECS operation, and an output locations. Jobs can be added to the job queue and executed.

In order to clean the EEG file, Fred needs to compose an APECS job. He presses the *NEW* button on the jobs side, and composition tools appear in-line. The new job appears as a new row, above previous jobs. Later, after it is saved, it can be resorted with the rest of the jobs. The composition tool includes fields for the job's name, description, APECS operation, input files, and output group. The name, description, and output group can be specified immediately, but the input files field cannot be specified until an APECS operation is selected. The type and quantity of input files is determined by the operation selection. If the portal does not contain any save APECS operations, then Fred cannot clean his file until he first creates a new operation.

He presses the *NEW* button on the operations side, and a five-part questionnaire appears above the job list. Each part of the questionnaire can be collapsed or expanded. After Fred fills values into the form, he presses the save button and the questionnaire disappears. His new job operation appears in the list of operations.

Fred returns to the APECS job editor, and his completes his job definition. He selects his new operation as the job's operation. He then specifies his blink template file and his EEG file as inputs for the job. Finally, Fred specifies that the APECS output should be automatically placed in a new group.

Fred presses the save button, which collapses the expanded information. Fred starts the job. The system puts the job into an execution queue. In the job browser, the queue number appears beside the new job. Fred sees that his job will not be performed immediately, so he leaves the collaboratory, (but his job remains in the queue). Later, Fred returns to the collaboratory and finds that his job is completed. In the job browser, the status field says, "completed" along with the date of completion. Fred uses the search engine to browse objects by group, and finds the results from the job bundled and archived into a new group.

Fred shares a waveform file with Susan

Fred has a file in the repository which he wants to share with Susan. Fred enters the repository and searches for his file. In the list of search results, he checks the box next to the file he wants to share with Susan. Fred presses the share button at the top of the display. A tool for sharing data appears as an overlay. The tool includes a drop-down list to select a user for sharing. The overlay tool also includes a button to share the file with another user, or remove the current sharer from the list.

In the sharing tool, Fred needs to specify sharing behavior for his file. He gives Susan "full-access" to the file, which allows her to both read the file and modify its meta-data. In the sharing tool, next to Susan's name, another drop-down box contains these permission parameters. (This feature is not pictured in the preliminary sketch).

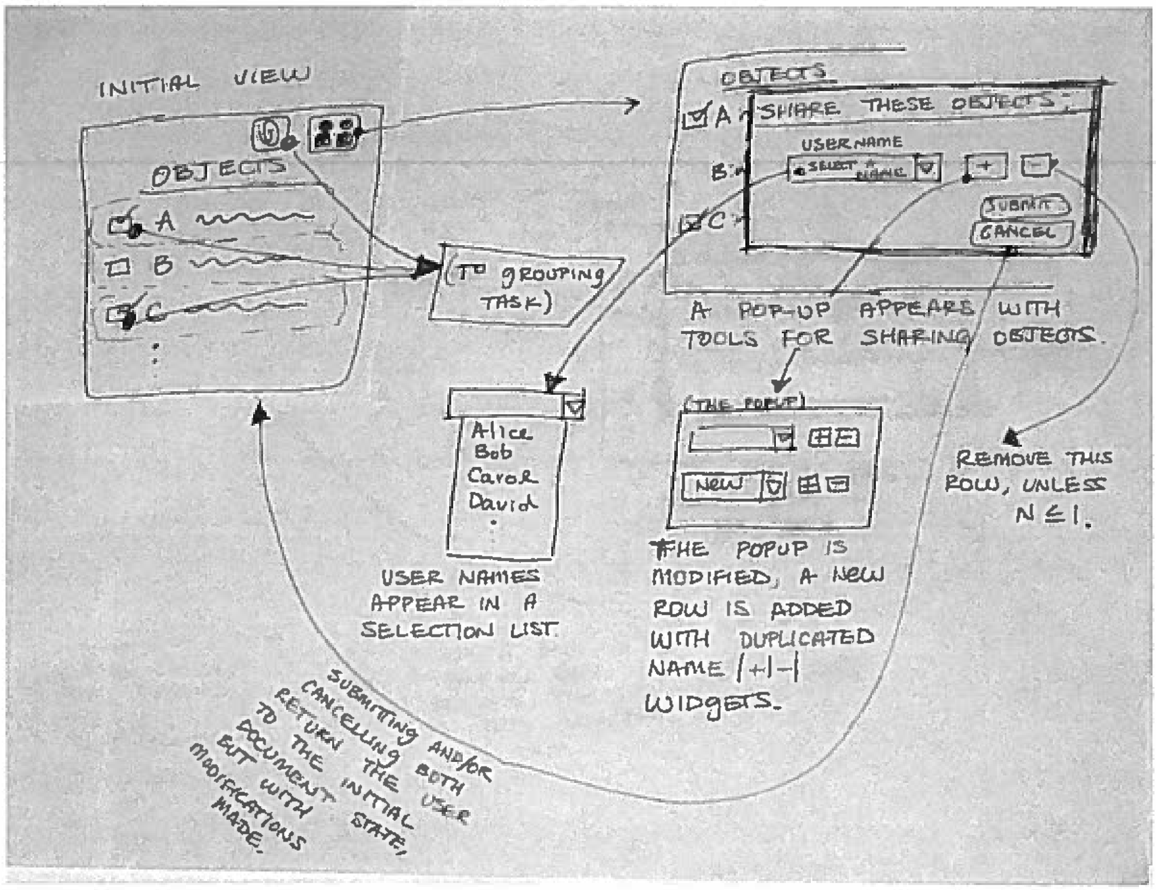

INITIAL VIEW
OBJECTS
A
B
C
(TO GROUPING TASK)
OBJECTS
A
B
C
SHARE THESE OBJECTS
USER NAME
SELECT A NAME
+
−
SUBMIT
CANCEL
A POP-UP APPEARS WITH TOOLS FOR SHARING OBJECTS.
Alice
Bob
Carol
David
USER NAMES APPEAR IN A SELECTION LIST
(THE POPUP)
New
THE POPUP IS MODIFIED, A NEW ROW IS ADDED WITH DUPLICATED NAME |+|−| WIDGETS.
REMOVE THIS ROW, UNLESS N ≤ 1.
SUBMITTING AND/OR CANCELLING BOTH RETURN THE USER TO THE INITIAL STATE, DOCUMENT BUT WITH MODIFICATIONS MADE.

FIGURE 3.10. An early sketch of the file-sharing interaction

Fred could instead give Susan "restricted" access where she could only read the file. After Fred closes the sharing tool, the shared file appears with a "+" next to its name in the file browser.

Later, Susan enters the repository. Her history log reports that Fred shared a file with her. The log indicates the filename, the filetype, and the timestamp when Fred shared the file. Susan uses the repository's search engine to finds the file.

Susan organizes a set of waveform files

Susan recently committed a large set of files to the repository. When the files were previously in her "dropbox," Susan labeled the files according to their source and subject. At the same time, the repository automatically ascertained meta-data

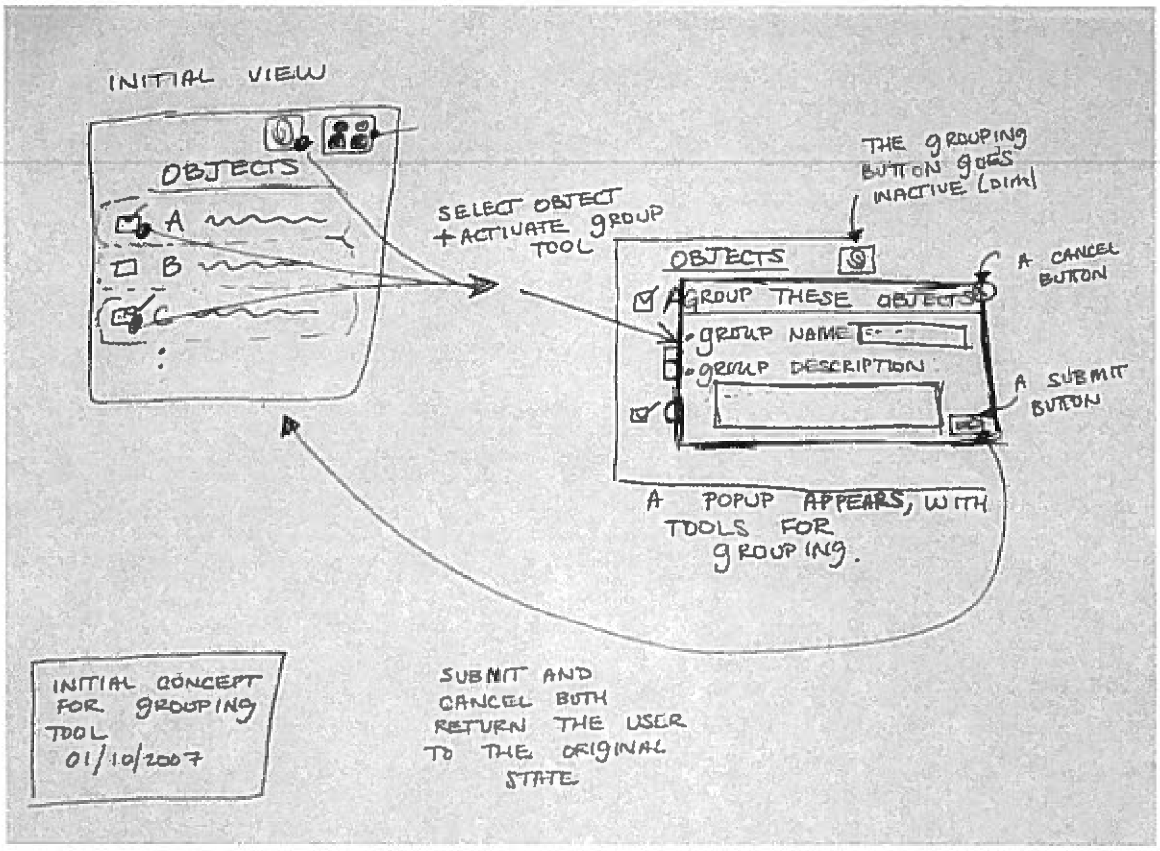

INITIAL VIEW
OBJECTS
A
B
C
SELECT OBJECT + ACTIVATE GROUP TOOL
THE GROUPING BUTTON GOES INACTIVE (DIM)
OBJECTS
GROUP THESE OBJECTS
GROUP NAME
GROUP DESCRIPTION
A CANCEL BUTTON
A SUBMIT BUTTON
A POPUP APPEARS, WITH TOOLS FOR GROUPING.
SUBMIT AND CANCEL BOTH RETURN THE USER TO THE ORIGINAL STATE
INITIAL CONCEPT FOR GROUPING TOOL 01/10/2007

FIGURE 3.11. An early sketch of the file-group interaction

about any waveform content. Now that the files are inside the repository, Susan wants to group the files by projects.

Susan enters the repository and searches for the new files. Her results appear in a list. Susan checks the box next the files which she wants to group. Susan presses the group button, and an overlay appears with a tool for grouping. The overlay is titled "Group these objects." The tool contains a field for the group name, and a field for the group description. (Not pictured) The tool also contains a drop-down list of other group names. If Susan wanted to add these files to a group, then she could selected a pre-existing group name instead of entering a new group name.

Susan names her group, describes her group, and then presses the select button. The overlay disappears. The group name appears in small text next to each of the grouped files.

3.4.2 Claims Analysis

The grouping and sharing tools appear in several interfaces, as "overlay" tools.

- + adds convenience to grouping and sharing
- - but will be confusing if object lists are not consistent throughout the portal

Before a search query can be constructed, an object type must be selected.

- + allows for query fields specific to that object type
- - but can lead to inconsistent states if the user changes their original object-type selection

An arbitrary number of query parameters can be added.

- + the search tool is flexible and specific

Each section of the APECS operation can be expanded and collapsed.

- + is a logical way to fit a linear list of questions into a small space
- - but can be difficult to find a specific question when several sections are collapsed.

CHAPTER IV

DESIGNING INTERFACES FOR THE WEB

Although SBD provides a solid methodology for eliciting requirements, there are some engineering challenges which SBD does not directly answer. For example, how do we engineer a state-based prototype from a collection of scenario-based requirements? This question is an active source of research. In 2006, Liang, et. al. surveyed twenty-two approaches for synthesizing state-based models from scenarios [25]. Related work addresses the problem of *implied* scenarios incorrectly appearing in resultant state machines [26]. Although previous research takes important steps towards automating software design, most of it focuses on standalone software applications. New challenges arise from the limitations of web-based applications. Before we can discuss these limitations, we need to define a few important tools for building web applications.

4.1 Tools

4.1.1 JSR168 Portlets

Portlets are user interface components which can be collected into a web portal. Portlets extend Java servlets, which means portlets respond to HTTP requests and produce markup fragments. A portlet "container" delivers a cohesive web application by aggregating the hypertext fragments from several individual portlets. The Java Portlet Specification (JSR168) defines common interoperability between a portlet and its parent container. Specifically, JSR168 articulates the render parameters for a portlet and the persistent data for the portlet session. In theory, a JSR168-compliant portlet can be hosted by any JSR168-compliant container.

Other popular technologies for serving markup fragments include Ruby on Rails [9] and Django [2]. Although these options have unique advantages, we ultimately adopted JSR168 portlet technology for three reasons. First, portlet containers provide useful out-of-the-box features, including tools for managing user accounts, modifying the container layout, and monitoring live sessions. Second, portlet-based systems can be highly componentized. Extending a system of portlets can be a straightforward procedure because the portlets are separated from the portlet container, which is separated from the underlying services. Finally, each of our scenario-based tasks naturally maps to an individual portlet. Later sections of this paper will discuss the bridge between scenario-based requirements and state-based portlets.

A recent survey evaluated seven portlet containers against fourteen criteria [12]. The criteria included JSR168 compliance, ease of installation, security, and other factors. According to this study, the top-rated containers are Liferay [8], Exo [4], and the GridSphere Portal Framework [ref]. We adopted GridSphere due to its large support-base and long-standing stability. Gridsphere is the container for many collaboratory projects, including [23] [ref] [ref].

4.1.2 Asynchronous Java and XML (AJAX)

Portlets produce hypertext markup fragments. Until recently, web-based hypertext interfaces rarely provided the interactivity and level of dynamism which users expect from conventional standalone applications. Historically, the common approach for providing web-based interactivity was to embed non-hypertext objects inside HTML markup. Applets [6], Java WebStart [7], and Flash [1] use this approach. Early attempts at hypertext interactivity include the iFRAME element in Internet Explorer 3, the LAYER element in Netscape 4, and Microsoft Remote Scripting. Although these technologies innovated HTML dynamics, their widespread adoption was limited. In 2000, Microsoft's Outlook Web Access introduced the XMLHttpRequest object as a replacement to Java Applets. The XMLHttpRequest is a scripting object which makes possible asynchronous communication between a web-browser and a remote web-server. In 2002, the XMLHttpRequest was adopted by Mozilla, Safari, and other browsers. In 2006, the World Wide Web Consortium released a working draft of the XmlHttpRequest API [10]. The use of the XMLHttpRequest is commonly referred to as "asynchronous Java and XML," or AJAX.

Today, AJAX applications are nearly ubiquitous on the world wide web. However, the underlying design patterns are less clearly defined. This section discusses the hypertext design space and provides a methodology for automatically transforming scenario-based requirements into functional AJAX prototypes.

4.2 Design Strategies

The relationship between a human user, a hypertext portal interface, and a web-server is similar to the relationship between a surgeon, a nurse, and a tray of surgical tools. After a patient is prepped, the surgeon will likely begin the operation with a scalpel. As the operation proceeds, the surgeon might use many other tools, including clamps, retractors, speculums, and suction tubes. The surgeon can hold only a few instruments at once. Consequently, the nurse assists by delivering the remaining tools to the surgeon as needed.

We can also use an automotive analogy to understand the hypertext portal design space. Imagine an auto-mechanic in a garage well-equipped with tools. The mechanic has a car which needs repair, so she slides on a backboard underneath the chassis. The mechanic starts the repair task with a flashlight and a wrench, but soon needs a screwdriver, pliers, and more wrenches of different size.

These analogies share a common theme of a user (or a surgeon, or a mechanic) using a set of tools to accomplish a task. These tools are a subset of a much larger collection. For example, a surgeon might not use every surgical instrument on the tray and the mechanic will not likely use every tool in the garage. In both these examples, unforseen task constraints determine the specific set of tools which is actually needed.

4.2.1 Defining θ and δ

During the SBD process, we identified a set of primary tasks which should be supported by our system. Now we will reconsider these tasks in terms of *data* and *tools*. Let us consider any one of these tasks, t. In order to accomplish t, a user must employ a particular set of tools, θ, to access a particular set of data, δ. Unfortunately, the branching nature of many tasks makes it difficult to know *a priori* which specific tools and data will be needed. In other words, θ and δ are determined at runtime. At best, a designer can identify a larger set of tools and data which might *possibly* be needed to accomplish a task. For our discussion, we will refer to three sets of θ and δ:

(1) δ_0 and θ_0 are the initial sets of data and tools, given to the user at the start of the task.

(2) δ_a and θ_a are the actual sets of data and tools which are needed to accomplish a task. δ_a and θ_a are determined at runtime.

(3) δ_p and θ_p are the possible sets of data and tools which might be needed to accomplish a task.

These sets are related in the following way:

$\theta_0 \subset \theta_a \subset \theta_p$

$\delta_0 \subset \delta_a \subset \delta_p$

In conventional standalone applications, we can employ a "brute force" approach and make $\theta_0 := \theta_p$ and $\delta_0 := \delta_p$. However, in web-based hypertext interfaces, the brute force approach can be impractical because the size of a hypertext document is constrained by user expectations about response-time and browser performance. Consequently, software engineers must employ a more subtle strategy and provide θ_0 and δ_0 as true subsets of θ_p and δ_p

In the "subset-strategy", the user begins a task with a document D_0, containing θ_0 and δ_0. As the user works towards a goal, tools and data are appended to the page. For the simplicity of our discussion, we will consider δ and θ contained within a hypertext document D. In other words:

$\{\theta_0, \delta_0\} \in D_0$

$\{\theta_a, \delta_a\} \in D_a$

$\{\theta_p, \delta_p\} \in D_p$

Figure 4.1 illustrates the lifecyle of a dynamic hypertext document, couched in terms of initial, actual, and possible sets. This subset-strategy introduces two design challenges: how do we identify D_0? And, how should additional content be served?

4.2.2 Identifying θ_0 and δ_0

In this section, we discuss an approach for revealing the initial sets of data and tools needed to accomplish a particular task. Our approach uses the hierarchical task analysis (HTA) diagrams, which are produced in the early stages of SBD. If we transform the HTAs into data-flow graphs, we can discover dependencies between subtasks. These dependencies can be used to identify θ_0 and δ_0. Our approach is as follows:

In the first step, identify the "terminal" tasks in the hierarchical task diagram. The terminal tasks are the secondary and tertiary subtasks which appear as leaf nodes. We will refer to t's terminal tasks as t_0, t_1, ..., t_n. Figure 4.2 illustrates the HTA for the task of cleaning waveform data with APECS. In our example, the terminal subtasks are: naming the operation, specifying operation parameters, naming the job, selecting an APECS operation, selecting input files, selecting an output destination, selecting an APECS job to run, adding a job to the run queue, and selecting a completed job.

In the second step, enumerate the tools needed to accomplish each task. In our example, task 2.2.2 ("selecting an input file") requires a tool for browsing possible input files. For another example, look at task 1.2.1 ("specify operation parameters"), which requires a five-part questionnaire. A third example is task 3.2, which requires a control mechanism to "start" and "stop" jobs. After the tools have been identified for n terminal subtasks, we will have n subsets of tools: θ_{t_0}, ..., θ_{t_n}. θ_{t_i} corresponds to the tools for task i. It is possible that some tools will belong to multiple tool subsets. For example, a tool for browsing files might appear in several subtasks.

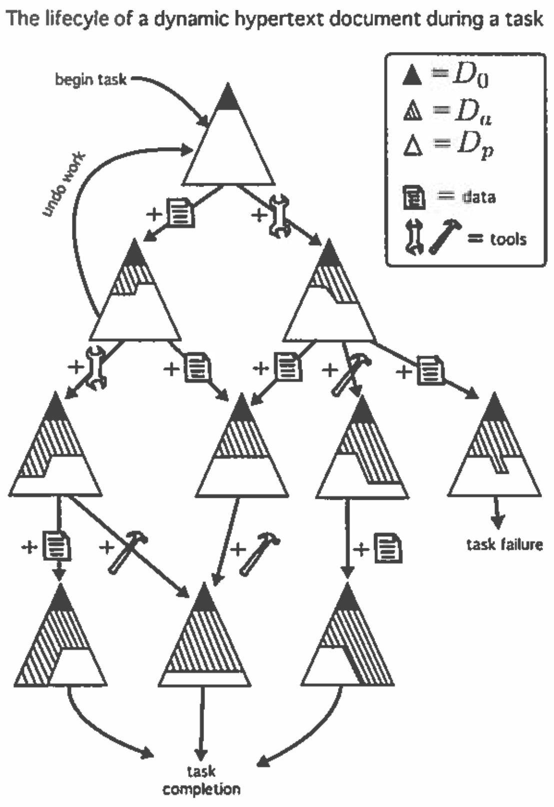


FIGURE 4.1. An example lifecycle of a dynamic hypertext document: The user begins a task with D_0, which contains an initial subset of the tools and data needed to accomplish the task. D_p is an imaginary document which contains every possible tool and data which might be needed to accomplish the task. D_a is the actual state of the document, containing only the tools and data which have been needed thus far. As the user works towards a goal, additional tools and data are appended to the document. Due to the branching nature of the task, D_a can take many forms. In some cases, document states converge. In other cases, the task can fail. (For example, the goal cannot be completed if required information does not exist).

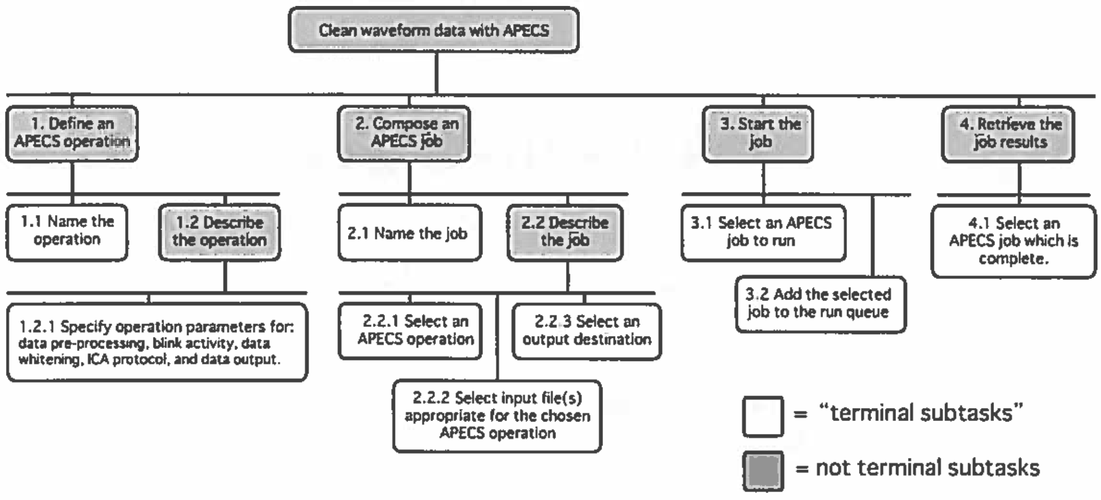

Clean waveform data with APECS
1. Define an APECS operation
2. Compose an APECS job
3. Start the job
4. Retrieve the job results
1.1 Name the operation
1.2 Describe the operation
2.1 Name the job
2.2 Describe the job
3.1 Select an APECS job to run
4.1 Select an APECS job which is complete.
3.2 Add the selected job to the run queue
1.2.1 Specify operation parameters for: data pre-processing, blink activity, data whitening, ICA protocol, and data output.
2.2.1 Select an APECS operation
2.2.3 Select an output destination
2.2.2 Select input file(s) appropriate for the chosen APECS operation
= "terminal subtasks"
= not terminal subtasks

FIGURE 4.2. An HTA for the task of cleaning waveform data with APECS

In the third step, enumerate the data needed to operate each tool. Each θ_{t_i} points towards δ_{t_i}, the set of data required to accomplish t_i. Returning to our example, the file browser in task 2.2.2 will require a list of all the files in the system, along with basic meta-data about those files. Similarly, the control mechanism in task 3.2 will require information about the current state of the selected APECS job.

In the fourth step, arrange the subtasks according to their data dependencies. In general, a dependency exists between subtasks t_j and t_k if the specific elements of δ_{t_k} cannot be determined until t_j is complete. In our example, there exist several data dependencies, one of which is between tasks 2.2.1 and 2.2.2. Each APECS operation can accept different types of input files. Therefore, the list of needed input files for task 2.2.2 will not be known until the user completes task 2.2.1. Figure 4.3 illustrates the dependency graph for the task of APECS cleaning.

In the final step, define θ_0 and δ_0 as the tools and data associated with the first-degree children in our dependency graph. In other words, D_0 should contain all θ_{t_i} and δ_{t_i} for independent tasks. In our example, figure 4.3 shows that D_0 should contain everything needed to accomplish tasks 1.1, 1.2.1, 2.1, and 2.2.3.

4.2.3 Serving θ_a and δ_a

Once we define the initial state of the hypertext interface, we need a strategy for appending additional data and tools to the document. AJAX provides technology for loading dynamic content, but it might not be obvious how to incorporate this content. We suggest two patterns for AJAX interfaces: *In-situ displays* and *on-the-fly tools.*

In-Situ Displays

In-situ displays are a design technique for appending information "in place." In-situ displays dynamically add detail to an interface without destroying the context of the current page. This design strategy is easiest to understand by example. Suppose a hypertext document contains a list of EEG files. The user can select a file to view more information about that file. The selection gesture triggers an AJAX request, which loads new content into an XMLHttpRequest 'responseText' buffer. The new

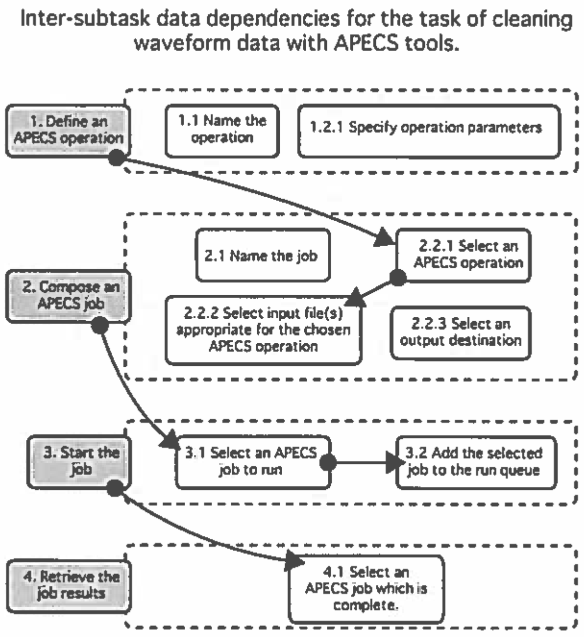

Inter-subtask data dependencies for the task of cleaning waveform data with APECS tools.
1. Define an APECS operation
1.1 Name the operation
1.2.1 Specify operation parameters
2. Compose an APECS job
2.1 Name the job
2.2.1 Select an APECS operation
2.2.2 Select input file(s) appropriate for the chosen APECS operation
2.2.3 Select an output destination
3. Start the job
3.1 Select an APECS job to run
3.2 Add the selected job to the run queue
4. Retrieve the job results
4.1 Select an APECS job which is complete.

FIGURE 4.3. A representation of data dependencies among tasks. In this illustration, the subtasks from 4.2 are rearranged. Terminal subtasks are grouped by their parent subtask. Arrows mark the data dependencies. For instance, the data required to accomplish task 2.2.1 cannot be determined until task 1 is complete. Similarly, task 3.2 is dependent on information produced by task 3.1. This dataflow diagram reveals the independent tasks which can be assigned to D_0: 1.1, 1.2.1, 2.1, and 2.2.3.

dynamic content might be displayed as a collapsable "div" beneath the filename. In other situations, the extra content might be displayed as a pop-up or a sidebar. The important idea is that additional information should be appended within the context of the user's current interface.

On-the-Fly Tools

On-the-fly tools are dynamically rendered widgets. Whereas in-situ displays address static data, on-the-fly tools deal with interactive elements. On-the-fly tools add interactivity to a page without destroying the current context. On-the-fly tools are appended to a page, instead of being separate from a page. In our collaboratory, we demonstrate this technique with a tool for composing APECS cleaning jobs. Each job connects a set of input files to a set of algorithmic parameters, also known as an APECS operation. When a user composes an APECS job, the types and quantity of required input files are not known until a specific operation is selected. Each operation has unique requirements for input files. For example, not all APECS operations use blink templates. The point of this example is that the input fields in the job-composing tool must be dynamic.

CHAPTER V

THE APECS PORTAL AND BEYOND

We applied our design strategies to engineer The APECS Portal, a collaborative neuroscience repository. Our collaboratory supports six primary tasks: creating an account, adding files, cleaning eye-blinks from waveforms, searching for data, organizing/grouping files, and sharing content with other users.

The APECS Portal stores files in a fifteen-terabyte Network Appliance (NetApp). However, the NetApp's conventional file structure cannot be used to represent alternative organizational topologies, such as provenance. Consequently, the APECS Portal implements a database containing waveform meta-data, file permissions, user histories, descriptions of experiments, and APECS cleaning logs. The database provides a flexible model for grouping files into arbitrary collections. The NetApp and the database together provide the foundation for our repository.

A collection of persistent services brokers communication between the database, NetApp, JSR168 portlets, and computational resources for APECS jobs. The File Watcher Service scans the repository and automatically extracts meta-data about newly uploaded files. Users can verify this "best-guess" meta-data inside the portal interface. The Job Management Service places APECS cleaning jobs in a queue, and dispatches cleaning tasks to a thirty-two node Intel Xeon cluster. Finally, the Database Gateway Service provides a unified point-of-access to marshall and unmarshall logical objects into and out of the database. In Section five of this paper, we discuss how an API for persistent services can play a role in future work.

The portal interface is constructed from several JSR168 portlets. In the "upload" portlet, users add files to the repository and inspect waveform meta-data. The "browse" portlet provides a search engine for EEG/ERPs and related files. The "organize" portlet includes tools for managing arbitrary data collections. In the "share" portlet, users can control policy for cooperative file access. Figures 5.2, 5.3, and 5.4 show screenshots from the portal interface. Figure 5.1 illustrates the overall system design.

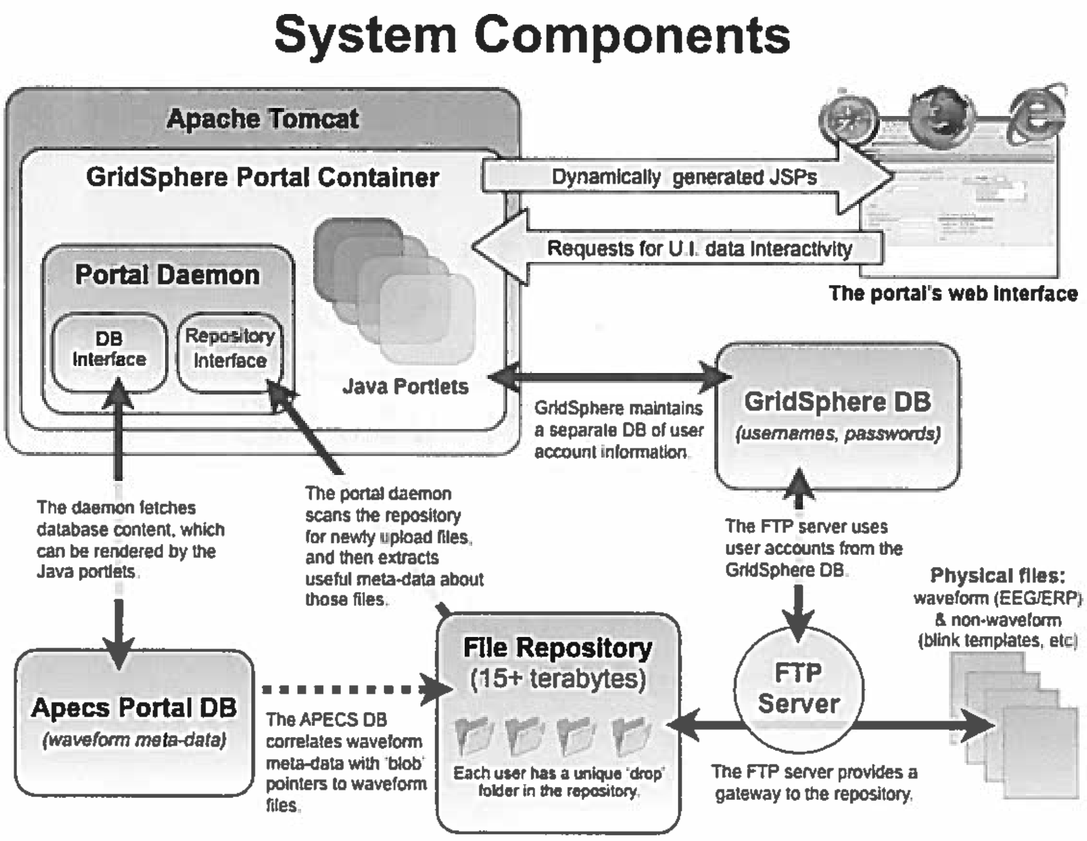
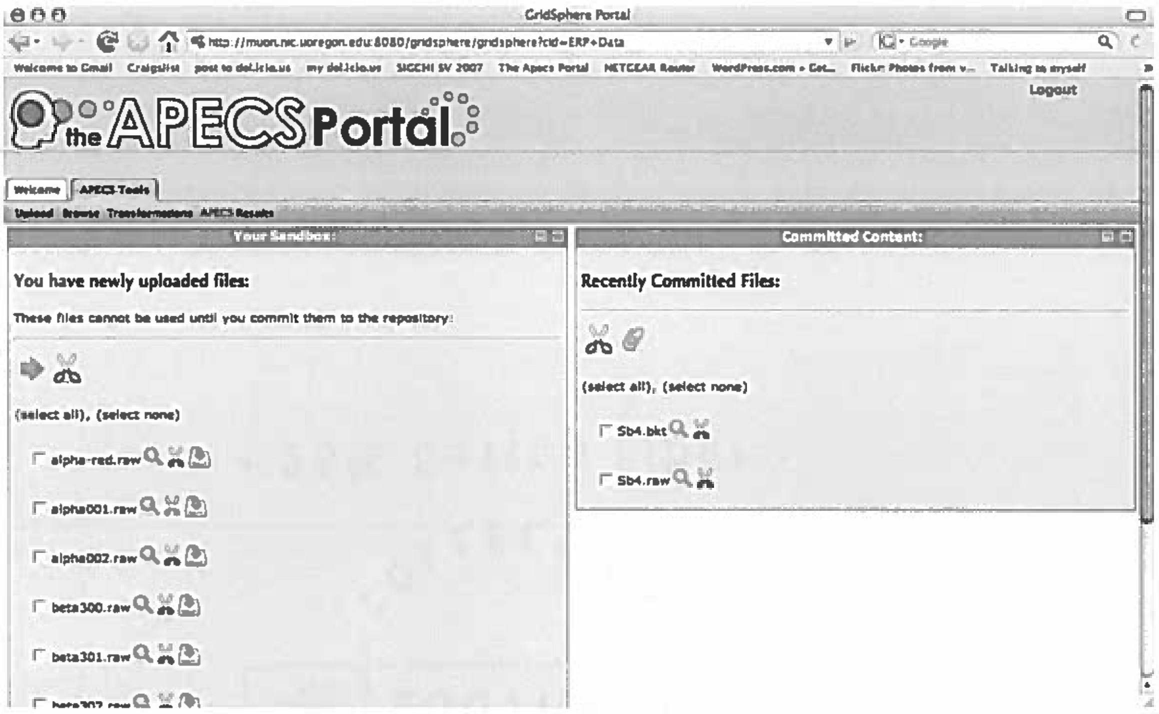

System Components
Apache Tomcat
GridSphere Portal Container
Portal Daemon
DB Interface
Repository Interface
Java Portlets
Dynamically generated JSPs
Requests for U.I. data interactivity
The portal's web interface
GridSphere maintains a separate DB of user account information.
GridSphere DB
(usernames, passwords)
The daemon fetches database content, which can be rendered by the Java portlets.
The portal daemon scans the repository for newly upload files, and then extracts useful meta-data about those files.
The FTP server uses user accounts from the GridSphere DB.
Physical files: waveform (EEG/ERP) & non-waveform (blink templates, etc)
Apecs Portal DB
(waveform meta-data)
The APECS DB correlates waveform meta-data with 'blob' pointers to waveform files.
File Repository
(15+ terabytes)
Each user has a unique 'drop' folder in the repository.
FTP Server
The FTP server provides a gateway to the repository.

FIGURE 5.1. The APECS Portal system design.



You have newly uploaded files:

These files cannot be used until you commit them to the repository:

(select all), (select none)

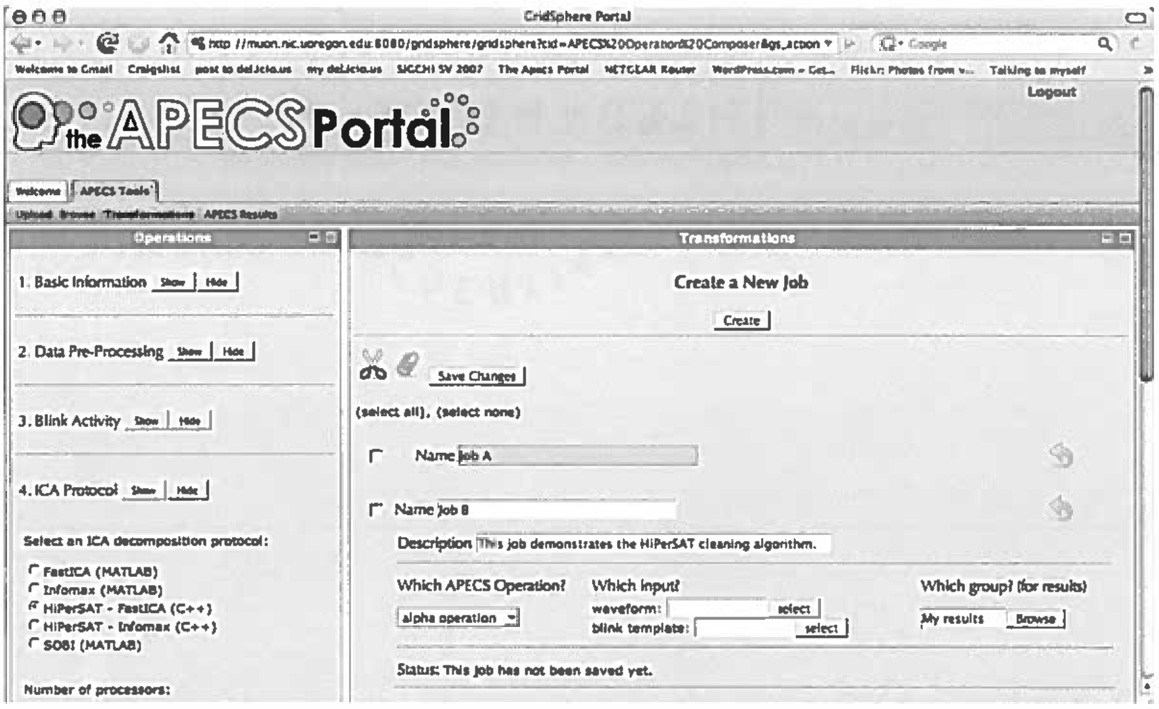
- alpha-red.raw
- alpha001.raw
- alpha002.raw
- beta300.raw
- beta301.raw
- beta302.raw

Recently Committed Files:

(select all), (select none)

- Sb4.bkt
- Sb4.raw

FIGURE 5.2. A screenshot from the APECS Portal, showing file import. The APECS Portal includes tools for uploading files, inspecting waveform meta-data, and then committing files to the repository.



Select an ICA decomposition protocol:

- FastICA (MATLAB)
- Infomax (MATLAB)
- [✓ HiPerSAT - FastICA (C++)]
- HiPerSAT - Infomax (C++)
- SOBI (MATLAB)

Number of processors:

Name Job A

Name Job B

Description This job demonstrates the HiPerSAT cleaning algorithm.

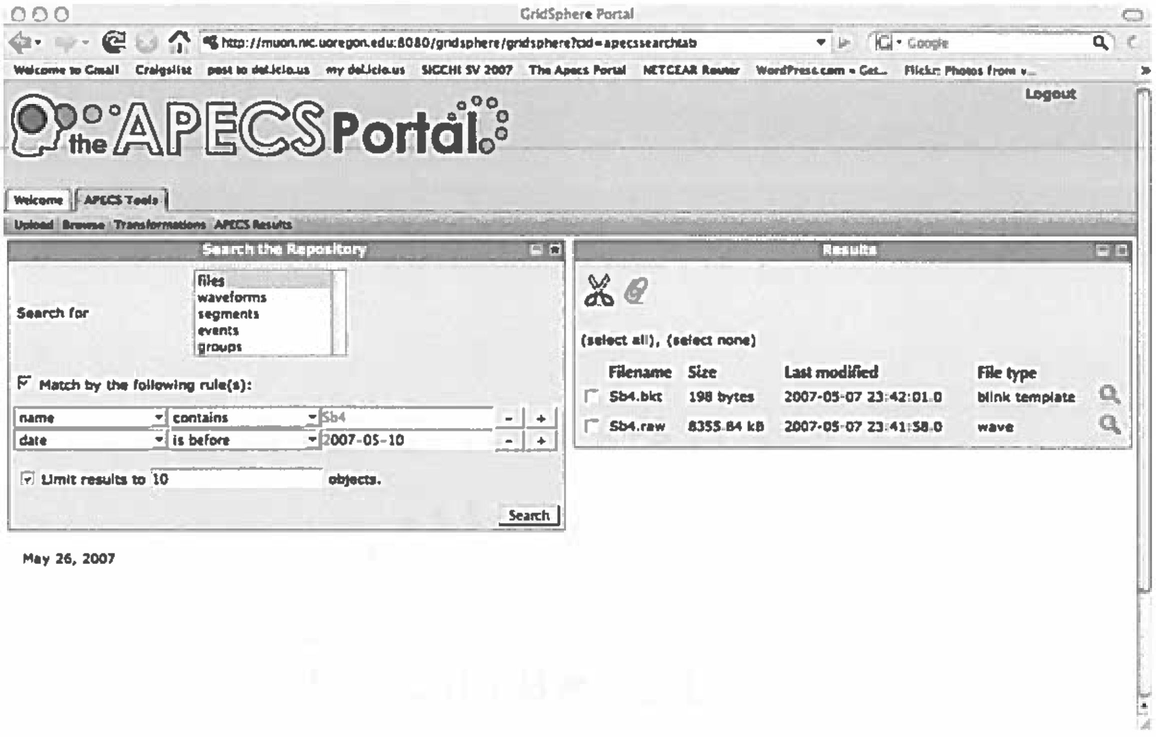
Which APECS Operation? alpha operation

Which input? waveform: blink template:

Which group? (for results) My results

Status: This job has not been saved yet.

FIGURE 5.3. A screenshot from the APECS Portal, showing job composition tools. The collaboratory includes tools for composing APECS jobs, which clean eyeblinks from waveform data.



http://muon.nic.uoregon.edu:8080/gridsphere/gridsphere?cid=apecssearchtab

Search for: files, waveforms, segments, events, groups

☑ Match by the following rule(s):

name	contains	Sb4
date	is before	2007-05-10

☑ Limit results to 10 objects.

May 26, 2007

(select all), (select none)

Filename	Size	Last modified	File type
Sb4.bkt	198 bytes	2007-05-07 23:42:01.0	blink template
Sb4.raw	8355.84 kB	2007-05-07 23:41:58.0	wave

FIGURE 5.4. A screenshot from the APECS Portal, showing search tools. The APECS Portal includes an EEG/ERP search engine.

5.1 The Next Step for APECS

Iterative Evaluation

The most important next step is to perform an iterative evaluation of the APECS portal, with real users. Towards this goal, the APECS Portal is undergoing an informal evaluation with data from an actual EEG study [21]. Preliminary results suggest the portal needs additional tools for managing information related to human subjects.

5.1.1 Provenance Visualization

The current APECS database tracks provenance, but very little of this information is distilled to the user. Future work remains to visualize provenance. The VisTrails project, from the SCI Institute, demonstrates a novel solution to visualizing hierarchically organized data with multi-user access [16]. It is easy to imagine this visualization strategy applied to EEG/ERP provenance data.

5.1.2 Distributed File Storage

Although the current APECS repository provides global access to a shared EEG/ERP database, it does not overcome limitations due to upload delays. For example, transmitting a bundle of EEGs from Pittsburgh, PA to Eugene, OR can take one full day (and sometimes longer). A more optimal approach would federate two separate repositories: one in Eugene and another in Pittsburgh. This proposed system makes the assumption that a distributed research team primarily wants to share meta-data about files, and not necessarily the files themselves. In a federated storage system, files can be physically stored on one of several repositories. A middleware package, such as [14], manages a global namespace over the collection of storage systems.

5.2 Future Work in Portlet Design

5.2.1 Improved AJAX Prefetch Strategies

Despite the design strategies we discussed in chapters three and four, web interactivity lags behind conventional interactivity. We can narrow this gap with solvent design patterns and better technology. The portlet paradigm presents one solution, and it can be improved with AJAX. An even more sophisticated approach would request content from the server before it is actually needed, otherwise known as prefetching.

A strong body of research explores web prefetching. In previous work, a web-server embedded prefetch directives into an HTML document [19]. The client's web-browser scheduled the fetch based on its own heuristics of network latency. Another study used a variation of the prediction by partial-matching algorithm to intelligently predict which content to request [15]. These previous techniques focus on fetching objects which are tagged and known. However, AJAX documents pose a new challenge. In an asynchronous environment, we need to prefetch objects which do not yet exist in the page. To support branching tasks, we also need to prefetch objects whose identity is determined at runtime.

As far as we know, the challenge of prefetching asynchronous web content is sparsely addressed in literature. Easy solutions are obstructed by the AJAX environment. In Javascript, AJAX requires developers to use continuation-passing and callback functions. However, recent work proposes a Javascript language extension (named MapJAX) to encapsulate the AJAX paradigm into objects and threads [27].

We can take steps towards automating the web-application engineering process by continuing to explore improvements in web browsers and scripting languages. For example, the XMLHttpRequest API desperately needs an automatic way to bundle concurrent requests into a single request. The decision about when to send a bundle of requests versus a single request should be based on the interactive context and the realtime performance of the network. Another area of future exploration is HTML "branching hints." Developers could use these "hints" to tag the sections of their markup whose specific content cannot be determined until runtime. Upon parsing these tags, the web browser could predictively (and asynchronously) fetch one or more alternatives.

5.2.2 User-Created Tools

The APECS Portal is essentially a container of portlets, and each portlet is a collection of lesser widgets. Every widget uses the same API to access underlying services. In a more sophisticated portal, the API would be public and users could design their own portlets and widgets. It is possible that user-created tools could aggregate neuroscience content in surprising and novel ways. We can extend the idea of user-created tools by imagining a web-based widget IDE. Future work might explore portlet-based development environments.

APPENDIX

DATABASE SCHEMA

```
create table files( file_id INT NOT NULL PRIMARY KEY,
file_content BLOB NOT NULL,
filename TINYTEXT NOT NULL,
size BIGINT NOT NULL,
modified_date TIMESTAMP,
upload_date TIMESTAMP,
comm_date TIMESTAMP,
format VARCHAR(20) NOT NULL,
committed BOOLEAN NOT NULL);

create table subjects( subject_id INT NOT NULL,
file_id INT NOT NULL);

create table waveform_references( ref_id INT NOT NULL PRIMARY KEY,
file_id INT NOT NULL,
version_num INT,
segmented BOOLEAN,
recorded_date TIMESTAMP,
rate INT,
num_channels INT,
board_gain INT,
conversion_bits INT,
amp_range INT,
num_events INT,
num_samples INT,
num_samples_per_segment INT,
num_categories INT,
num_segments INT);

create table events( event_id INT NOT NULL PRIMARY KEY,
ref_id INT,
event_code INT,
label VARCHAR(20),
type VARCHAR(20),
track VARCHAR(10),
onset VARCHAR(16),
duration INT);

create table segments( seg_id INT NOT NULL PRIMARY KEY,
ref_id INT NOT NULL,
seg_number INT NOT NULL);

create table categories( cat_id INT NOT NULL PRIMARY KEY,
cat_name VARCHAR(15),
ref_id INT);

create table averaged_waveforms( ref_id INT NOT NULL PRIMARY KEY,
num_trials INT,
num_segments INT);

create table operation_types( typeid INT NOT NULL,
       name TINYTEXT);

create table operation_descriptions( op_id INT NOT NULL PRIMARY KEY,
typeid INT NOT NULL,
name TINYTEXT,
description TINYTEXT,
lastmodified TIMESTAMP);

create table operation_contents( op_id INT NOT NULL,
field VARCHAR(40),
val TINYTEXT,
PRIMARY KEY(op_id, field) );

create table apecs_job_status( transid INT NOT NULL PRIMARY KEY,
lastupdated DATE,
status VARCHAR(30) );

create table apecs_run_history (run_id INT NOT NULL PRIMARY KEY,
owner_id VARCHAR(32) NOT NULL,
transid INT NOT NULL,
rundate TIMESTAMP,
result_group_id INT);

create table operation_inhooks( op_id INT NOT NULL,
hook_id INT NOT NULL,
input_type VARCHAR(20),
quantity INT,
PRIMARY KEY(hook_id) );

create table operation_outhooks( op_id INT NOT NULL,
hook_id INT NOT NULL,
  output_type VARCHAR(20),
  quantity INT,
PRIMARY KEY(hook_id) );

create table transformation_input(trans_id INT NOT NULL,
input_id INT NOT NULL,
hook_id INT NOT NULL,
PRIMARY KEY(trans_id, input_id, hook_id) );

create table transformation_output(trans_id INT NOT NULL,
output_id INT NOT NULL,
hook_id INT NOT NULL,
PRIMARY KEY(trans_id, output_id, hook_id) );


create table transformation_descriptions( trans_id INT
NOT NULL PRIMARY KEY,
opid INT NOT NULL,
description TINYTEXT,
name TINYTEXT,
lastmodified TIMESTAMP,
type VARCHAR(20),
savetogroup TINYTEXT );

create table transformation_save_history( trans_id INT
NOT NULL PRIMARY KEY,
lastsaved TIMESTAMP );

create table cleaning_jobs( job_id INT NOT NULL PRIMARY KEY,
trans_id INT NOT NULL,
job_name TINYTEXT,
description TEXT,
segment BOOLEAN,
number_of_seg INT,
elim_lowvar BOOLEAN,
lowvar_thresh FLOAT,
use_veog BOOLEAN,
use_blinktemplate BOOLEAN,
use_range BOOLEAN,
min_correlation FLOAT,
min_threshold FLOAT,
max_threshold FLOAT,
inc_threshold FLOAT,
whiten BOOLEAN,
validate_whiten BOOLEAN,
store_white BOOLEAN,
use_sphering BOOLEAN,
output_weight_mat BOOLEAN,
output_sphering BOOLEAN,
output_mixing BOOLEAN,
output_unmixing BOOLEAN,
output_ind BOOLEAN);

create table object_groups( group_id INT NOT NULL PRIMARY KEY,
group_name TINYTEXT,
description TEXT);

create table group_contents( group_id INT NOT NULL,
object_id INT NOT NULL);

create table owners( userid VARCHAR(32) NOT NULL,
objectid INT NOT NULL);

create table user_permissions( userid VARCHAR(32) NOT NULL,
objectid INT NOT NULL,
visible BOOLEAN NOT NULL,
modifiable BOOLEAN NOT NULL);

create table group_permissions( userid VARCHAR(32) NOT NULL,
groupid INT NOT NULL,
visible BOOLEAN NOT NULL,
modifiable BOOLEAN NOT NULL);

create table trash( objectid INT NOT NULL );
```

BIBLIOGRAPHY

[1] Adobe flash. http://www.adobe.com/products/flash/about/.

[2] The django project. http://www.djangoproject.org.

[3] Electro geodesics, inc. http://www.egi.com.

[4] Exo platform. http://www.exoplatform.com.

[5] Icablink toolbox [computer software] (version 1.21). Available from http://www.people.ku.edu/ jdien/downloads.html.

[6] Java applets. http://java.sun.com/applets/.

[7] Java web start. http://java.sun.com/products/javawebstart/.

[8] Liferay portal. http://www.liferay.com.

[9] Ruby on rails. http://www.rubyonrails.org.

[10] The world wide web consortium's working draft of the xmlhttprequest api. http://www.w3.org/TR/XMLHttpRequest/.

[11] A collaborative informatics infrastructure for multi-scale science. *Proceedings of the Challenges of Large Applications in Distributed Environments (CLADE) Workshop*, pages 24–33, June 2004. Honolulu, HI.

[12] R. Allan, D. Chohan, and X. Yang. A Service oriented architecture for portals using portlets. In A. Akram, editor, *UK e-Science All Hands Conference 2005*, September 2005.

[13] V. Astakhov, A. Gupta, S. Santini, and J. S. Grethe. Data integration in the biomedical informatics research network (birn). In B. Ludäscher and L. Raschid, editors, *DILS*, volume 3615 of *Lecture Notes in Computer Science*, pages 317–320. Springer, 2005.

[14] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The sdsc storage resource broker. In *CASCON '98: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, page 5. IBM Press, 1998.

[15] C. Bouras, A. Konidaris, and D. Kostoulas. Predictive prefetching on the web and its potential impact in the wide area, 2004.

[16] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: visualization meets data management. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 745–747, New York, NY, USA, 2006. ACM Press.

[17] D. Dou, G. Frishkoff, J. Rong, R. Frank, A. Malony, and D. Tucker. Development of NeuroElectroMagnetic Ontologies (NEMO): A Framework for Mining Brain Wave Ontologies. In *(To appear) Proceedings of the 2007 ACM SIGKDD conference (SIGKDD07)*, 2007.

[18] M. Fabiani, G. Gratton, and M. Coles. *Handbook of Psychophsiology*, chapter 3. Cambridge University Press, 2000.

[19] D. Fisher and G. Saksena. Link prefetching in mozilla: a server-driven approach. pages 283–291, 2004.

[20] R. Frank and G. Frishkoff. Automated protocol for evaluation of electromagnetic component separation (apecs). *Clinical Neurophysiology*, 118(1):80–97, 2007.

[21] G. A. Frishkoff. Hemispheric differences in strong versus weak semantic priming: Evidence from event-related brain potentials. *Brain and Language*, 100:23–43, January 2007.

[22] G. A. Frishkoff, D. Tucker, C. Davey, and M. Scherg. Frontal and posterior sources of event-related potentials in semantic comprehension. *Brain Research: Cognitive Brain Research*, 20(3):329–354, 2004.

[23] J. Grethe, C. Baru, and e. a. A. Gupta. Biomedical informatics research network: Building a national collaboratory to hasten the derivation of new understanding and treatment of disease. *In Proceedings of Healthgrid 2005, from Grid to Healthgrid*, pages 100–109, April 2005.

[24] R. T. Kouzes, J. D. Myers, and W. A. Wulf. Collaboratories: Doing science on the internet. *Computer*, 29(8):40–46, 1996.

[25] H. Liang, J. Dingel, and Z. Diskin. A comparative survey of scenario-based to state-based model synthesis approaches. In *SCESM '06: Proceedings of the 2006 international workshop on Scenarios and state machines: models, algorithms, and tools*, pages 5–12, New York, NY, USA, 2006. ACM Press.

[26] H. Muccini. Detecting implied scenarios analyzing non-local branching choices. In M. Pezzè, editor, *FASE*, volume 2621 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2003.

[27] D. Myers, J. Carlisle, J. Cowling, and B. Liskov. Mapjax: Data structure abstractions for asynchronous web applications. In *Proceedings of the 2007 USENIX Annual Technical Conference*, Santa Clara, CA, June 2007.

[28] A. Proverbio and A. Zani. *The cognitive electrophysiology of mind and brain*, chapter 2. New York: Academic Press, 2002.

[29] J. Rong, D. Dou, G. Frishkoff, R. Frank, A. Malony, and D. Tucker. A Semi-automatic Framework for Mining ERP Patterns. In *(To appear) Proceedings of The 2007 IEEE International Symposium on Data Mining and Information Retrieval (IEEE DMIR-07)* , 2007.

[30] M. B. Rosson and J. M. Carroll. *Usability engineering: scenario-based development of human-computer interaction.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[31] J. Sprague, E. Doerry, S. A. Douglas, and M. Westerfield. The zebrafish information network (zfin): a resource for genetic, genomic and developmental research. *Nucleic Acids Research*, 29(1):87–90, 2001.