Interface Design and Implementation of a Collaborative Observatory for Natural Environments

Bryce Lee

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-65
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-65.html

May 21, 2008

Copyright © 2008, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Interface Design and Implementation of a Collaborative Observatory for Natural Environments

by Bryce Lee

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Ken Goldberg, Professor

Research Advisor

(Date)

* * * * * * *

Dan Garcia, Lecturer SOE

Second Reader

(Date)

Dedication

This report is dedicated to my family, whose support and encouragement have been pillars in my life. My parents' faith in me and my sisters' unwavering interest in my technical ramblings helped me through the most daunting tasks. My grandmother's hard work and sacrifice have motivated me to pursue my goals with diligence and determination. I am fortunate to have these selfless people in my life.

Abstract

Research of natural environments often requires extensive field studies beyond the capacity of a single individual. In CONE Welder, we explore harnessing the power of human computation to aid ongoing research on avian activity and possible links to climate changes. Our approach utilizes a singly deployed pan-tilt-zoom web camera to gather observational data. The camera supports multiple operators at any given time. Our frame selection technique weights requests by a dissatisfaction quotient, which quantifies how well a request has been fulfilled since submission. We calculate fulfillment by considering video-specific expectations, including latency, coverage time, and coverage area. Each participant can take snapshots of the live video feed, generating a photographic history of the site's activity. An accompanying game extracts information from these photographs, creating a self-sustaining ecosystem for players and researchers. Players identify animals in a photograph by defining zones, or rectangular selections. Each zone is an opportunity to earn points by correctly classifying the enclosed species. When a consensus classification has been reached, the game rewards the original classifiers. Our scoring model extends beyond classifications, awarding behaviors and activities based on how closely they match our researchers' desires. We deliver all interaction through a unified Flash-based client, which can run on many common computing platforms. This client, in conjunction with our video relay server, implements an enhanced M-JPEG decoder that efficiently extracts frames. Deployed in Spring 2008, the project has already produced a vast quantity of data during its short lifetime. Over 200 participants have taken nearly 2,400 photographs and classified 28 species within them. Participants have provided good temporal coverage of the site, documenting animal activity over all hours. Consensus classifications have identified some species of particular interest to our researchers. This paper describes the key concepts, interface design, and implementation details of CONE Welder. We conclude with early results from our launch and a discussion of future directions.

Contents

1 Introduction **6**
- 1.1 Related Work . . . 6
- 1.2 Previous Experience . . . 8
- 1.3 Goals . . . 9
- 1.4 Collaborative Camera Control . . . 10
- 1.5 Game Formulation . . . 11

2 Architecture Overview **12**
- 2.1 Camera . . . 13
- 2.2 Server . . . 14
- 2.3 Client . . . 14

3 Client Interface **15**
- 3.1 Camera . . . 16
- 3.2 Classification . . . 16
- 3.3 Statistics . . . 18

4 Flash M-JPEG Video Player **18**
- 4.1 Design . . . 19
- 4.2 Frame Detection Optimization . . . 20
- 4.3 Results and Discussion . . . 21

5 Communication **22**
- 5.1 Camera Status . . . 22
- 5.2 Camera Control . . . 23
- 5.3 Web Framework . . . 24

6 Game Model **25**
- 6.1 Design . . . 26
- 6.2 Rating-based Awards . . . 26
- 6.3 Time-based Awards . . . 27

7 Experiments **27**
- 7.1 Data Generation . . . 28
- 7.2 Classifications . . . 29

8 Conclusion and Future Work **30**

Acknowledgement **31**

A Interface Concepts **33**

B Communication Socket **34**

C Sample Snapshots **35**

References **36**

1 Introduction

Observing natural environments can be a challenging task. Areas of interest are often hard to reach and provide limited connectivity to the outside world. Additionally, weather conditions can subject the observer to a dangerous work environment. The activity can strain the participant's mental endurance with long bouts of solitude and vigilance. Collaborative Observatories for Natural Environments (CONE) explores the use of tele-robotic cameras as an alternative medium for observations. Initiated by Song and Goldberg in 2005 [7], the project fosters an online community around a virtual observatory. CONE investigates methods to best satisfy simultaneous access to limited resources, such as camera control and video feed. The project is also interested in ways to leverage large groups of users to extract high-level information from collected data.

CONE Welder is the 5th incarnation of the CONE project and represents a joint effort among UC Berkeley, Texas A&M, the Smithsonian Institute, and the Welder Wildlife Foundation. Deployed at the Welder Wildlife Refuge in Sinton, Texas, the project helps investigate possible shifts in the breeding ranges of subtropical birds. Supported by recent sightings of the Green Jay and similar species, these shifts could be linked to global climate changes [6]. However, proof of this correlation requires evidence more stringent than anecdotal sightings. CONE Welder's objective is to systematically collect data to support ongoing research efforts. The project's public accessibility through the Internet supports a large user base spanning multiple time zones, significantly increasing the temporal coverage of the area. An accompanying game encourages participants to collaboratively process pictures, flagging potentially useful data for researchers.

Development and deployment of CONE Welder was the collaborative effort of many individuals from the aforementioned institutions. In addition, it utilizes the some technology and best practices from its predecessor, CONE Sutro Forest. While we will provide a high-level overview of the entire system, this paper will primarily focus on the CONE Welder interface. As the interface architects, we were responsible for the complete redesign of the collaborative application and accompanying web client. These components comprise all logic and interaction around the collaborative camera system. We will discuss the motivating factors behind their development and explain underlying design choices. The paper will conclude with preliminary results and briefly explore possible extensions to the system.

1.1 Related Work

CONE builds upon Song and Goldberg's ShareCam, [15, 16], a project that explored algorithms to effectively share control of a single camera resource and investigated using panoramas to specify camera movement. ShareCam branched from experiments looking into multi-user control of telerobots [11] and later human subjects [12]. Song has since

extended panorama interfaces for CONE's sister projects. His recent work utilizes a static panorama to access historical video footage through frame requests [17]. The CONE architecture is the result of several past experiments. The Demonstrate project [3], which allowed users to control a camera overlooking UC Berkeley's Sproul Plaza, investigated scalability concerns. Another early prototype, located at a residential location near the San Francisco bay, explored relatively inaccessible deployments. Finally, CONE Sutro Forest, discussed in the next section, significantly improved the server and client software [14, 10]. This system was a starting point for this project.

Chong et al. propose a classification system for telerobotic operation, based on the number of involved operators and robots [9]. CONE Welder would be considered a multiple-operator single-robot (MOSR) system under this taxonomy. MOSR-related research expands across multiple disciplines, including mining operations by Wilkinson [20]. A multiple-operator multiple-robot (MOMR) system would be an interesting future path for CONE. The addition of more robots would allow a mixture of collaboration and competition. Liu et al. [13] explore a similar situation in a game involving two robotic arms, controlled by multiple users through the Internet.

Chen et al. combine bird-watching and networked cameras in the Bird-Watching Learning system (BWL) [8]. While both systems have academic objectives, there are several key differences between CONE Welder and BWL. First, BWL participants must be physically present at the observation site and use camera-equipped PDAs. Furthermore, BWL explores using the network to classify images at a central location. We believe CONE Welder's telepresence and classification provide more accessibility and interaction with the environment. However, BWL explores administrative tools for curriculum around bird data, a possible extension for CONE Welder.

CONE Welder's zoning and classification game has the same goals as ESP [18] and PeekaBoom [19] by von Ahn et al.: identify and label objects in pictures. Both systems are publicly available and rely on cumulative knowledge. However, our project's overarching objective led to significant design differences. First, we have a finite set of labels, allowing features not possible under ESP. For example, we provide visual hints for labels from a reference set, helping users confirm their selections. Also, species-specific labels establish identification granularity, avoiding confusion from nested areas (such as body parts vs. the entire animal). Furthermore, PeekaBoom makes different assumptions about users. PeekaBoom depends on a general familiarity with objects pictured in the data set, using conflicting classifications as an indicator of possible mislabeling. We cannot make the same assumption in our system; a user may be familiar with only a subset of birds. However, we make the more useful assumption (for our usage) that users should be able to identify living animals and allow users to define zones without classifying them. Another key difference between the systems is the intended audiences. von Ahn's work aims at producing freestanding classifications. As a result, he implements many safeguards, such as IP checking, and does not publicly reveal consensus classification results to the players. The target audience for CONE Welder's results is researchers and players. The data serves

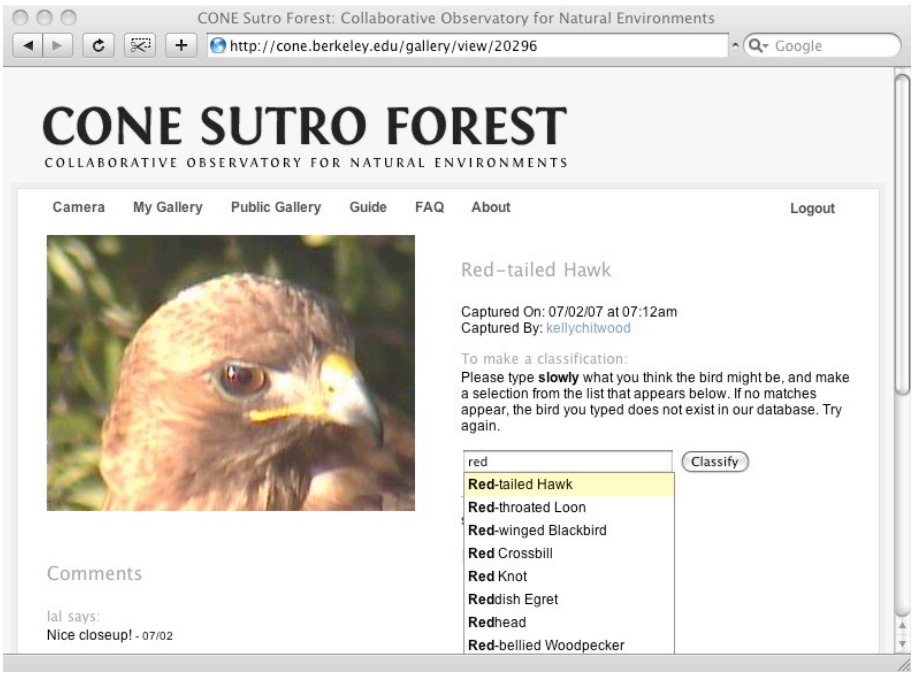


Figure 1: CONE Sutro Forest Species Classification Interface

as a helpful research aid, providing better coverage at the possible expense of accuracy. These relaxed expectations help us better address the users themselves, who have a vested interest in the data set as photographers. While we reduce outside influences by hiding voting details and other information, we take some liberty in our data exposure so that the users can enjoy the results of their collaborative work.

1.2 Previous Experience

CONE Welder's predecessor, CONE Sutro Forest (CONE-SF), was deployed in the backyard of Craiglist founder Craig Newmark's residence in Spring 2007. The camera overlooked Sutro Forest, a wooded area nestled in San Francisco, California. The project used a completely rewritten software stack, designed to improve compatibility and handle the demands of much larger audience. During its seven-month lifespan, CONE Sutro Forest attracted visitors from over 90 countries and was eventually translated into Chinese. 3,200 registered users took 27,000 photographs with a single web camera and collaboratively classified 30 unique species with the interface shown in Figure 1. The project enjoyed a very loyal following with returning users and even a blog dedicated to the project and daily activities.

We selected Ruby on Rails for CONE-SF's web framework due to its portability and database abstraction. The framework's self-contained nature helped us develop the

application offline through virtual machines. Also, its Active Record technology aided us in writing associations and behaviors on top of otherwise raw data, such as photographs. However, we encountered significant scalability issues after launching the project. The convenience of automatic data associations became a mixed blessing, causing extraneous table joins for even the simplest queries. As the number of photographs and other generated data increased, request fulfillment grew to several minutes. These long requests caused instances in our server cluster to go awry, starving other processes and eventually crashing the system. To mitigate this issue, we implemented a caching scheme for commonly requested data. While caching greatly reduced the server deadlocks, this functionality resided outside the main code base, duplicating much of our application logic and complicating future changes.

Users accessed the web camera through a Java applet, which streamed a live video feed and submitted movement and snapshot requests. The applet excelled at providing cross-platform compatibility and overcame many previously encountered security obstacles. However, the interface provided no direct communication between users, an often-requested feature in user feedback. Users found creative alternatives, like abusing the comments section on photographs for small talk. The Java applet also used very naïve algorithms to interact with the CONE camera server. Some of these design choices led to inefficiencies, such as unnecessary polling, while others caused undesired behavior. Finally, the java applet was a separate entity from the rest of the application, such as the photograph viewer and classifier. This division was due to the strong marriage of this functionality to the Ruby on Rails views. Users resorted to running multiple browser windows in order to simultaneously access all features.

We originally designed the accompanying game to encourage the use of all features. Based on two activities, the scoring metric was very transparent, perhaps too much so. Ten days after launch, an enthusiastic participant reverse-engineered the system and posted a very accurate explanation on his blog. The scoring system also ignored different conditions under which points were achieved, such as competition and skill. As a supplementary feature, the game was not well-represented in the database. We instead stored a single numerical value with each user, representing their total score. This value was updated whenever the user took a photo or classified an image correctly. No information was stored about the conditions under which the points were earned, such as award type and time earned. While sufficient for a few metrics, this system was not very extensible, especially with the previously mentioned ad hoc caching.

1.3 Goals

Based on this experience, we concluded the existing client application was unsuitable for CONE Welder. The system was too brittle to incorporate new features and would have issues supporting a larger user base. However, we decided to retain the CONE camera server. This software proved to be very stable under different loads, only failing once when

the network connection was physically severed. Additionally, the server interface was built around SOAP and other common standards, allowing us to easily adapt its functionality to a new client.

Our overarching goal in this redesign was to create a virtual observatory that engaged a wide range of users in bird-watching and related activities. To fully immerse users in this experience, we needed a client that incorporated all functionality into a unified interface. The client would have to allow greater interaction with the photographs and other users, generating useful data for both research and the game. We needed an accompanying back-end system that supported a very dynamic environment with context-dependent behaviors. The system had to also scale gracefully and seamlessly support new features and heterogeneous clients. Lastly, we wanted a game engine that would retain veteran users, while being receptive to new members. Scoring models needed to be flexible enough to translate researchers' requests into awarded behaviors.

1.4 Collaborative Camera Control

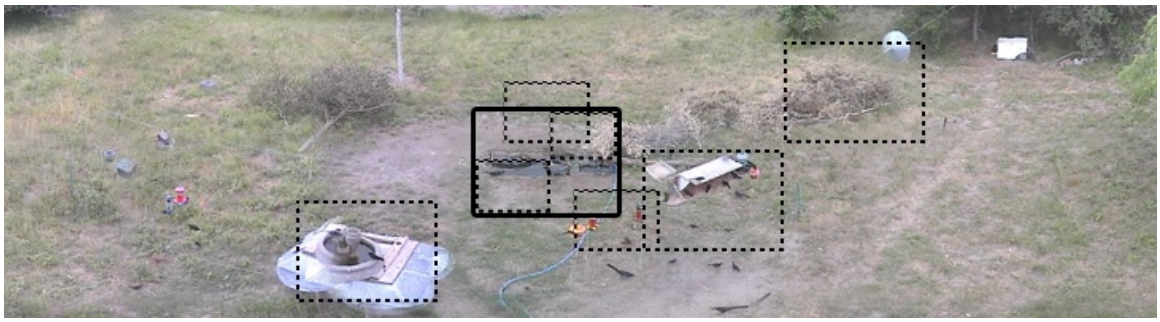


Figure 2: Frame selection among multiple requests, adapted from [10]. Dotted rectangles represent user frame requests r_i. The solid bordered rectangle indicates the frame f chosen by the camera. Our algorithm considers intersection of the area A_{r_i} enclosed by the requested frame and the area A_f enclosed by the selected frame.

An inherent problem of multiple-operator single-robot systems is the inability to completely satisfy conflicting requests. In our CONE system, the camera can only display one region to all users. Without proper request handling, interaction becomes a free-for-all struggle for control, leading to hostility and frustration. To prevent this undesired situation, MOSR systems must respond in ways that optimize satisfaction across all users. We approach this problem by solving the inverse problem, minimizing user dissatisfaction. In particular, we are interested in reducing the time taken to fulfill a request or "time-dissatisfaction". Ideally, more weight should be given to older requests. This metric is particularly useful in our system to prevent starvation, where requests are starved from being fulfilled indefinitely. We define request fulfillment as how well the current response

satisfies the original request. To quantify this amount for frame requests, we use intersection over maximum (IOM), defined in (1).

$$S(f, r_i) = \frac{P_i}{max(A_f, A_{r_i})} \tag{1}$$

In this equation, f is the camera's selected frame, r_i is request i's frame selection, A_{r_i} is the area enclosed by r_i, A_f is the area enclosed by f, and P_i is the intersection of A_f and A_{r_i}. The maximum value is achieved when the requested frame and selected frame are identical, resulting in a complete intersection. This maxima considers both frame location and zoom level, since each requester is expecting a level of detail with their coverage. Fulfillment also depends on coverage time. Ideally, we would provide coverage as long as desired. Since this is often not possible, we define a set duration of time that a request can receive attention before being considered fulfilled.

Schmidt and Dahl implement these metrics in the "d-weighted" frame selector, developed for CONE Sutro Forest. The system chooses a new frame that minimizes the current dissatisfaction over all current requests. Dissatisfaction is calculated by how poorly the selected frame fulfills the requested frames. This value factors in past fulfillment, using the request's previous dissatisfaction level as a scaling factor. A request's dissatisfaction will decrease during fulfillment, but incrementally increase for each frame selection where the requested frame is neglected. The system also considers previous fulfillment time, inflating IOM to reduce dissatisfaction as the frame receives more coverage.

1.5 Game Formulation

CONE Welder's main goal is to be a self-sustaining system, where producers (online participants) both gather data and generate useful information about it for our consumers (researchers) without direct interaction between the two parties. However, our consumers and producers have very different motivations for using the system. The system's producers seek to establish personas within the system. They work towards generating a virtual footprint with their archived snapshots and rankings. On the other hand, our consumers have very little interest in this virtual presence. They want actions and results over the data set, which may not be of particular interest to the producers.

We approach this disparity by defining an underlying game. This medium allows consumers to reward producers for desired behaviors, thus forming a symbiotic relationship. In this game, we represent a user's achievement through points. Points hold no meaning in any outside context. Since we want the game to be self-sufficient throughout the project's potentially multi-year lifetime, we do have achievement levels or any other absolute point references. We instead rely on the competitive spirit of individuals, using these counts to rank our producers. These points gain value through scarcity. Like currency, our game only has a limited number of points available at any given time. At the

same time, we must provide enough points in the system to award any producer who wants to contribute. We address this issue by encapsulating points in awards, which are based on activities defined over the gathered data set. As a result, the available number of points at any given time is directly related to the amount of gathered data, encouraging the continued production of new data.

Awards provide discrete representations of consumers' desires. We define the attainment and point value of each award as the combination of two factors: competition and skill. Competition is based on peer participation around the award and can include a variety of factors, such as temporal considerations. Skill is a relative assessment in our system. As a "netizen" application, our system defines skill based on consensus truths or assessments over the generated data set. Since the generated data sets were designed around our consumers' wishes, skill can be used as a satisfaction meter. Thus, awards provide a connection between the desires of the consumers and producers. We will later explore the award system and provide details on select implementations.

2 Architecture Overview

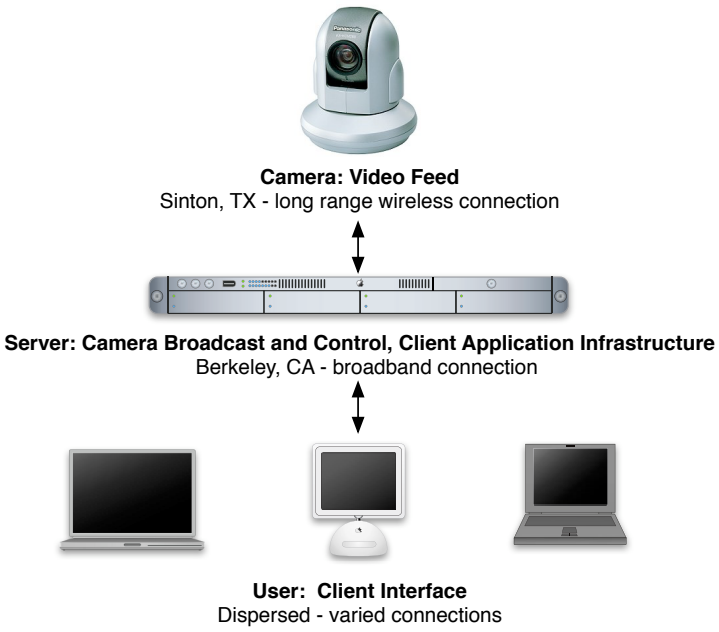


Figure 3: CONE Environment Layout

CONE consists of three major components: a single web camera, a multi-purpose server, and a cross-platform client interface. This design was very important for CONE Welder, since each component could be developed and maintained independently. The separation

allowed the UC Berkeley team to concentrate on application development, while our Texas A&M counterparts focused on the camera's deployment. The separation also enforced strict APIs, enabling isolated changes to the system. Even major revisions, such as a new application interface, required very little modification to other components. As shown in Figure 3, the camera and client do not directly interact with each other, instead passing through the server. By channeling communication through an intermediate source, we can improve performance and ease maintenance. For example, having the server broadcast the video feed greatly reduces the necessary camera bandwidth requirements. This layout also allows us to easily modify content and functionality at a single location, promoting thin-client development.

The following sections describe the CONE infrastructure in more detail, including responsibilities and implementation choices. In-depth explanations of the camera and associated server components (relay server, frame request handler) are beyond the scope of this report and will only be briefly discussed.

2.1 Camera

A single autofocus, autoaperature networked webcam lies at the heart of the CONE system, providing a live video feed of the remote setting. We chose the Panasonic KX-HCM280 Network Camera, which was also used for CONE Sutro Forest. The camera has a maximum resolution of 640 by 480 pixels and maximum frame-rate of 30 FPS. Due to limited available bandwidth at our deployment site, we reduced the resolution to 320 by 240 pixels to increase frame throughput. The camera's built-in IP server allows it to be deployed anywhere within reach of the Internet and a power source. Also, the native video format, MJPEG, uses intraframe compression. As a result, we can send each progressive frame to the client separately, simplifying our broadcasting mechanism. Finally, the camera has excellent pan and tilt capabilities, with a range of 350 and 210 degrees respectively. Combined with a 21x optical zoom (42x digital zoom), the KX-HCM280 can survey large areas and focus on small distant objects, such as birds.

Accessing the Internet was very challenging at the Welder Wildlife Refuge. Unlike previous CONE deployments, no landline connection existed. The closest source was a nearby administration building, which itself relied on long-range wireless broadband. To access this connection, we installed a Linksys WRT54G router nearby the camera to serve as a wireless network bridge. Since the building was outside standard wifi range, we also installed bidirectional antennas at each location to amplify the signal. The wireless overhead and contending traffic from the administration staff resulted in a bandwidth limit from the camera for streaming video of 300kbps (around 3 frames per second).

2.2 Server

Camera and client services reside on a quad-core 2GHz Opteron server with 4GB of RAM running CentOS. A socket-based relay helps overcome limited camera bandwidth by replicating the video stream. The server has exclusive access to the camera, providing the fastest video stream to this process. As new frames arrive, they are placed within a circular buffer. Connected clients are serviced in a round robin fashion, using the buffer to handle slower connections. Overhead is minimal and performance degrades gracefully as demand increases. An Apache AXIS web service exposes camera controls to clients. Requests are forwarded to a persistent kernel process, which applies collaborative filters (explained later) to form aggregate requests for the camera. The kernel is also responsible for providing status updates and running tools, such as an automatic panorama generator. CONE Welder introduces a third process, enabling live messaging between users. This new feature also allows socket-based access to status information for efficiency gains described later.

The server also hosts the client application infrastructure. We chose to implement the new version with cakePHP, an open-source web framework. Inspired by Ruby on Rails, cakePHP provides a very portable solution, running on servers with standard Linux-Apache-MySQL-PHP(LAMP) software stacks. In addition, the framework offers the same convenient database abstraction as Ruby on Rails, but delegates more power to the developer. Extensions enable the user to selectively toggle associations on demand, reducing the resulting table joins. Departing from the previous designs, the new infrastructure separates functionality from presentation. The system accepts standard POST and GET requests and returns results in a consistent XML format.

2.3 Client

A clean server API allowed us to develop a thin client with a richer interface. Adobe Flex, an open source SDK built around the Flash platform, was the ideal vehicle for this application. Flash can run within browsers like Java, but also benefits from broader compatibility, thanks to the popularity of YouTube [5] and other Flash-based services. Flex's built-in visual components offer the amenities of a standard desktop application, while its programming language (Actionscript) facilitates custom development. Most importantly, the framework offers access to low-level services, such as binary sockets and HTTP requests, required to interact with our server.

The level of interaction possible under Flex compelled us to reevaluate our design. Past efforts to overhaul the interface had failed, yet there were definitely key elements that had proven themselves in real world use. However, we felt a fresh design was necessary to properly integrate new functionality and provide a consistent metaphor. The rest of this paper is dedicated to the resulting interface, followed by an explanation of key implementation choices.

3 Client Interface

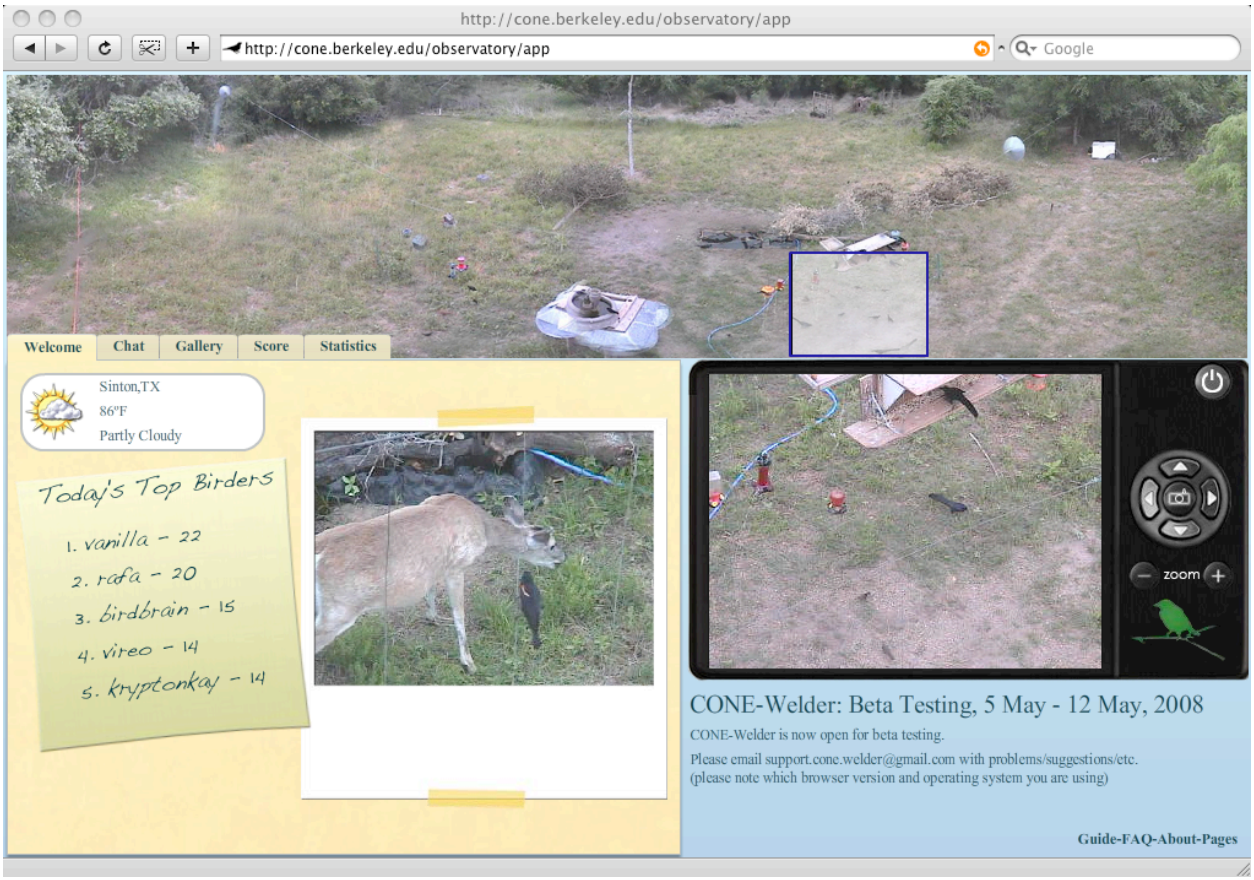


Figure 4: CONE Welder Client Interface with Activated Web Camera and Welcome Tab

We wanted users to perceive CONE Welder as a personal nature experience. The new interface, shown in Figure 4, pursues this vision by emphasizing the camera vantage point as a virtual lookout. A key feature is a large static panoramic backdrop of the entire view space. We were able to make the panorama much larger than previous versions due to a wider layout. Statistics from CONE Sutro Forest suggested over 96 percent of visitors having resolutions of at least 1024 by 768 pixels. In addition to establishing the scene's perspective, the larger format is also critical for our deployment site. As explained in the next section, the panorama serves a pivotal role in operating the camera. Unlike Sutro Forest, points of interests are both far away and widely dispersed. The large size allows users to better survey the entire region. Most major interaction is contained in two visual metaphors modeled after real-life objects: a digital camera and manila folder. The camera encapsulates video functionality, drawing upon the device's affordances to make the interface more intuitive. All non-volatile content is contained within the folder, an evolution from past multi-page layouts. We explored alternative metaphors during development, but these alternate designs (see Appendix A) posed workflow and layout

issues. For example, a book layout prototype was hard to navigate by flipping pages and hampered our design by dividing the space into facing pages.

3.1 Camera

To access the web camera, the user first presses the power button, causing live video to appear in the viewfinder. The initial off state helps conserve bandwidth for the server and clients more interested in archived data. Turning the camera "on" also activates a drawing surface over the panoramic backdrop. Users can request to focus on a specific region by clicking the center of the area of interest and dragging outwards. A rectangle of fixed-ratio dimensions, representing the requested frame, will appear and resize with mouse movements. The request is sent to the server once the user releases the mouse button. Directional and zoom buttons on the camera allow incremental changes to the requested frame. Other frame requests are displayed on the panorama in colors corresponding to the requester's screen name in the chat buddy list. CONE Welder uses intersection over maximum (IOM) with time-dissatisfaction, a technique developed for CONE Sutro Forest, to select a single frame from all requests. Chosen frames are also shown on the panorama with a distinctly thicker border.

The button with a camera icon allows users to capture still snapshots from the video feed. A popup message notifies the user if the snapshot was saved or if he or she had exceeded the daily snapshot threshold. We limit each user to 10 snapshots per day to help limit storage usage and force participants to be selective in their photography. Thanks to the split interface, a user can view the gallery and delete unwanted pictures without losing view of the camera. The design also enables the integrated chat room to be displayed next to the camera, allowing users to actively discuss the live feed.

3.2 Classification

Users can access all snapshots through the gallery tab, which presents them as a grid of projector slides. Each slide provides a brief summary of the contained photograph, including time taken and photographer. Since researchers may only be interested in photographs of a particular species, filters can be applied through a drop down menu. Users can also filter on other criteria, such as personal and unclassified photographs. To see photograph-specific details, a user double-clicks on the associated slide. The resulting view allows the user to add and read related comments. In addition, a series of stars allow the user to rate the photo, similar to functionality found on YouTube. This simple interaction helps identify novel images that may be of interest to researchers. A single classification is often not sufficient for an entire photograph, since multiple animals can be present. Also, a classification by itself does not specify the object's location, forcing the viewer to guess the intended target. CONE Welder resolves these issues with classification over zones, which

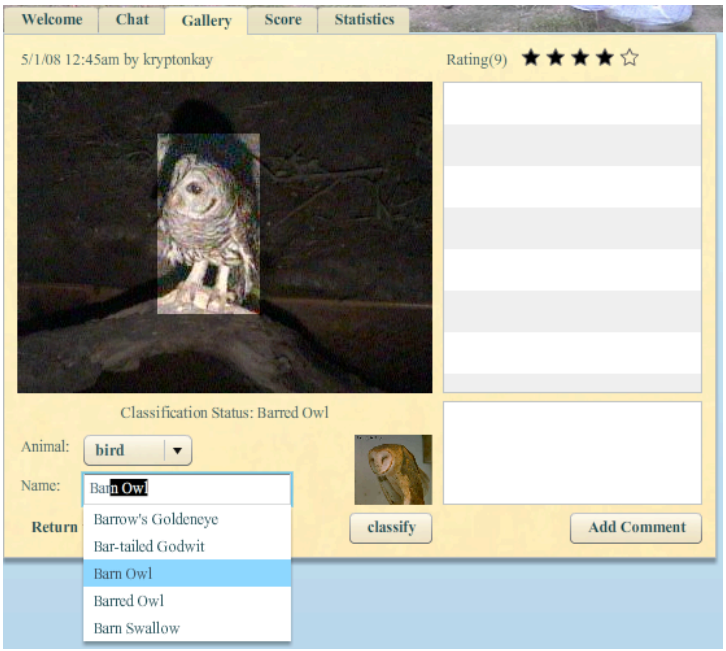


Figure 5: Example of Zone Classification

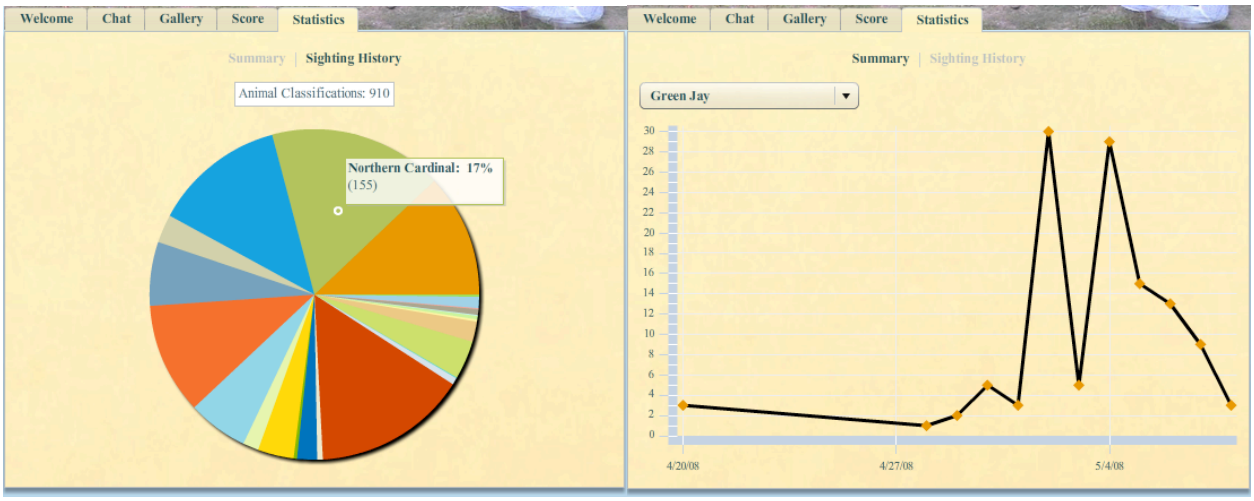
are user-specified bounding boxes on the image. Zones are similar to tagging features found in other photo services, such as Facebook [4] albums. To view zones, a user moves their mouse over the photograph. Defined zones will appear as rectangles on the image. If a zone is classified, the species name will show up when the mouse is over the zone's rectangle. Users add zones by clicking the "add zones" button in the details view, activating a drawing surface over the image. Interaction is similar to the panorama drawing surface, except there are no dimension constraints and multiple zones can be defined at a time. These looser constraints allow the user to create a rectangular selection by dragging in any direction on the surface.

To classify a zone, the user clicks the corresponding rectangle on the image. As shown in Figure 5, the surrounding area dims to highlight the region. Zone-specific comments are shown to facilitate more granular discussions. A classification panel below the photograph allows the user to select a general animal category (such as bird or insect) and specify the species' name. User submitted names could be problematic due to spelling variations. We avoid this issue by restricting input to a set defined for the selected animal group. A list of potential name completions appears as the user starts typing. When an item is selected from this list, a small profile image is shown for verification. After the user has selected the desired species, he or she clicks the "classify" button to submit their classification.

Submitted classifications help establish zone identifications through a voting scheme. When the total number of classifications exceeds a predefined threshold, the zone assumes the classification that has received more than 75 percent of the votes. If these conditions

are not met, the zone will remain unclassified. We limit each user to one vote per zone and do not reveal how others have voted. However, the accuracy of the produced consensus classifications is an open problem we hope to explore over time with our collected data.

3.3 Statistics



(a) Pie Chart of Species Classification Distribution (b) Graph of Green Jay Sightings vs. Time

Figure 6: Example Visualizations of Statistics

Visualizing data sets can often help researchers interpret results. CONE Welder explores the utility of this medium by generating charts from the classification data, shown in Figure 6. Our cakePHP application produces aggregate views of the data, such as classified species and sighting history, which we display in the client using Adobe's Flex Data Visualization library [1]. Similar visualizations are also available for users to view their score breakdown and history. The pie chart provides an overview of the classified species and allows viewers to quickly compare the relative number of sightings. The second chart uses the timestamps of classified photographs to plot the number of daily sightings for a particular species. A general sighting plot is also available to help identify general trends. These early designs are merely a starting point; future usage by researchers will help us develop more useful representations.

4 Flash M-JPEG Video Player

At the time of development, Flash only supported video streaming using the Flash Video Format (FLV). FLV is significantly different to M-JPEG, compressing across frames. While

FLV is becoming an increasingly popular format for newer web cameras, our camera only outputs in M-JPEG. To make a compatible video stream, we explored converting our camera feed from M-JPEG to FLV using external tools. Our early prototype, based on [2], involved a persistent FFMPEG process, which gathered frames through a socket connection to the camera and produced a single SWF file with the FLV stream. While we could now display the video in Flash, the solution had significant design flaws. First, the translation introduced noticeable latency, sometimes in the order of several seconds. A time shift would not have been an issue for non-interactive live video streams. However, the responsiveness of our camera controls and snapshot functionality relied on a quickly reacting video stream. In addition, our video relay server depended on self-contained frames in order to serve clients at different points in the stream. Since FLV is a proprietary format, we would have needed to use either Flash Media Server or an open source alternative for video broadcasting, introducing yet another external process.

The following sections describe the design and implementation of our M-JPEG compatible player. We will also cover possible optimizations and compare the performance with that of our old Java player.

4.1 Design

The core task of an M-JPEG player is the detection of individual frames within the data stream, encoded as JPEG images. We approached this task with a specialized buffer, evolved from earlier work on the Java M-JPEG player. At the simplest level, this buffer queues byte arrays passed in from outside sources. We chose to abstract this functionality away from the socket processing for added flexibility, which we explore in the next section. As bytes arrive, the buffer checks for two unique sequences, representing the beginning and end of the frame. It is important to note that it is not possible for an end sequence to arrive before the start sequence, since our relay server only sends whole frames. However, we included this check for compatibility with other sources. When a frame is detected, the buffer notifies a parent process, which controls the socket connection and visual elements. This process can request the detected frame, which is shifted off the buffer. The associated byte data is then passed to one of two *Image* instances. *Image* is a class in the Flex SDK, allowing bitmaps in common formats to be rendered in the applet's window. Once the image has loaded, the parent hides the *Image* object containing the previous frame, producing a primitive double buffer effect.

Our design considers Flash platform-specific behaviors. Due to its origin in animation, Flash uses a frame-based execution model. Within each frame, only one class of tasks can take place. This property divides the application into three revolving stages: code-execution, event processing, and rendering. Therefore, our components primarily communicate through publicly-defined functions, avoiding the delay incurred by event propagation. Another consequence of this execution model is a set upper limit on rendering. Unlike the Java implementation, we had to explicitly specify a frame rate in

order to meet the transmission rate. The Flash platform is also single-threaded, which eliminates the need for locking the buffer when adding and shifting data.

4.2 Frame Detection Optimization

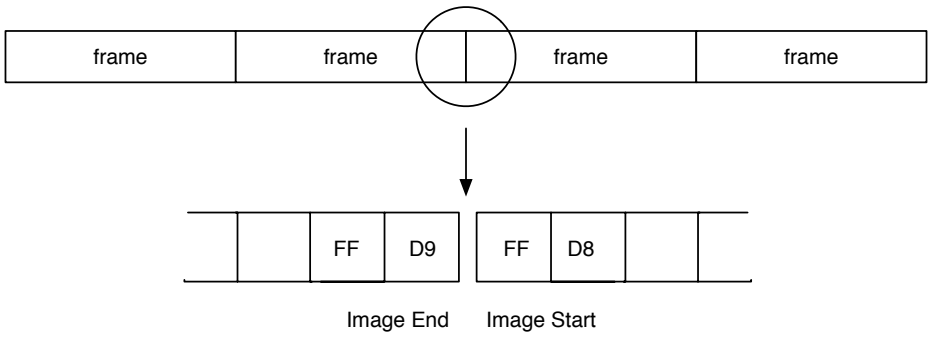


Figure 7: Diagram of M-JPEG byte stream.

Aside from rendering and socket transfers, frame detection accounts for the majority of the stream processing. Most of this traversal is done in vain, since we only want to find the image delimiters. These 4 bytes represent less than 0.01 percent of the average image size, around 20KB in our particular deployment. However, this traversal appears unavoidable, since the JPEG header does not contain a length attribute. Our optimizations explore methods to intelligently derive the length and reduce the number of comparisons needed.

To benchmark our optimized buffers, we used a 2.16GHz Intel Core Duo MacBook Pro with 2GB RAM running Flash Player 9.0.115.0 in Mac OS X Leopard 10.5.2. The test measured the CPU usage of the Flash Player process over 10 minutes at 1-second intervals. For our sample data, we used a feed from a dedicated relay server streaming video at around 30 FPS.

The first approach used past data as a heuristic for predicting future image lengths. This method relied on sequential images having similar byte sizes, which was often true for relatively static scenes. We tested this condition with a 1500 frame sequence of a street with moving cars and walking people. Variation was less than 3 percent of the average image length. Therefore, a previous frame's length was a good estimate of the position of the next frame's end sequence. We implemented this logic in our buffer by recording the byte length of previously detected frames. The buffer operated as previously described for detecting the initial frame. On all subsequent frames, the buffer waited until the queue matched or exceeded to previously recorded size. Once this condition was met, the buffer looked for the JPEG end sequence around this point. The search would pause if it did not find the delimiter within a certain interval or if it reached the end of the byte queue, resuming on the next round of execution with a larger search radius or when new data was

available respectively. When the end sequence was located, the buffer extracted the frame as before and updated the stored image length.

While benchmarks showed a 10 percent decrease in the CPU usage, the heuristic hurt performance when the scene changed dramatically, a very likely possibility when users were in control. The frame size from a new scene can be significantly larger than a previous location. This ratio is especially true at night, where completely black scenes have significantly smaller JPEG sizes than lit areas. Large recorded frame sizes from previous scenes caused the buffer to queue up multiple frames before searching for the end sequence, leading to significant frame rate degradation.

Our second approach explored using the CONE relay server to insert a custom size header. This addition was relatively simple, since the relay server already quantized the stream into individual frames. The length was prefixed to the JPEG byte array as a 32-bit integer. This addition had a negligible effect on size, increasing a 20KB frame by less than .02 percent. Since the relay server only sends whole frames, we modified the client buffer to extract this value once the first 4 bytes were received. The buffer shifted these bytes off the queue and waited until the queue grows to the read amount. Unlike the previous method, no search was necessary and we could immediately extract the frame data when available. This solution was backwards compatible with traditional M-JPEG decoders as long as they use both JPEG sequences to locate the frame. A more robust, variable-length solution with unique markers is necessary for backwards compatibility. If the frame size exceeded 63 KB, there was a possibility this header would conflict with the JPEG markers (FFD8 and FFD9).

This second approach matched the efficiency of the first approach in our benchmark tests. Since this method was resistant to changes in image size, we decided to use it for our production environment. We will next describe its performance compared to the previous Java version.

4.3 Results and Discussion

To determine the efficiency of our Flash implementation, we compared it to the Java player from CONE Sutro Forest. We used the benchmarks from in the previous section on the same reference machine. Achievable frame rate was equivalent in both implementations and therefore not included in these tests. The results, shown in Figure 8, demonstrated that the Flash application was slightly more efficient than the Java equivalent, with an average of 6 percent less CPU usage. While there was a marked difference in the testing environment, the actual frame rate from the CONE Welder camera was much lower. Therefore, the efficiency improvement in our deployment was not as noticeable.

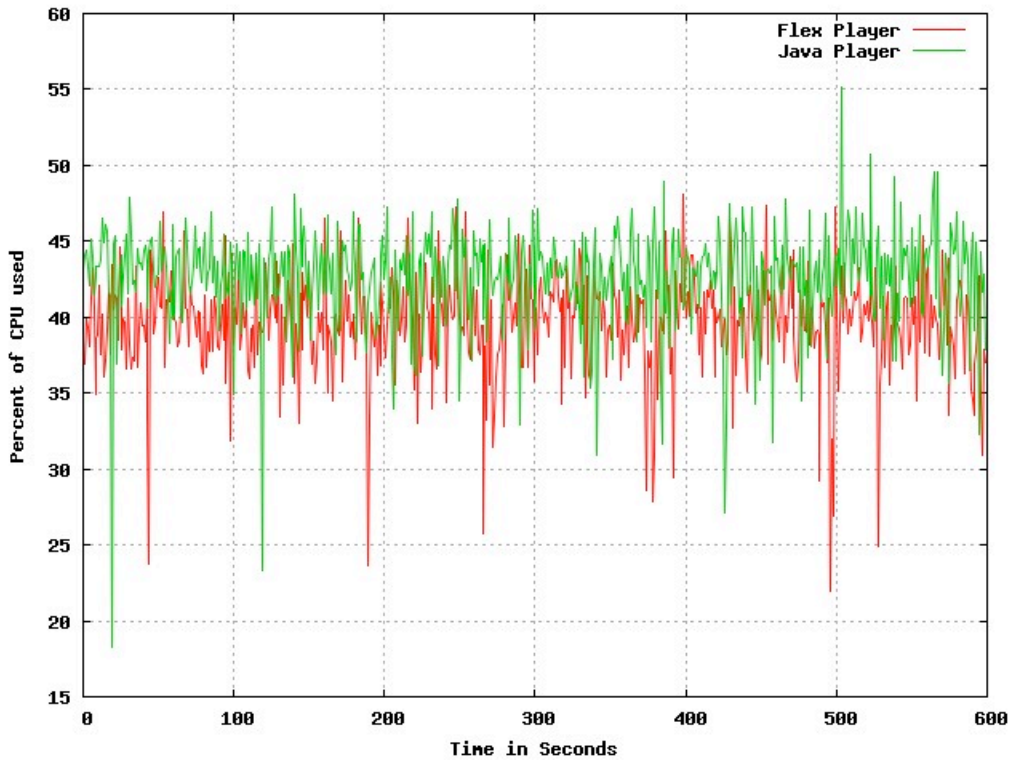


Figure 8: Comparison of Java and Flex M-JPEG player CPU usage

5 Communication

The interaction between the user and Flash client differs significantly from the underlying communication between the Flash client and server. Users perceive CONE Welder as robust experience and expect all requests to be processed. They expect status information to be up to date and the perceived state, such as their logged in status, to persist. However, the CONE server delivers a stateless experience through the camera SOAP service and cakePHP POST interface. The server does not attempt to remedy logical failures. This section explores enhancements to both the server and Flash client to better bridge these realities.

5.1 Camera Status

A key feature of the camera interface is the display of current camera position and frame requests. This information is drawn on the panorama to show what factors are influencing the camera movement. The original CONE server exposed this information through a SOAP web service. As a result, the CONE Sutro Forest client polled the service every five seconds for new information. However, status information can reach stabilization, defined

as the state where the camera position does not change and there are no outstanding requests. During this state, polling only introduces an unnecessary overhead.

We reduce this inefficiency in CONE Welder by extending the socket communication protocol used by the chat feature. This underlying server interaction is completely hidden from the user. When the CONE server receives a frame request through the SOAP service, it broadcasts a new status ready message to all connected clients, as detailed in Appendix B. If the client wants the new information, it responds with a status request message. The server then sends the status information over the socket. In the Flash implementation, a new status ready message causes the client to request updates every few seconds, mimicking the old polling interaction. The client also makes this request when it first starts. During this time, incoming status advertisements are ignored. When new status information arrives, the client compares the new values with the last update. If the data has stabilized as previously described, the client stops polling. Therefore this interaction provides current status information without constantly polling.

5.2 Camera Control

The camera control system logs out idle users after a set time period to better serve active participants. Logged out users can still view the video feed, but any frame requests are ignored. In the Java client, no logout messages were displayed, and users had to reload the client to restore camera controls. This state frequently confused participants, who were not aware of the logout action. Users expected every request to be made, regardless of their idle time. The Flash client implements a hidden interaction, built around the SOAP interface, to log in users when necessary to fulfill requests. When a user makes a camera request, a new camera request object is instantiated with the proper parameters and added to a queue. The client follows the logic described in Figure 9 while the queue is not empty. It examines the first queued request and makes the associated SOAP call. If the web service returns true, the client then shifts the queue and repeats the process. If the web service returns false, the client requests its login status from the server. This check helps identify calls that can logically fail, such as snapshots when the user exceeds their limit. In this case, a related message (such as "Snapshot Limit Exceeded") is shown to the client, and the process is restarted as previously mentioned. If the client finds out it is not logged in, a login request will be made, and it will reattempt the request upon login.

This method ensures that each user request is processed in order. As a consequence of the passive login scheme, the client does not have to explicitly log in users upon load. Therefore, camera control is better limited to active participants with this method.

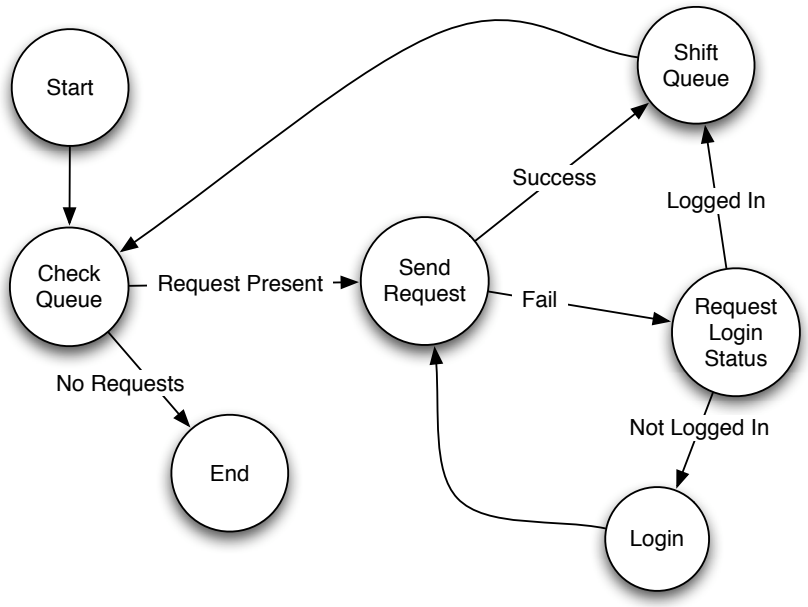


Figure 9: State Machine Diagram for Client-Side Camera Request Processing

5.3 Web Framework

Communication between the client and content server often contains more information than the requested data set. There may be associated error messages or system status flags to help better understand the result. For example, a request may fail due to the client's session key expiring. The client must be able to interpret this metadata and react consistently throughout the entire application. We approach this problem by establishing standard interfaces in both the cakePHP web application and Flash client.

```
<result success="1" message="Observation successfully retrieved">
  <observation photographer="vanilla" taken="5/11/08 8:44am"
                file="vanilla_1210520655.jpg" deletable="0">
    <ratings count="0" average=""/>
    <comments/>
    <subselections>
      <subselection id="2289" userName="vanilla" time_selected="2008-05-11 08:47:48"
                    left="4" top="52" width="232" height="180"
                    classification="unclassified"/>
    </subselections>
  </observation>
</result>
```

Figure 10: Example XML Result for Photograph Information

Our cakePHP application responds in an XML format, like the example in Figure 10, whenever the request path ends with a ".xml" extension. Routing on the request extension allows us to customize responses for different client classes in the future. The root element specifies system flags and messages as attributes. For consistency, we implement the XML generation as an application-wide template, allowing the current action to specify these attributes' values. The requested data set is included as a root child and abides by an XML format established for that particular action.

The Flash client requests and interprets these results through a single *TransactionEngine* instance, a custom component built around Flex's *HTTPRequest* class. By passing all requests through a single object, we can process them in a pipelined fashion. This feature is especially important for interfacing with the cakePHP framework, where high security settings cause the session key to change between requests. Also, this setup allows us to stop all outgoing requests at any point if needed. The Flex client can request data by passing a *TransactionRequest* to the *TransactionEngine*. Each *TransactionRequest* includes key-value request parameters, request mode (such as POST or GET), and a callback function to receive the results. When its request queue is not empty, the *TransactionEngine* translates the first *TransactionRequest* into an HTTP request and sends it to cakePHP application. Upon receiving a response, the *TransactionEngine* analyzes the root element in the result, executing application-wide behaviors if necessary. For example, the client can exit if it discovers the session key has expired. The *TransactionEngine* then dispatches a *TransactionEvent*, which contains the root's children XML and the meta information as properties, with the callback function as the intended listener. It is the original requester's responsibility to notify the *TransactionEngine* when it has received the data, removing the callback function listener and allowing other requests to be processed.

6 Game Model

CONE Welder's game element is a powerful incentive for encouraging continued participation in activities useful to related research. Competition can help attract a larger audience beyond the birding community and other groups with an inherent interest in the project. At the same time, a cumulative point system forces constant participation in order to stay within the top rankings. The game's usefulness depends on satisfying the needs of two separate groups, players and researchers. Players want a scoring model with a clear correlation between actions and points. Awards should not be excessively difficult to achieve, but they also need to consider different skill levels. In addition, the game should be welcoming of new players, while recognizing the achievements of longtime veterans. On the other hand, researchers need a system that generates content specifically tailored for their needs. They would like to see awards target activities useful to their research and be based on how well desired conditions were met. The following sections explore the design of our scoring system and two implemented behavior-based awards.

6.1 Design

We divide achievable awards into two major categories: cumulative awards and daily awards. Cumulative awards encourage basic activities, such as taking snapshots or being the first to classify an image correctly. Their point values are static, since their achievement is not entirely indicative of skill level. This award helps new users achieve points early on. The associated prerequisites can be based on any collected data, even volatile ones, such as consensus classifications. Related actions in the cakePHP application include triggers to automatically update cumulative awards. As a result, these awards are immediately visible once earned and can be later retracted if the winning conditions are no longer true. Therefore, cumulative awards are a way to provide a dynamic scoreboard throughout the day.

Daily awards help recognize particular skills and encourage work desired by researchers. They are generated on a daily basis, based on activities during the previous day. Conditions can include any nonvolatile information within this scope. We enforce this limitation, since the system needs to be able to regenerate all awards based on past information. Changes after the target day could potentially alter the recipient and point values of daily awards if they were to include this type of data. Unlike cumulative awards, point values can be based on the particular conditions around the award. This flexibility allows us to award points based on how well the recipient met the researchers' goals. Methods to calculate the point value are highly dependent on the desired behavior. We provide two examples in the next sections.

6.2 Rating-based Awards

General interest of a photograph can be related to the presence of novel events that are valuable to researchers. We attempt to capture this interest through the star rating system. The interaction is very simple, requiring only a single mouse click. However, aggregate calculations over the data can help identify these interesting pictures. We reward the photographers of these pictures with the daily "Eagle Eye" award. To determine the winner, we consider all photographs taken on the target day. Valid candidates within this set must also have at least one rating from the target day. Each candidate photograph p_i is assigned a value $V(p_i)$ according to the following formula:

$$V(p_i) = \sum_{r \in R_i} \left(r - \frac{S_{min} + S_{max}}{2} \right), S_{min} \leq r \leq S_{max} \tag{2}$$

where R_i is the set of submitted ratings for p_i, S_{min} is the least possible stars given by a rating, and S_{max} is the most possible stars given by a rating. This formula is based on the idea that not every submitted rating is positive. It adjusts the rating system so that a mid-range rating is considered neutral. The photograph with the highest positive

calculated value receives the award. In the event of a tie, all winners receive the award, but points are equally distributed among them. Low-rated photographs could potentially win this award when the candidate set is small. We account for differing levels of competition by rewarding points as a function of the candidate size. Each award a has a point value $P(a)$, calculated as follows:

$$P(a) = B_A \left(\frac{C_a}{D_A} \right), a \in A \tag{3}$$

where A is the award type of a, B_A is the base point value for A, C_a is size of the candidate photograph set for a, and D_A is a scaling factor for A. A maximum and minimum value is also imposed to restrict the possible point range.

6.3 Time-based Awards

Ideally, CONE Welder will eventually provide good temporal coverage with a user group spanning multiple time zones. Until then, we need to encourage current users to use the camera at the beginning and end of the day. CONE Welder's "Early Bird" and "Night Owl" awards are incentives for this behavior. These awards are given to the photographers with the earliest and latest photos during a certain time interval. These conditions by themselves are easily exploitable. Through trial and error, a user could discover the cutoff times and take pictures around them. Also, the target time interval can be better estimated by actual user behavior. During public beta, a snapshot of a rare species outside normal daylight hours caused users to logon around this time on subsequent nights. We consider these conditions in our scoring model by comparing the winning photograph to the runner up. The awarded point value is determined by an exponential decay function displayed in Figure 11.

The function awards points based on how close the winning photograph is time-wise to the runner up photograph. We designed this function to have a maximum value of 10 points and a half-life at one hour. It helps prevent people from targeting set times by awarding a negligible amount of points for solitary camera operation.

7 Experiments

CONE Welder officially launched on May 12, 2008 after a two-week public beta. By the time of this writing, 21 days into operation, over 200 registered users had taken nearly 2,400 photographs. They had produced a sizable amount of associated data, including 1,500 comments, 3,000 zones, 1,800 ratings, and 10,700 submitted classifications. Players had earned over 7,600 awards and 9,800 total points. While we envision this project's operation

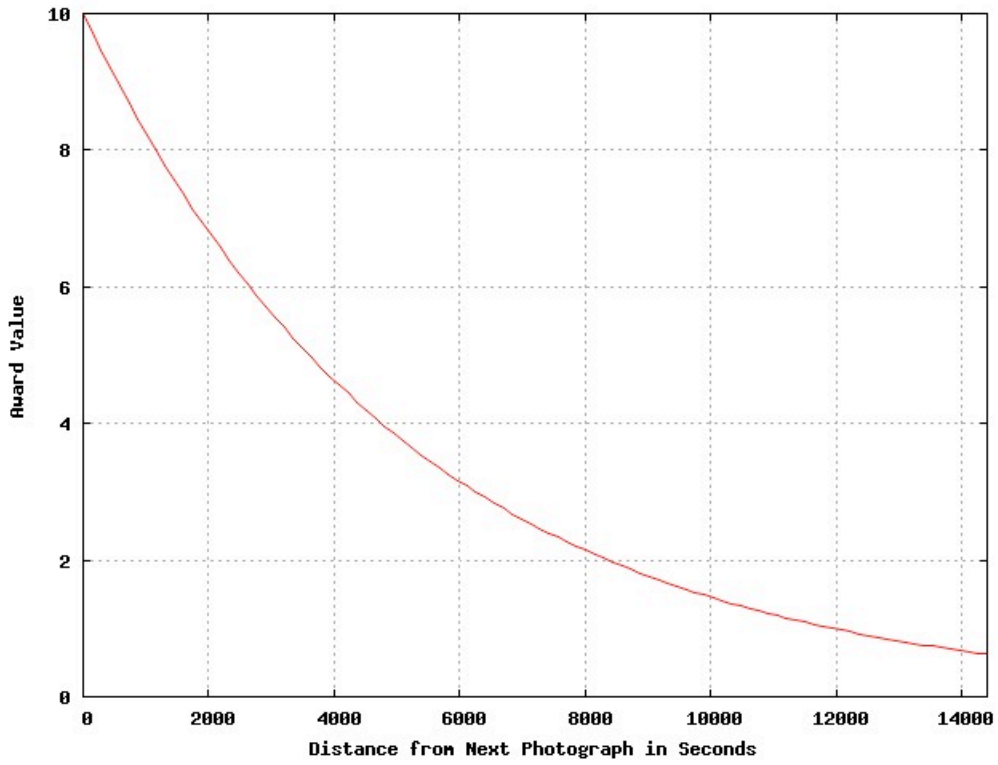


Figure 11: Graph of Exponential Decay Function for Determining Award Value

spanning multiple years, this initial data set can help gauge our system's effectiveness. The following sections discuss these results in relation to our original objectives.

7.1 Data Generation

A constant influx of photographs is key to CONE Welder, since this data provides a reference for researchers and award opportunities in the online game. An ideal photo collection would exhibit good temporal coverage, providing a chronology of daily activities. Continuous zone, comment, and classification submissions are also critical, since photographs can become quickly neglected once buried beneath newer ones. We analyze the time distribution of these activities in Figure 12, which aggregates each activity by hour:

Some interesting patterns emerge from the data. First, there are a substantial number of photographs taken in every hour between 7AM and 8PM CDT. This time interval coincides with the current sunrise and sunset times of the deployment site, which are 6:46AM and 8:06PM CDT respectively. While it would be great to have a consistent number of photographs throughout the entire day, these results are nonetheless encouraging for our temporal coverage objective. Despite a lower photo count, pictures taken during the night have provided a very detailed look into nocturnal activity. Plus, we believe our user base

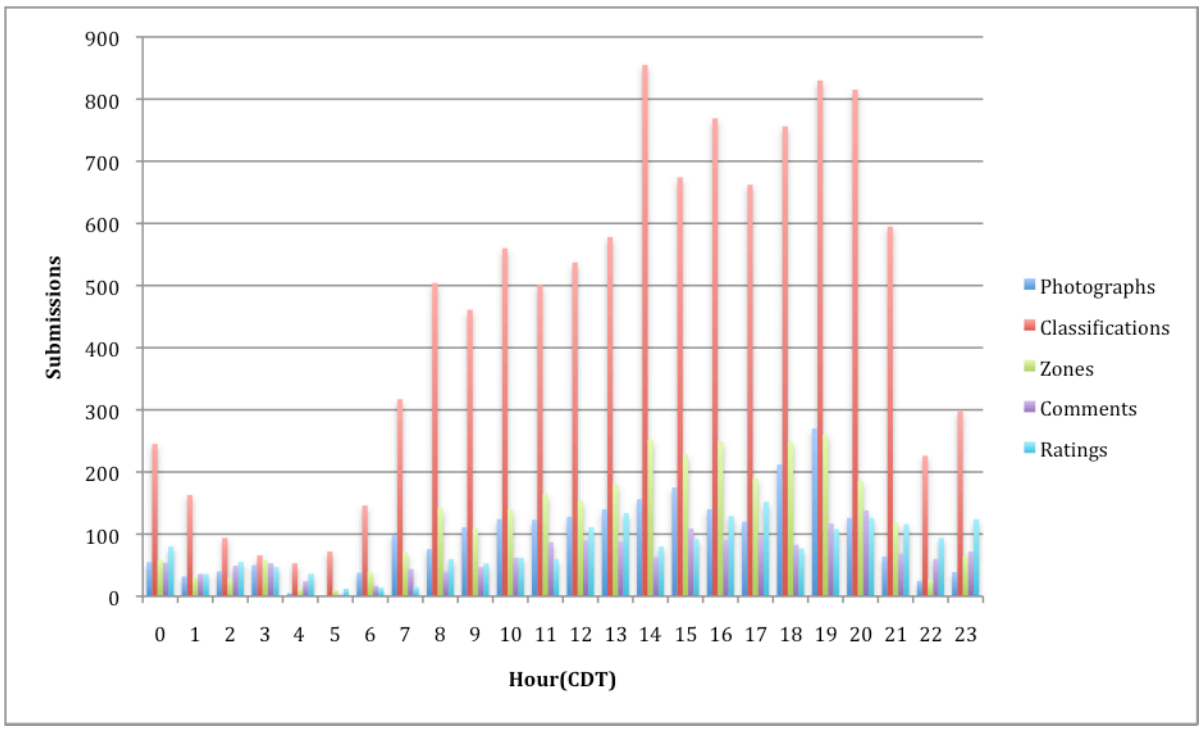


Figure 12: Time Distribution of User Activities: April 28, 2008 - May 17, 2008

will eventually span multiple time zones to further increase these other hours. Another interesting observation is that comments, ratings, and zones surpass photography at night. A future time-sensitive gallery could account for this shift by showing earlier photos first, helping promote these activities beyond the initial front-page attention.

7.2 Classifications

Classifications are also very important to researchers. This information helps easily identify new species and track previously seen ones. Our data shows that participants defined zones, each a classification opportunity, on 93 percent of all photographs. Among these zoned photographs, 73 percent had at least one zone with a consensus classification. Furthermore, consensus classifications were established with an average of 4.5 votes, .5 more than necessary if all were in agreement. These extra votes, which help validate the consensus classification, are most likely due to a time delay we place on publicly displaying this information. Users have identified a total of 28 unique species to date, shown in Figure 13. We are very pleased with these results, since CONE Sutro Forest only had a slightly higher species count of 30 at the end of its seven-month lifetime. The presence of particular sub-tropical birds within this early data set, such as the Green Jay and White-tipped Dove, is an encouraging sign of the project's potential usefulness to researchers.

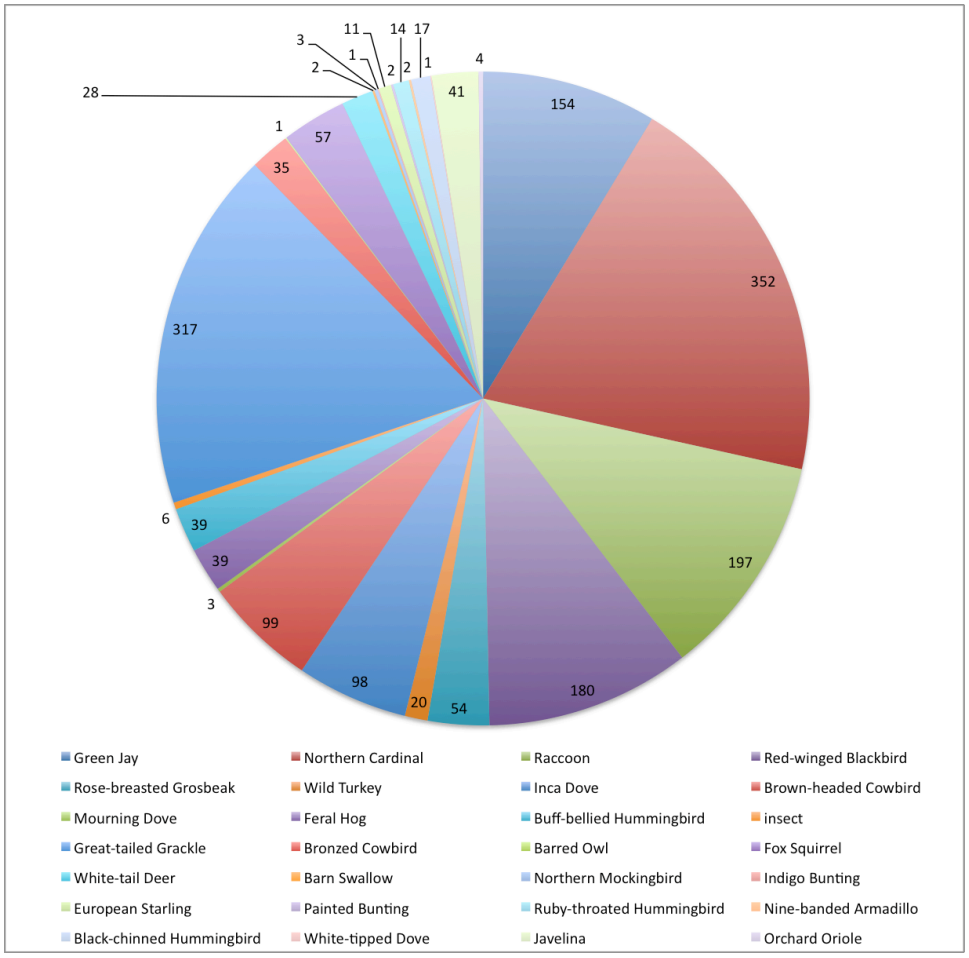


Figure 13: Consensus Species Classification Totals: April 28, 2008 - May 17, 2008

8 Conclusion and Future Work

Since its launch, CONE Welder has attracted a growing number of participants who have greatly extended natural observation at the deployment site. A light allowed participants to continue actively observing even at late hours during the night. The server and client applications have both scaled gracefully with an increasing user population and content. The new interface has been a success according to user feedback. Former CONE Sutro Forest members have been pleased with the layout and especially enjoy the new chat feature, allowing them to finally communicate with each other. The zone feature has kept the posted comments clean of classification directives, such as "classify the bird in upper-right corner". Users are fully utilizing the new interactive features, such as the rating system. This influx of new user-generated data allowed us to better refine the scoring models presented in this paper. The daily awards have helped us automatically identify stunning pictures to display (see Appendix C). Researchers have also found the

project to be helpful. Colored leg bands can be clearly distinguished in photographs among other features. The client interface has provided them a quick way to view observations of any species thanks to the drop down filter menu in the gallery.

Feedback from both users and researchers has highlighted opportunities for future development. Users have expressed interest in accessing their data outside the Flash client. Many of these users are also programmers and want to power blogs or other mashups with this data. CONE Welder's XML API can be extended to support a developer network. This extension may also be a perfect opportunity to explore developing CONE interfaces for other computing platforms. It is not hard to imagine someone enjoying the live video feed on a mobile phone during short periods of opportunistic downtime. Such development will demand even more efficient video processing.

Researchers have been quick to recognize potential extensions to the current system. Similar to species classification, color identification of bands on birds' legs would be very helpful. This addition would open up another potential metric for new awards. The filter options can also be improved to better assist researchers. A search feature based on attributes beyond classification, such as time frame or camera position, would allow quicker access to data of interest. However, additional metadata and a consistent indexing model will be needed for more powerful filtering. Similar to the user feedback, researchers want to access the data in formats outside the Flash gallery. A standard web-based solution specifically tailored for scientific study would be an excellent companion to the existing general interface.

The Welder Wildlife Foundation is currently developing educational curriculum around CONE Welder. This work includes study sheets and assignments gathering data. While the scope of this pilot program is geographically limited, future work can develop CONE into a useful teaching platform. A specialized interface would help better meet the accountability demands for class assignments. Multiple CONE systems could introduce students to a broad range of locations and many different bird species.

Acknowledgement

CONE Welder was a monumental undertaking, requiring the expertise and dedication of individuals across multiple disciplines. Professor Ken Goldberg provided superb direction and advice throughout the entire development process. Also deserving thanks are Connie Wang for her design work, Jeff Tang for his technical support, Yan Zhang for his database help, and Eric Liaw for his interface guides. The web camera's deployment and networking was a daunting feat by itself. Professor Dezhen Song and his students at Texas A&M met this challenge in the face of major obstacles and deserve much credit. We were fortunate enough to work with John Rappole from the Smithsonian National Zoological Park and Selma Glasscock from the Welder Wildlife Foundation. Each individual was a key player in

research considerations and preparing the deployment site. CONE Welder also built upon substantial work developed for CONE Sutro Forest. Ted Schmidt, Andrew Dahl, and Patti Bao contributed greatly to this earlier project. The NSF, Microsoft, and Panasonic's research support was also fundamental to the CONE project. Finally, we would like to thank Dr. Dan Garcia for his advice and feedback on this report.

A Interface Concepts

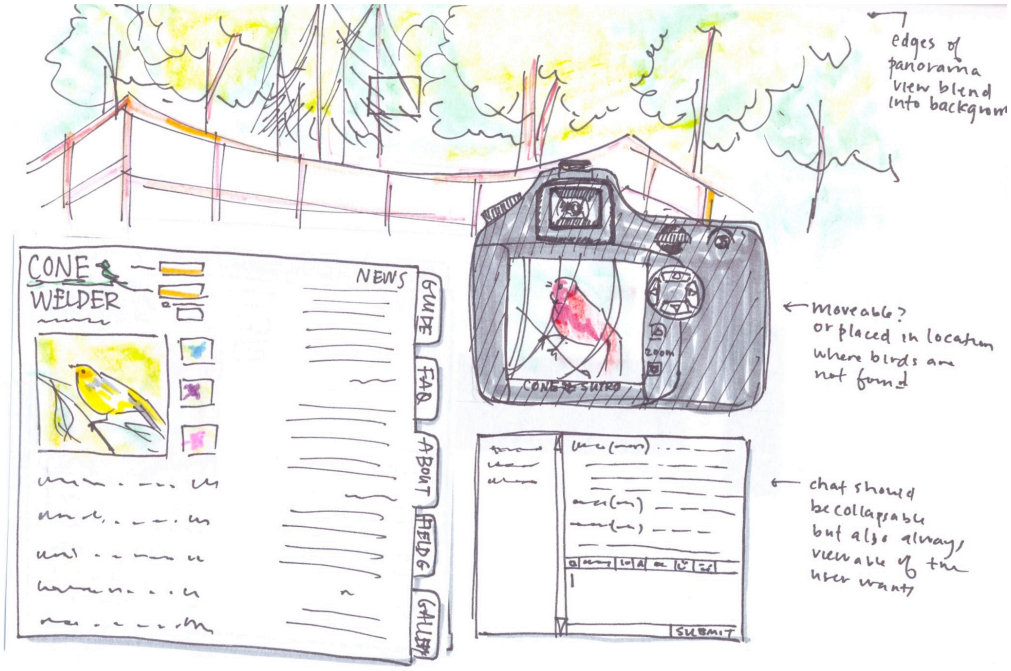


Figure 14: Early CONE Welder Client Interface Design by Connie Wang

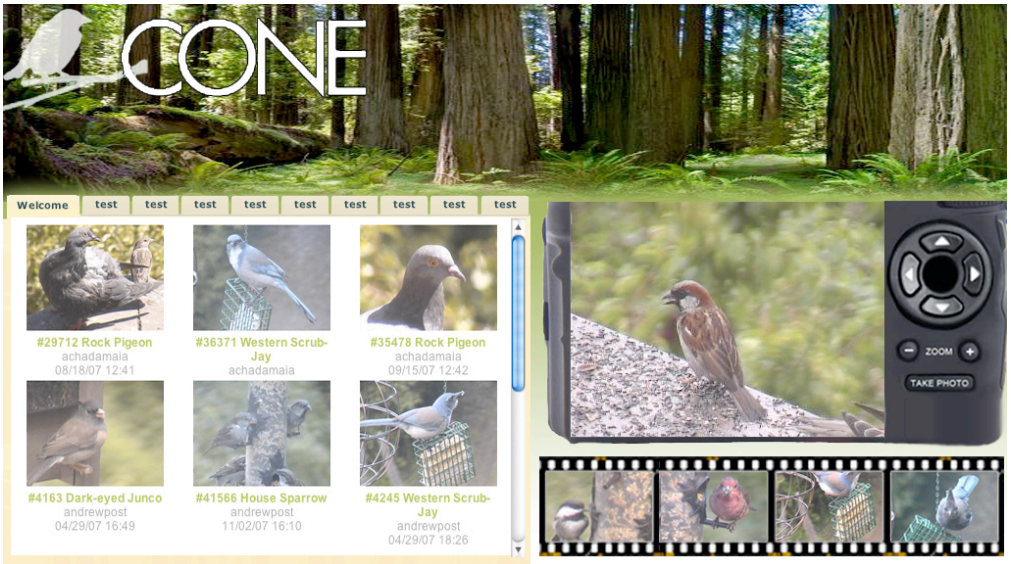


Figure 15: Revised CONE Welder Client Interface Design by Jeff Tang

B Communication Socket

The following tables detail the messages sent by the server and client. Some functionality was not used in the final CONE Welder release.

Table 1: Outgoing Server Messages

Name	Description	Syntax
`loggedin`	Notifies a client when they have successful logged in	`loggedin:`
`status`	Contains camera status information, including current camera position and outstanding requests	`status:camera;camX;camY;camZ:`
`newstatready`	Advertises the availability of new status information	`newstatready:`
`beat`	Requests response to keep socket connection alive	`beat:`
`system`	Contains a system status message	`system:text:`
`message`	Contains a message sent by another user	`message:username:text:`

Table 2: Outgoing Client Messages

Name	Description	Syntax
`login`	Requests the server to login the specified user	`login:username:userid:password`
`send`	Requests a message be sent to all users from this client	`send:text`
`statusreq`	Requests current camera and user status	`statusreq:`
`thump`	Responds to empty beat message	`thump:`

C Sample Snapshots

These photographs were taken during the CONE Welder public beta, between May 5, 2008 and May 12, 2008.

References

[1] Adobe - flex 3. http://www.adobe.com/products/flex/.

[2] Convert axis webcam stream to flash (.swf/.flv). http://www.daantje.nl/2007/05/07/convert-axis-webcam-stream-to-flash-swfflv/.

[3] Demonstrate. http://demonstrate.berkeley.edu/.

[4] Facebook. http://www.facebook.com.

[5] Youtube. http://www.youtube.com.

[6] *The changing climate of South Texas 1900-2100: Problems and prospects; impacts and implications*, chapter Apparent Rapid Range Change in South Texas Birds: Response to Climate Change? Texas A&M University, 2007.

[7] *Robotics Research*, chapter Networked Robotic Cameras for Collaborative Observation of Natural Environments. Springer, 2007.

[8] Yuh-Shyan Chen, T. C. Kao, and Jay-Ping Sheu. A mobile learning system for scaffolding bird watching learning. *J. Comp. Assisted Learning*, 19(3):347–359, 2003.

[9] Nak Young Chong, T. Kotoku, K. Ohba, K. Komoriya, N. Matsuhira, and K. Tanie. Remote coordinated controls in multiple telerobot cooperation. *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, 4:3138–3143 vol.4, 2000.

[10] A. Dahl. Implementation of a collaborative observatory for natural environments. Master's thesis, UC Berkeley, 2007.

[11] K. Goldberg, B. Chen, R. Solomon, S. Bui, B. Farzin, J. Heitler, D. Poon, and G. Smith. Collaborative teleoperation via the internet. *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, 2:2019–2024 vol.2, 2000.

[12] K. Goldberg, Dezhen Song, and A. Levandowski. Collaborative teleoperation using networked spatial dynamic voting. *Proceedings of the IEEE*, 91(3):430–439, Mar 2003.

[13] Jingtai Liu, Lei Sun, Tao Chen, Xingbo Huang, and Chunying Zhao. Competitive multi-robot teleoperation. *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 75–80, 18-22 April 2005.

[14] T. Schmidt. Design, algorithms, and architecture of an improved collaborative observatory for natural environments. Master's thesis, UC Berkeley, 2007.

[15] D. Song and K. Goldberg. Sharecam part 1: interface, system architecture, and implementation of a collaboratively controlled robotic webcam. *Intelligent Robots and*

Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on, 2:1080–1086 vol.2, 27-31 Oct. 2003.

[16] D. Song, A. Pashkevich, and K. Goldberg. Sharecam part ii: approximate and distributed algorithms for a collaboratively controlled robotic webcam. *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, 2:1087–1093 vol.2, 27-31 Oct. 2003.

[17] D. Song, N. Qin, and K. Goldberg. Systems, control models, and codec for collaborative observation of remote environments with an autonomous networked robotic camera. *Autonomous Robots*, 2008.

[18] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 319–326, New York, NY, USA, 2004. ACM.

[19] L. von Ahn, R. Liu, and M. Blum. Peekaboom: a game for locating objects in images. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 55–64, New York, NY, USA, 2006. ACM.

[20] N. Wilkinson. Cooperative control in tele-operated mining environments. In *Proceedings of the UBC Human Interface Technology Conference*, Vancouver, BC Canada, 2004. UBC.