Recombinant Design: Leveraging Process Capture for Collective Creativity

by

Annie Ding

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2006

Author .....
Department of Electrical Engineering and Computer Science
May 25, 2006

Certified by .....
John Maeda
Associate Professor of Design and Computation
E. Rudge and Nancy Allen Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Recombinant Design: Leveraging Process Capture for Collective Creativity

by

Annie Ding

Submitted to the Department of Electrical Engineering and Computer Science
on May 25, 2006, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Design is omnipresent and fundamental to the modern world, yet so little of the rich semantic information of the design evolution is preserved. If we are to gain the greatest knowledge and utility from a creative work, we must understand and preserve the process by which it was designed. To pursue this understanding, I have designed and implemented two electronic media-based process capture frameworks that automatically capture and share process as well as provide process reviewing tools. The first system, Chronicler, is a universal capture framework which captures fine-grained process information at an action resolution, demonstrated through the example of a painting program. The second system, Artwork Genealogy, a component of the OPENSTUDIO project, uses versions embedded with process metadata to document the evolution of artwork in an open collaborative community. This web-based system was launched to users in February 2006 and continues to collect art processes from an active and growing community.

Through simple, friendly user interfaces, these two systems encourage designers to donate to a repository of shared, searchable design information from which design rationale, the explanations for design decisions, can be inferred. The comparison of these two systems, through data mining and user analysis, shows the effectiveness of these methods for collaborative process capture and combinatorial process reuse. In particular, I demonstrate the ability of process capture systems to give rise to emergent behaviors, uncover process regularities, and to empower designers through five key areas: learning from past work, reusing ideas and work, expression, attribution, and evaluation.

Thesis Supervisor: John Maeda
Title: Associate Professor of Design and Computation
E. Rudge and Nancy Allen Professor of Media Arts and Sciences

Acknowledgments

To all those who have helped me get this far,
My good friends at MIT, for all the help and laughs over the past five years,
Fellow members of the PLW, past and present, for showing me the power of creativity with a lot of diligence,
My advisor, John Maeda, for sharing his vision and showing me that there are no limits to what a person can do,

Most of all, to Jared and my family, for everything,
I thank you.

Contents

List of Figures

Chapter 1

Introduction

Our everyday surroundings are enriched with the end products of conscious design. Countless hours and thoughts are embedded in designing the car you drive to work, the building you call home, the art hanging on your walls, the books you read, and even the bottle of water you drink. Behind each of these lies a wealth of creativity, an entire design process involving inspiration evolving into implementation as well as experiments resulting in successes and failures. However, this process information is all too often transient, taken for granted and lost in the rush to move on to the next production.

If we understand and preserve the process by which a creative work is designed, we are empowered to acquire knowledge and utility far beyond the surface value of the work. We can capture the wealth of semantics behind creation that is typically hidden from us. This thesis presents two software systems that encourage designers to capture and share design processes, creating a rich repository of reusable design data and rationale.

One of the most interesting examples of process capture occurred fifty years ago when Henri-Georges Clouzot made a revolutionary film, *The Mystery of Picasso* (Figure 1-1). This film was an attempt to preserve Pablo Picasso's art creation process for eternity [11]. Clouzot used time-lapse video to capture the surface on which Picasso created many paintings and allowed viewers to watch as Picasso transformed his painting of flowers into a fish and then a chicken, discovering what Clouzot de-

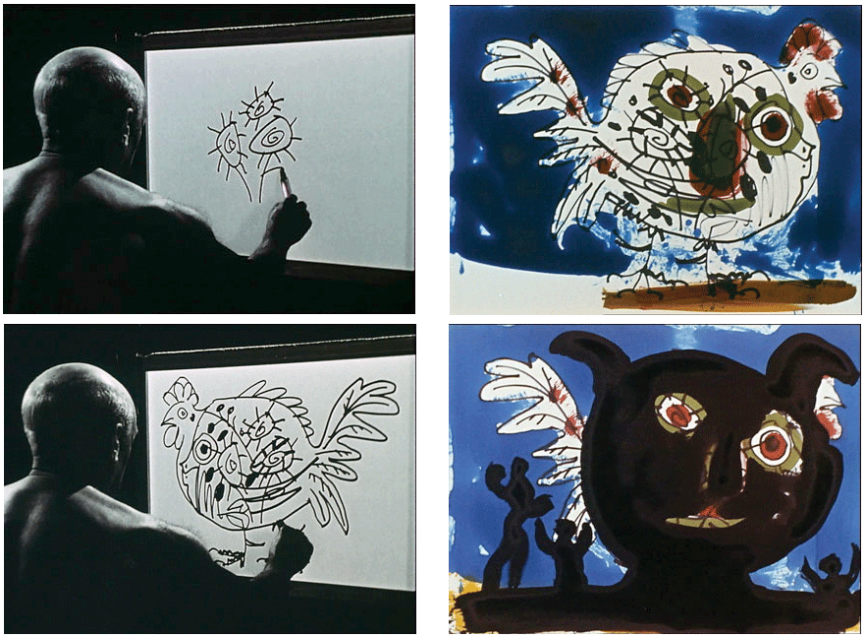

Figure 1-1: Clouzot's *The Mystery of Picasso* allows us to watch Picasso's art creation process.

scribed as "the secret mechanism that guides the creator." With his film, he was able to capture the intricacies of Picasso's design process from sketch to finished painting. As Clouzot wisely said to Picasso, "When we are all dead, you and me and everyone, the film will still continue to be projected."

Clouzot was able to foresee the immense value of process capture in bringing to light a new and complex dimension of Picasso's work that was invisible even in the myriad of final paintings. The field of process capture, however, was still in its infancy at that time and lay dormant until computers and digital storage dramatically reduced the cost and flexibility of capture solutions. Today, our world is replete with content creation and manipulation, and parts of the process capture methodology can be seen in applications across all domains through features such as histories and undo/redo.

In designing a system to advance the state of the art in process capture, there are two core questions that must be asked. First, what does a process capture system need to capture? Understanding and answering this question means understanding

the flow of thought underlying the design process. In many respects, the design process across all domains can be boiled down to an iterative search through infinite possibilites driven by some internal or external motivations and heuristics. This process can involve targeting a problem, researching the space, brainstorming solutions, collaborating with others, drafting proposals, implementing prototypes, identifying successes, learning from mistakes, and getting outside feedback, which can then lead to new creations. During this process, layers are slowly exposed as the space is explored, giving insights as to the next incremental step to be taken.

Based on this construction, there are two key elements that can be captured at each point in the process: the decision made and the reason underlying that decision. The decisions are what we want to explicitly capture as the design process - the time dimension of work as it progresses through this iteration, the history stringing together the ideas on the way to the final creation. The reasons are commonly called rationale capture and is significantly more difficult, often being recorded only with the explicit help of the user. Often, even the designer herself is not fully conscious of the reason for certain design decisions because she takes many of the steps for granted. Where the design process answers the question of how something was designed, design rationale answers the question of why. The focus in this thesis is on capturing the process in ways so that design rationale can be inferred.

The second core question to be asked is: What specific value does process and rationale capture generate? This question helps us determine which features of a process capture framework are most desirable, forming the basis for the engineering decisions made during design and implementation of capture frameworks.

1.1 The Applications of Process

Specifically, I identify the following five applications of process information, which are most useful in a collective environment where processes are shared. Process can

1. Enable learning.

2. Enable reuse.

3. Serve as a method of attribution.

4. Provide a better basis for evaluation.

5. Become a means of expression.

1.1.1 Learning

The most common use of process capture is as a learning tool. Specifically, this learning is enabled in two contexts: individual and shared. When an individual is able to preserve his own creation processes, he gains greater insight into his own work by reflecting on concrete steps. His former work serves as a launching point for new work, affording efficiency since work does not need to be repeated. This kind of reflection on process also helps individuals realize open areas for exploration and improvement.

The utility of design process information increases dramatically when it becomes shared, i.e. when it moves from the individual space into a collective environment. Humans learn by emulation and shared processes give us sources to emulate. A creation process is yet another precious learning resource. For formulaic actions, it makes the methods explicit; for areas that involve sparks of inspiration and creativity, it gives us intuition into what usually seems cryptic. Mathematical integration becomes systematic when we are given the steps and rules. Monet's lilies would become less distant and mysterious if we could watch him paint them. When process is shared amongst and collected from many, people can learn from each other, build off of other's successes, and avoid other's errors.

1.1.2 Reuse

The ideas from work can be reused, and consistently are in all domains. Reuse can be as simple as copying as in the case of designer knockoffs, but reuse can also be creative. Furthermore, even parts of the work itself may be reused. For example,

Picasso made a creative extension of his existing work when he changed a flower painting into a chicken painting. Reuse buys efficiency and sometimes reflects sources of inspiration.

1.1.3 Attribution

In a collaborative world, process becomes a means for attribution. As we described above, ideas are constantly recycled; this can range from one work inspiring an idea that leads to a new work to an exact copy of a previous work. Thus, process captures gives us the capability to make an author's role more explicit. We can tell exactly which steps of the process were created by whom and which actions generate trails of authorship that cumulatively form a rich genealogical network.

1.1.4 Evaluation

Examining a creation process can also help us better understand the value of work. Was something created in a matter of seconds or was it painstakingly made over the course of a year? Is some design a blatant copy or purely original? Is this the seminal work that inspired many successors? These are all questions we often ask when we assign value to the objects around us. An understanding of the process helps answer these questions.

1.1.5 Expression

Lastly, and surprisingly, a shared process can become a means of expression over time. Like movies and animations do for images, process adds the time element to the creation of a work. When the process shared, the sequence of events tell a story about how a design came into existence. As we see someone cook a meal, or make a painting, we not only are get a tutorial, but we also see how they feel.

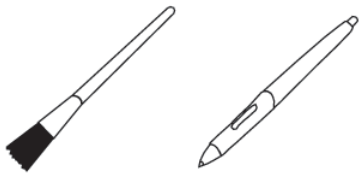

Figure 1-2: Classic art tools, such as the paint brush do not allow us to capture history, but new digital tools do [32].

1.2 Process and Rationale Capture Challenges

Today, despite all the advantages of process capture, we still let many design processes disappear. One reason for this is that there is no tangible representation for many of the steps of a design process, especially the early brainstorming steps. Thus, capturing the process involves placing the burden of explicitly conveying thoughts onto the creator. However, there are still many steps of the design process that have concrete representations, such as sketches and prototypes.

The multitudes of creation software for everything ranging from text documents to mechanical models show great promise for an increase in process capture. When design content is created digitally gestures and states can be captured and saved by the computer (Figure 1-2). Yet still few design programs capture the process and hardly any share the design information. The reason for this deficiency is lies in three key challenges that make process capture and utilization difficult.

1.2.1 The Representation Challenge

First, there exists the problem of finding a good process representation. Given the benefit from design reuse, it is important to have the ability to use pieces of the process as building blocks to effectively recombine processes. For example, a process might be captured by a person making notes and text documentation, or by filming the process. However, these methods do not give the user any tangible reference for the design. What is of much greater use is if the process can be captured in a way so that at each step of the process, a user can actually see, touch, and manipulate the

design.

Representation granularity also presents a challenge. The sampling of the process cannot be too fine grained as otherwise storing, searching and manipulating this data will be intractable. On the other hand, if the process is captured at too coarse a grain, some of the most important steps might be missed entirely. The granularity of representation also affects the storage space required to build this repository of information and the performance of such a system that needs to frequently record. Similarly, the correct intermediate detail representation must be found so that searches can perform useful comparisons at efficient speeds. Finally, the representation must take into account the difficulties of a collaborative space. Each part of a process must be associated with the correct author.

1.2.2 The Usability Challenge

An even more difficult problem is making process capture systems usable. While software can record actions, rationale is nearly impossible to infer without external, explicitly-specified author input. Most people do not have the patience to document their design processes. If the capture method is not streamlined and unobtrusive, it can seem burdensome and detrimental to the flow of ideas, inhibiting the process itself. Thus, such a system must balance providing methods for invisibly capturing information, such as recording capabilities, with methods to capture additional information which might not be easily inferable, like through annotation features. It must provide benefits that far outweigh the cost of providing information to the system.

Furthermore, it must make the information captured easy to reuse. The process information must be displayed in an intuitive manner, such as through simple information visualizations. These visualizations must capture the development of the process and allow the user to interact with process states.

1.2.3 The Sharing and Collaboration Challenge

Lastly, the information must be shared so that people can benefit from the design experience of others. Sharing is most useful when people can also collaborate. Collaboration makes the implementation of process capture systems much more complex because many people could be modifying and creating processes that could overlap and must be captured at the same time.

However, the greatest challenge of all is finding the optimal balance of these attributes for given purposes. A very detailed representation may come at the expense of a fluid UI. A solution that may work for one person, could become impossible to use in a collaborative arena. The wide spectrum of benefits and challenges has resulted in the creation of many software products that we will describe in detail in the related work.

1.3 Engineering Contributions: Designing Systems for Process Capture

To better pursue the goal of capturing and understanding process, I have created two systems to explore solutions to these challenges. This thesis approaches the problems by focusing on capturing the design process in an easily reusable way in a collective environment. It approaches the problem of capturing rationale by presenting the process to users in intuitive ways that allow the user himself to easily infer rationale.

The first system, Chronicler, is a universal capture framework which allows design collaboration. An example application developed in this framework captures the digital art drawing process in fine-grained steps. This detailed approach captured every brush stroke and gesture of the artist and allowed for artists to annotate key steps along the way. The system also provides a user interface for reviewing and searching through the shared design processes and design metadata.

The second system, the Art Genealogy component of the OPENSTUDIO project leverages the wide OPENSTUDIO online user base to capture many more design

processes. It uses a coarser representation, similar to a version control system for artwork creation. The versioning of evolving artwork creates a network of authorship that results in a rich "genealogy" for each piece. In the OPENSTUDIO environment where artwork is bought, sold, and re-edited, the Art Genealogy gave artists proper attribution for their work and provided another way for making expressive connections.

These two systems show the tradeoffs that result from different design decisions made in pursuit of the same overall goal. They present frameworks for the integration of process capture in design program and provide representations that allow for works to be extended and combined in a collective environment. Finally, they emphasize simple user interfaces to encourage as many designers as possible to share their design processes in a central location, generating a wealth of user data.

1.4 Additional Contributions: Collecting and Analyzing User Processes

The wide use of the Art Genealogy system within the social setting of OPENSTUDIO has also led to the collection of a large repository of user design processes. An initial analysis of this data shows the promise of extracting regularities from design processes to recognize style and authorship. Furthermore, the Art Genealogy system has resulted in many emergent behaviors supporting the five major design process and rationale applications outlined in this section. Users autonomously create, sell, reuse, and extend artwork.

1.5 Why Use Art?

Although the value of a design process applies to all kinds of fields, art as a domain of study offers unique opportunities. Art is a field that is considered inherently creative and has no rules. Thus, because so much of what is creative can't be reasoned or explained in words, it also has an aura of mystery about it, enticing our curiosity.

We often look at a piece of artwork in awe, wondering how an artist could have done it, wishing we could do it ourselves. What if we could have seen how Michelangelo patiently painted the ceiling of the Sistine Chapel over the course of four years, or watched as Van Gogh passionately laid stroke after stroke of thick paint onto his canvases? Many historians and documentary makers spend their careers trying to unravel these mysteries. Art is a field where you cannot just create work by recipes, and thus process capture really serves to reveal some of the intuition that can be revealed merely by watching an artist.

Additionally, the beauty of visual art is that there is a concrete visual reflection of thoughts and ideas throughout the design process. Most artists start with sketches, and these line by line, stroke by stroke, eventually grow into final works. The world of digital art gives us the opportunity to capture every gesture as a work is created. Capture mechanisms can be embedded into the digital tools themselves, taking advantage of the natural software extension of Clouzot's work.

Because art has no real rules, it becomes possible to morph and rearrange artwork over the course of time, making the time dimension plastic. This is particularly of interest in process capture because it means that the steps of the process can be rearranged and reused.

Lastly, art is a domain where originality and authorship are particularly valued, yet often works are inspired from others. A process capture mechanism can support new models of art development where art can be derivative or collaborative.

1.6 Overview

Chapter 2 gives an overview of past design process capturing and rationale systems and their strengths and weaknesses. It also provides an overview of the OPENSTUDIO project.

Chapter 3 gives an in-depth account of the strategies and challenges behind the design and implementation of the Chronicler and Art Genealogy systems. It then details the user interface design for both systems.

Chapter 4 compares and contrasts the designs of both systems. It also addresses the user tests and data collected as well as some of the emergent behaviors from Art Genealogy. Lastly, it provides the results from some initial data analysis in extracting regularities from processes.

Chapter 5 presents some future work that can be done in light of these systems and the generated data.

Finally, chapter 6 concludes with a summary list of the contributions made through this work.

Chapter 2

Related Work

In building a system that allows us to understand the design process, we must draw from many fields of related work, representing a confluence of recording and rationale capturing technology, artificial intelligence, psychology, and collaborative software environments. This section provides an overview of the broad spectrum of design process capture systems as well as an overview of the OPENSTUDIO framework, which both of my capture frameworks supplement.

2.1 Process and Rationale Capture Systems

At present, a vast range of process capture frameworks has been developed mostly in the areas of software development and digital art and design media. Each of these frameworks highlights different points on the spectrum of representations and usability and illustrate the tradeoffs made by the design decisions in each. Here I survey some of the major research projects and commercial tools used for process capture and show where they lie on the axes of representation in terms of granularity and flexibility, usability, and collaboration.

The frameworks reviewed here as previous work are as diverse in capability and focus as the capture domains they try to chronicle. In terms of representation, some systems capture information on a time basis while others capture from an event based approach. Some merely record, while others try to capture the process in a reusable

way. Some use a continous, fine grained representation, while others try to discretize the process. Also, in terms of usability, though some systems require users to enter explicit information, other capture systems are designed to remain invisible to the user. In addition, while although many works focus solely on process capture, other research has focused on the artificial intelligence approach of letting both people and machines understand design rationale. The following sections explore each general type of capture framework in light of these dimensions, with emphasis on digital media process capturing for many of the same reasons that I have chosen to use the digital art domain.

2.1.1 Video Process Capture

On one end of the spectrum is the continuous capture of the design process using video recordings. This was pioneered in the art domain by Henri Clouzot's film, *The Mystery of Picasso* [11], in which Clouzot filmed the back of a translucent canvas on which Picasso was painting. The mystery behind Picasso's technique, which has been admired by the world for decades, was unveiled for us to watch. The film effectively captured the time dimension we never get to see in viewing Picasso's finished paintings, giving us an sense of what made Picasso's creation process unique, an everyday person could watch as he transformed a painting of flowers into a chicken (Figure 1-1).

Today, software tools have allowed us to apply this same approach to the work that people do on a computer. Programs such as Wink [23] and Snapz Pro [18] allow users to record activity on their computer screens, creating movies that allow viewers to have a taste of the creator's perspective (Figure 2-1). These applications then allow the creator to annotate or voiceover the video to create tutorials. Raison d'Etre [10] took this idea to the next level, capturing the perspectives from different members of a design team in videos stored on a centralized database.

This video capture technique has been shown to be immensely successful in educational uses. For example, popular tutorials made with similar programs, such as the popular Ruby on Rails introduction tutorial [1], have been viewed by multitudes on the web. The approach is incredibly simple, allowing people to just watch the

Figure 2-1: Snapz Pro [18] allows users to make movies of their actions on the computer and has been used to make many digital tutorials.

process as if they were standing next to the creator. The movie interface puts little burden on the user and watching a process is a very intuitive way to learn. It also applies generally to any type of work that can be seen by a recording device and has very little interference with the creation process. Thus, the process reviewing modules of both systems presented in this thesis also allow the user to view the process in animation form.

However, one major weakness of this approach is that in a film, the process capture representation is not at all tied to the medium of creation. The representation only allows you to see what is happening but in itself does not give any extra insight to how it happens. Thus, parts of the process cannot be directly reused. It is a very passive approach from the perspective of the viewers. In addition, the videos cannot be easily searched without the high costs and overhead of manual annotation, or the errors inherent in automated analysis algorithms.

2.1.2 Discretized Process Capture

Alternatively, the process can be captured by developing a representation that breaks it down into meaningful discrete parts and capturing those parts in sequence. The STARS system [26] worked in this way by developing a language for capturing the processes behind Boeing engineering. It divided processes into parts called activities. These activities included tools, input and output, the state of the activity, code for that activity, and a description of the activity. The Process Engine then synchronized the use of these activities in a multi-user environment, managing the states of all the activities.

Meyers and Zumel used process capture within the Rationale Construction Framework (RCF) show in Figure 2-2 in order to later extract rationale from the process [28]. The process capture was built within a CAD tool in which designers could create models such as gears. The process itself was captured as a stream of tool events. Important events included creation, deletion, modification, and annotation of design objects. Events such as undo and redo were also collected. The rationale extraction performed by this framework is shown in the Implicit Rationale Solicitation section.

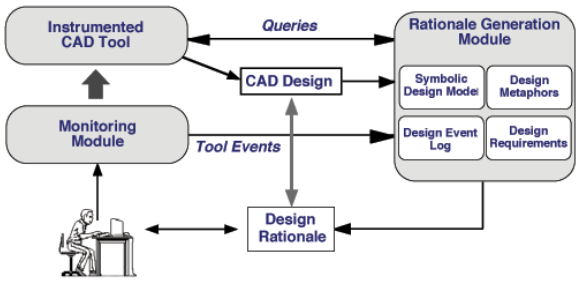

Figure 2-2: The RCF [28] architecture.

Treehouse Historian [32], a system for capturing the process behind digital raster drawings, also broke the design process down into very small steps. The Historian captured raw user input in the form of mouse and key events streams. As a user was drawing, the user could mark important key frames in the process (Figure 2-3). Because the representation capture was tied closely to the creation application, the representation allowed the process and key frames to be regenerated from scratch. Thus, a person could reuse an existing process and add work to a previously captured key frame or replay a process event by event.

The discretized process capture is advantageous in many ways because, as in the case with RCF, the representation lends it self well to comparison and extracting similarities and differences. Furthermore, as in the case for the stack models that supports undo and redo, it allows a person to navigate through the process of a design, with an editable state at each point. Similarly, the action stack representations allow design reuse. However, in cases where the process capture is tied very closely to the capture application, the process capture framework cannot be generalized to different applications and domains. Lastly, in using a discrete process representation, the granularity of each step is of great importance. For example, in the case of Treehouse Historian the mouse and key event granularity was so small that it resulted in performance lags. These systems influenced the action-based event capture used in both Chronicler and Art Genealogy.

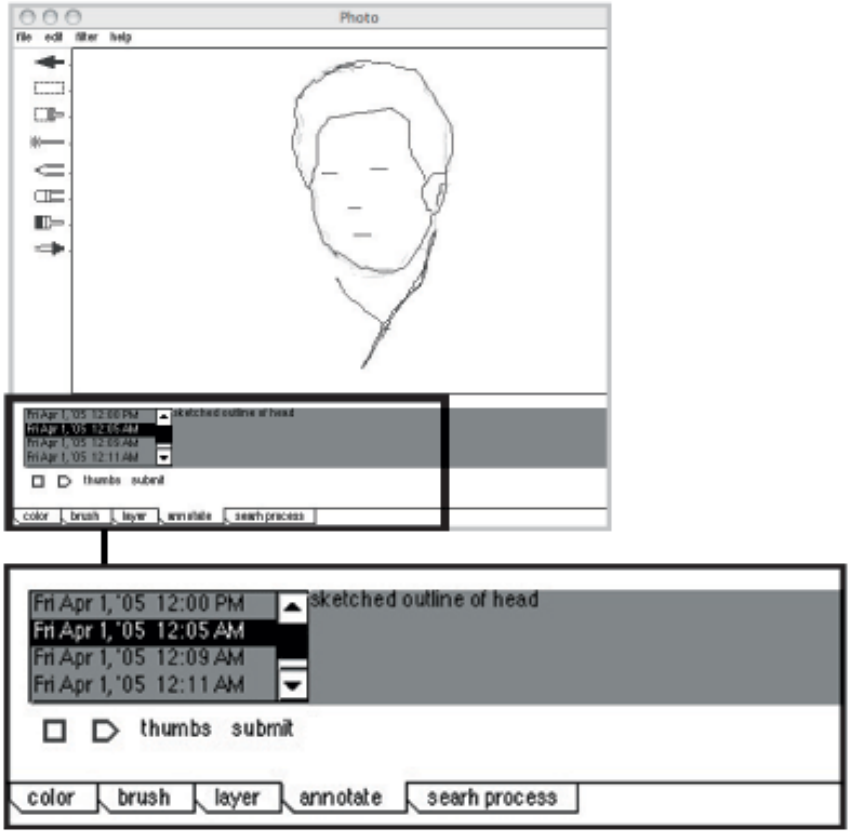

Figure 2-3: The Treehouse Historian [32] allows users to annotate steps of drawings after they are captured.

2.1.3 Rationale Capture

Of great relevance to this thesis is the approach taken by many researchers in the artificial intelligence domain for capturing design rationale. The intent of these systems is to capture the reasoning used behind design actions. These systems range from explicit documentation systems to systems that automatically extract rationale from the captured process. Because rationale capture often requires including the process, many of these system address process capture as addressed in the previous section.

One approach to rationale capture is direct solicitation, where the user is explicitly asked to input explanations about their design choices. A seminal work in this area was Gruber and Russel's [17] treatment of a new framework for capturing design rationale using a semiformal representation tool and model-based explanations in the creation of instructional materials. In their analysis of capturing design knowledge, they clearly defined design rationale as being the explanations of the structure, behavior, and function of artifacts. They also stated a major use of design rationale: sharing and preserving rationale for future use. Their semiformal representation system, IDE, captured knowledge via the creation application itself, having the stop and write explanations about the design process has he or she worked. Their model-based system, DME, had user input explanations that could be used as answers to questions about a model.

Another text explanation based capture framework, RADIO, an electronic notebook project, allowed software designers to contribute and access design knowledge represented in the form of technology books [4]. A successful commercial tool that also relies on direct solicitation of rationale is Maya Mentor [6], Maya's 3D modeling tutorial service. Maya Mentor consists of explicitly created tutorials that give step-by-step guidance from within their applications. It also gives access to a database of text tutorials, videos, and user comments.

These systems work very well from the perspective of preserving information so that future designers can benefit from past experience. However, they put the entire onus of capturing information onto the creator. Furthermore, often these systems fail

to tie the rationale closely with the creation itself, leaving it up to the user to make the connections.

Although Meyers and Zumel's Rationale Construction Framework [28] captured the design process, the main goal of the framework was to extract design rationale without burdening the user. In order to understand the design, the framework created implicit groups of the tool events called design metaphors. Recognizing patterns of tool usage and relating the patterns with design intent created these metaphors. The intent hypotheses are generated through a repository of background information on the design domain. As a result, the level of explicit human designer input is limited to adding annotations to the design process when desired. Process Makes Perfect also captured rationale implicitly by allowing users to mark key frames just by hitting a key while drawing [32]. This approach was then combined with direct solicitation by allowing users to input annotations on those key frames. Chronicler uses a similar system inspired by RCF and Treehouse Historian, aiming to track the meaningful moments inline with the process through minimal user input.

2.1.4 UI

In terms of user interface, the conclusion that can be drawn from these systems is that the UI must be as unobtrusive as possible. As noted by Meyers and Zumel [28], this is important because those that are obtrusive, putting the onus of gathering all rationale information from the user, defeat the purpose entirely. That is, a user will refuse to use a process capture system if it interferes with the design itself. Thus, the benefit of a capture system must outweigh any additional effort that it puts upon a user. This idea is applied well in the case of version control systems, which are described in the following section, as evidenced by their widespread usage.

2.1.5 Undo/redo

In addition, undo and redo models that are commonplace in most software applications essentially capture the process by noting the history of an application's use.

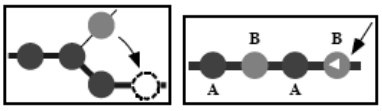

Figure 2-4: Gina [8] supports selective redo and undo of actions using a branching model.

For example, Adobe Photoshop's [3] history feature breaks the process down into discrete steps of tool actions. These actions create a stack of undo/redo steps and allow the user to see the past few events that have been recorded. Unfortunately, this information is not saved across sessions in Photoshop, and thus has only transient use. Furthermore, all final "production" formats for images such as JPEG and PNG discard this information.

More sophisticated systems such as GINA [8]and Amulet [27] support branching timeline models of undo and redo (Figure 2-4), capturing steps that linear models discard. Edwards et al [15] present the idea of multi-level timelines in undo/redo models, creating a hierarchy of changes, where users can interact with subsections of a design history as well as the history on whole.

2.2 Version control systems

Version control systems for software development are quite similar to process capture systems in the detailed information they store, and greatly empower users because of the collaborative capabilities. One can view version control as a process capture system that uses a very coarse grained representation, with each version as a step.

The most commonly used version control software is CVS, the Concurrent Versioning System [9]. CVS allows multiple software developers to manage and edit multiple directories of source code while keeping track of all the revisions done. It uses a client-server architecture in which all the revisions are stored on a repository on the server-side and the versions are pulled from, edited and then stored back to the repository from the client end. CVS stores the most recent version of a document and then uses backwards differences to track the past versions of text documents.

Via a command-line interface, users can also leave comments, revert to old versions, or make 'diff' comparisons to see the difference between two versions. CVS also supports branching in the development process and keeps track of authorship as a project develops. Subversion shares the same goals and uses as CVS, but includes improvements for revision tracking and access [30].

Adobe Bridge and Version Cue [2] are commercial tools that bring version control to design documents created using Adobes design suite. While the Bridge provides centralized storage, the Version Cue allows the user to manage versions, revert to old versions, create branches, and view the workflow. It also allows both the creator, and other collaborators to leave comments on a version.

However, the main difference that sets version control tools apart from process and rationale capturing systems is in their intent. The goal of most version control systems is to allow multiple people to work concurrently on a project without stepping on each other's toes, not preserving the process of the making the project. They allow users to revert to previous versions, but do not emphasize reviewing the process on whole. Usually, in software development, the work in version control systems is saved and only revisited in the worst-case scenario, when something fails to compile or a bug arises. They also fail to support the ability to search across versions. The result is that version control systems are very end driven and although they preserve the process, few make the process easy to view.

To address this discrepancy, I worked with two other team members to create Save Your Design [14], a graphical user interface extension to Subversion. This system allowed all the basic functionalities of the underlying Subversion system, but specifically with visual data in mind. Through a simple but intuitive interface, it allowed users to actually get a rough view of the process by visualizing previous versions in a timeline as shown in Figure 2-5. A user could then drag and drop versions to and from the repository to make revisions. It also allowed users to do direct comparisons of different versions of visual images, an option not available in Subversion and CVS because they only allow textual comparisons.

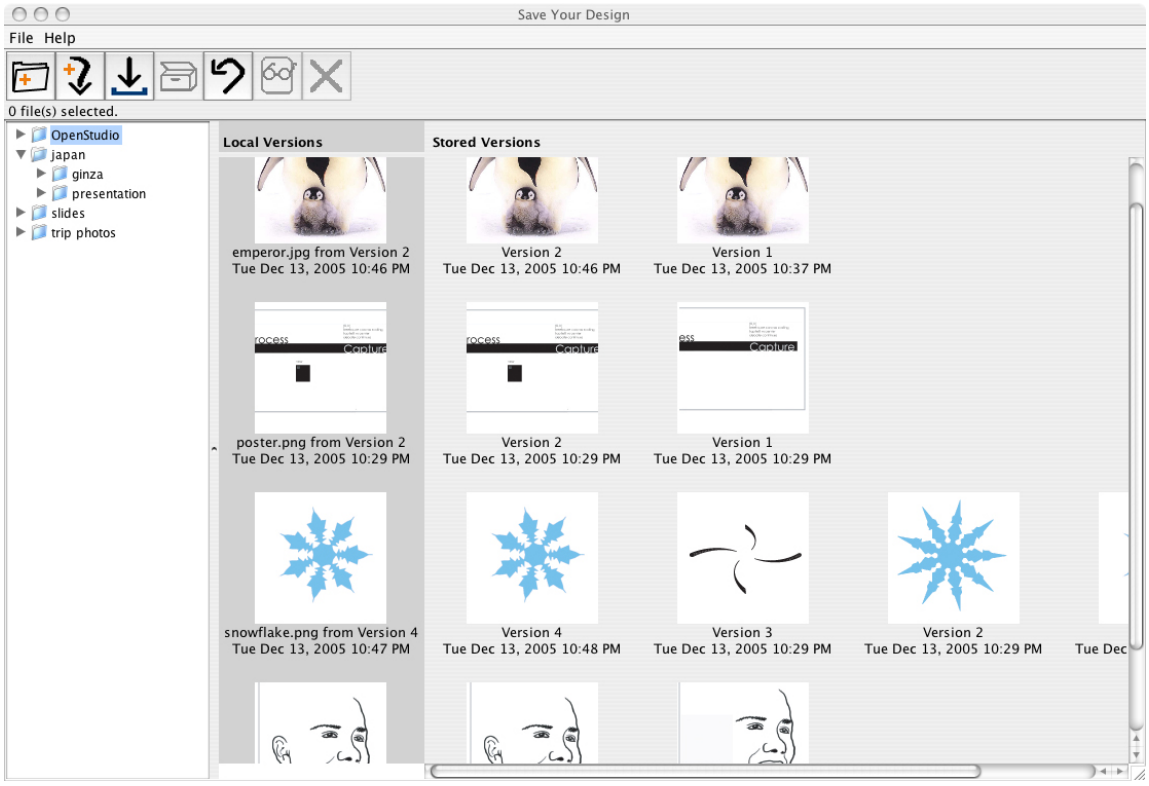

Figure 2-5: Save Your Design [14] is a graphical user interface extension to Subversion [30], visualizing the version changes in design documents.

2.3 Knowledge Representaion

Throughout all the different systems presented, process and knowledge representation play a major role. The general consensus for any knowledge capture is that intermediate representations work best. This has been shown in Ullman's work on perceptual information [36], where he found that, in analyzing facial images, features of an intermediate size gave the greatest benefit in comparison. The reasoning behind this is that too fine a granularity of representation captures so much information that too many false negatives appear when trying to extract regularities. Likewise, too thick a granularity results in too many false positives. Shum has shown that the importance of intermediate representations also applies to the domain of design rationale capture [33].

2.4 Collaborative Information Sharing Systems

In contrast to the version control systems, collaborative information collecting systems are not as concerned about tracking the versions of a specific work, but fulfill the vision of preserving collective knowledge for posterity in a broader sense. These systems generally collect narratives of the process and possibly final documents, not intermediate elements of the process itself. The rising popularity of user content driven websites has pushed forward many sites and system that rely on collective contribution.

The most successful of these has been the Wiki [25]. Wikis permit users to accumulate knowledge in a webpage by allowing the viewers themselves to be editors. Though the Wiki user interface is somewhat limited, the simplicity has lead to phenomenal amounts of information contribution as in the case with Wikipedia [37].

Nkakaoji et al designed a framework to support collective creativity in using visual images, allowing users in a community to reuse other images created in the community [29]. They designed two systems, one that allowed users to search for images based on user contributed knowledge-based rules, and another which searched on the

basis of user generated word associations. Today, this idea is demonstrated through websites like Flikr [21] in which users share their images and donate to a growing repository of searchable and taggable data.

In the field of AI, the Open Mind Common Sense project [34] is collecting various pieces of common sense knowledge by turning to the general public for contributions to the database in an effort to cover this enormous domain of knowledge. This knowledge is then being used by multiple projects as a knowledge base for intelligent systems.

Furthermore, many websites that enable collaborative work and information collection have flourished. For example, the recent fad of DIY or Do It Yourself projects has resulted in many websites in which people share the processes tutorials for creating something. Instructables [24] is one example of these websites, in which a community shares projects with instructions on how to create them along any data that might be helpful in the creation process, such as CAD files (Figure 2-6). These projects range in anything from making soy milk to building machine tools. ThinkCycle [35] is an academic, non-profit site that allows people to collaborate on design projects by sharing information and communicating through message boards.

2.5 Digital Rights

Lastly, the issue of licensing becomes of great importance in an environment which allows design reuse and process exposure. Difficult issues of ownership arise, many of which are unprecedented. For example, how do you properly attribute credit on a derivative work? Does the original creator get more credit for having done the first work? Should credit be attributed on the basis of effort? Furthermore, once the process is exposed, can someone actually own parts of the process? The Creative Commons licenses [13] begin to address the complex issue of ownership of digital information, and include licenses that allow reuse. Models for royalties also address this issue, but with the easy proliferation and manipulation of digital content, attribution models for this area still require development.

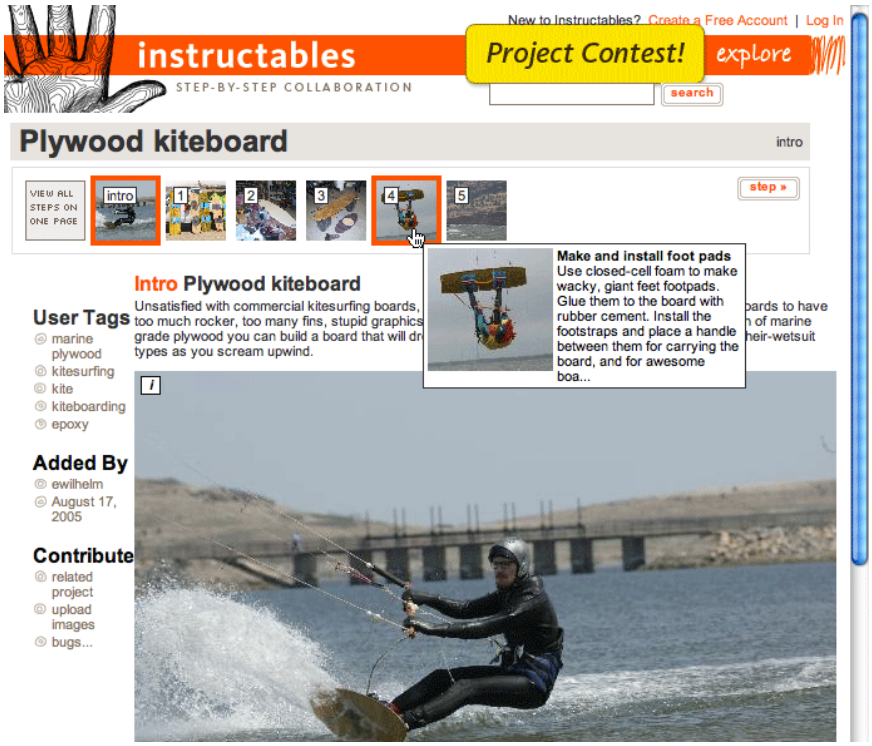

Figure 2-6: The Instructables [24] website lets a community of people share projects and tutorials on how to build and design a variety of objects.

2.6 Openstudio Project

Lastly, both the Chronicler and Art Geneology systems were built on the foundations and designed to complement the OPENSTUDIO project [5], a group effort by other members of the Physical Language Workshop at the MIT Media lab and myself. OPENSTUDIO, shown in Figure 2-7, is an online environment for the creation, sale, purchasing, and displaying of digital artwork in a growing community. It combines the online site with free creative tools built on a general application framework and connected via SMPL [31], a universal communication protocol, to a central document server. Currently, the only open tool is Draw, a simple, light-weight, vector-based Java webstart application.

Currently OPENSTUDIO has over 200 members with an expanding invitation system which results in new members joining every day. Although the work created and publicly displayed on the site can be seen by anyone browsing the web, only members are able to create and store their work on the system and make buyer/seller transactions. The result is a powerful community-based economic system.

Once an artist creates a piece, it can be saved either to a private inventory or to a public gallery where it can be placed for sale using a virtual monetary unit, the *burak*. Any piece on sale can be purchased by any other member in OPENSTUDIO and the transaction can be seen by all members as well. These individual transactions cumulatively reveal the values and roles of people in the community. Furthermore, community members can tag each other's work, creating an 'artsonomy', and adding rich semantic and communicative data.

Finally, the purchasing power in OPENSTUDIO also includes the ability to edit any piece bought and create derivative works. This thesis also explores the ability to track the authorship of these derivative works as well as the process within individual drawings created with Draw.

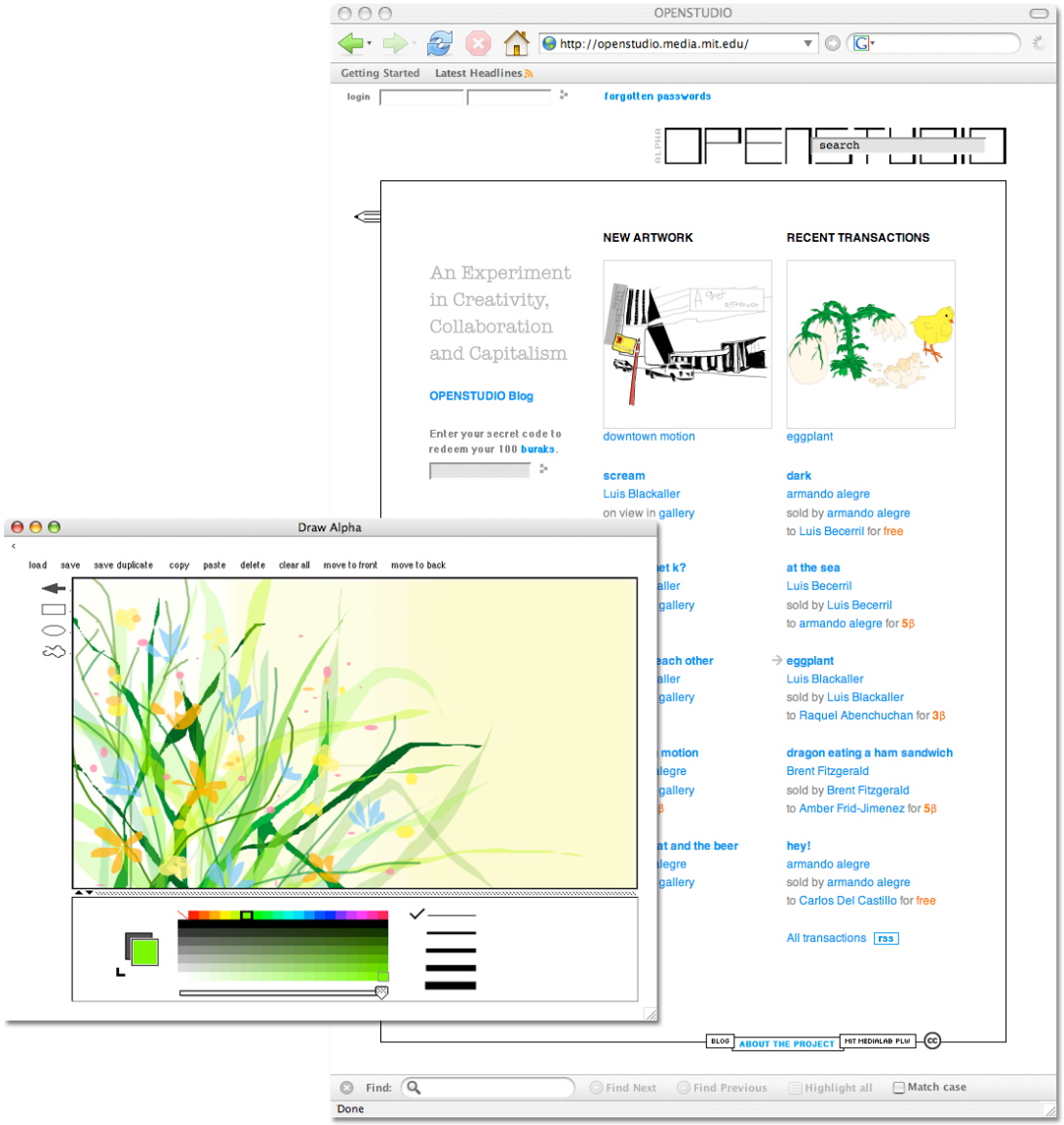

Figure 2-7: OPENSTUDIO, a web-based environment for creating, buying, selling, and collecting creative works in a community.

Chapter 3

Designing Process Capture Systems

In building a process capture system, my major design goals were shaped by the representation, usability and collaboration challenges I presented in Chapter 1. The course of researching the wide gamut of process capture systems resulted in the implementation of two different systems. This section describes the design decisions, strategies, and experiments in both.

My goal was to find a good intermediate representation for process that would allow easy manipulation and search as well as recreation of the process. I was aiming for a representation that would be flexible enough to apply to many different domains and could illustrate the often non-linear characteristics of process development. In addition, I wanted to design a simple, unobtrusive user interface so as to lessen the burden on the artist to contribute information and to ensure that the design process itself was not hindered by the capture system. Lastly, it was important that a process capture system could enable collaboration and open process sharing.

The first system, Chronicler, is a Java-based universal capture framework built from scratch and intended for supplementing the OPENSTUDIO design suite. Chronicler uses a fine-grained representation for process at the action level and automatically gives any application built within the framework the capability to transparently record actions as well as utilize a user interface for annotating, searching and review-

ing the process.

In order to leverage the creative processes of the artistic community of OPENSTUDIO and capture real user data, I turned to a different approach, adapting the existing design application framework for OPENSTUDIO to capture process. To do this, I created Art Genealogy, a web-based system launched to users in February 2006, which used a more coarse-grained approach for an intermediate representation. It documents the creation process first by acting as a simple shared version control system where the genealogy, or version evolution, of a piece of artwork is visible to the whole community. Additionally, each version is embedded with small amounts of metadata for the purposes of capturing some of the important details that proved useful in Chronicler. Each version can be tagged with comments both by the artist herself or any other member in the community. Using this system in an autonomous community of artists resulted in many interesting emergent behaviors and applications of process knowledge.

In moving from Chronicler to Art Genealogy, the focus shifted from building a process capture framework to support design application development to finding an elegant way to integrate process capture in an existing framework and suite of applications. A much more in-depth evaluation of the similarities and differences, and the resulting advantages and disadvantages of the two systems is in Chapter 4.

3.1 Chronicler

3.1.1 Goals

Chronicler is a universal design process capture framework, automatically enabling process capture and review for any design application built within it. The design of Chronicler can be roughly split into the design of three main parts: the process representation, the general framework architecture, and the front-end user interface. All three were driven by six main goals :

1. Domain independence

2. Combinatorial time manipulation

3. Process Search and comparison

4. Multiple user support

5. Rationale extraction

6. User friendliness

One goal of Chronicler is to isolate the capture framework from domain-specific needs. Although I use a digital paint tool as an example application, this process capture system extends readily and naturally to capturing the process behind other applications such as a text document or spreadsheet.

Secondly, the capture framework should effectively capture the time element of a work, supporting methods to review and manipulate that dimension such as through histories and unlimited undo and redo of actions. An extension of that idea is supporting the recombinant use of process information, so that bits and pieces of different works can be reused by recombining clips of the processes. Any combination of process steps should make a viable end product, allowing "process mixing."

Thirdly, the process information should be searchable. This involves making the retrieval of process information efficient. Additionally, it requires processes to be easily comparable in order to recognize similarities and differences.

Fourth, the system must support multiple users. As a multi-user system, it must allow collaboration on works, with methods to properly attribute authors for their work. It should also allow frequent storage and retrieval of processes in order to build a sizable shared process-knowledge repository.

Fifth of all, the capture framework must also allow the rationale behind a design to be inferred from the raw process.

Lastly, the system must be very user-friendly. Without good usability, none of the other goals can be achieved because it is only through user-friendly interfaces that artists will be willing to donate process knowledge to the system. Ideally, this will take no extra effort beyond the design itself on the artists' part. The artist must also

be able to see the benefits of process capture immediately to encourage her to input more design information.

3.1.2 Process Representation

At the core of this system is the representation used for process capture. In the past, capture systems have tried the approach of capturing all user interaction data, including every mouse gesture, in the hopes of maintaining all the information relative to the process [32]. However this came at the cost of efficiency, and in many ways information like individual mouse movements were too detailed a representation for effectively comparing different processes. Alternatively, the approach to saving the process that is commonly used today is that of saving versions [9] [30], however these are often too thick grained to give insight into how something was created. Research has shown the effectiveness of using intermediate representations for knowledge representation and more specifically for process information [36] [33] . Chronicler has a relatively fine-grained intermediate approach by using individual action changes as the base element of representation.

Action Sequences

We enable the above goals by representing processes as action trees where the nodes are individual states in the process and the edges are discrete, quantized, and deterministic actions specific to the desired domain. That is, rather than saving individual states along the process, the action changes that make those states are saved in a sequence as shown in Figure 3-1. In the context of design applications, actions can be thought of as the atomic changes that are made by using tools, menus, or meaningful key or mouse events. The end result is that the process itself represents a work or document. The process becomes transparent because each final document is merely the sequence of actions that created it.

Like branching timeline models [8] [27], this representation allows changes to be appended to any previous point of a process, allowing the development of a tree

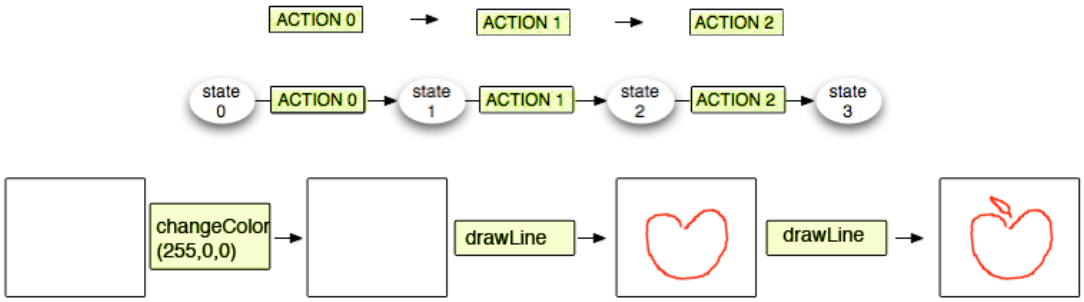

Figure 3-1: Rather than saving all the states of a process, Chronicler saves the sequence of actions that can later be used to recreate states. The figure above shows the action sequence representation and the idea of having actions as transitions and states as nodes. The last row shows the application of this representation in the Paint program.

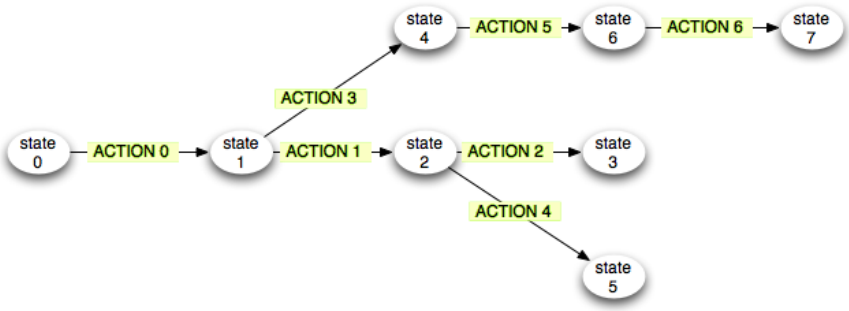

Figure 3-2: The action sequence representation also allows branching and tree structures to form.

structure (Figure 3-2). Also, this representation does not use deletions. Rather than have a delete action in the time sequence, a deletion is done through an equivalent way of undoing an action.

In addition to these benefits, Chronicler's representation effectively allows us to achieve many of the goals we set out for the system.

Domain independence

This representation does not rely on any domain specific information. It can be applied to any application that can be decomposed into atomic actions. Actions are

the types of changes that can typically be made in an application by using tools, menus, or keystrokes. For example, an action in text editing may be typing a word, actions in video editing might be cutting a clip or adding audio. In the case of our test example, a digital paint program, an example action is creating a new brush stroke. For example, the action sequence for Figure 3-1 is a changeColor action followed by two drawLine actions.

Embedded Rationale

Chronicler's action resolution was also chosen because actions are the unit by which most people usually describe and understand a process. For example, a person will typically describe drawing a smiley face by saying "draw two dots for eyes and then a curved line for a smile," not by describing each pixel color. Thus, this representation can support rationale inference because the steps are already described at the resolution at which most people understand processes. The rationale extraction is further supported by allowing each action to be associated with a user-supplied annotation. Annotations open the door for explicit rationale capture rather than inference from actions.

Combinatorial Time Manipulation

This representation effectively captures the time dimension of the design by saving a chronological sequence. The actions are used as combinatorial building blocks. The same basic set of actions can be combined in different sequences in order to create any number of finished products. Thus, like in CVS, this representation enables non-linear editing. Every path of the work progress is saved and can be extended upon or branched out later. This tree can then be crawled action-by-action in order to reach any state of a design. One can start at the leaves of the tree and follow the child-to-parent connections in order to find the root, or any other place higher in the tree along that path.

In such a way, this tree structure allows undo, redo, and revert capabilities. Because finished work in this representation is just a sequence of action steps, in order

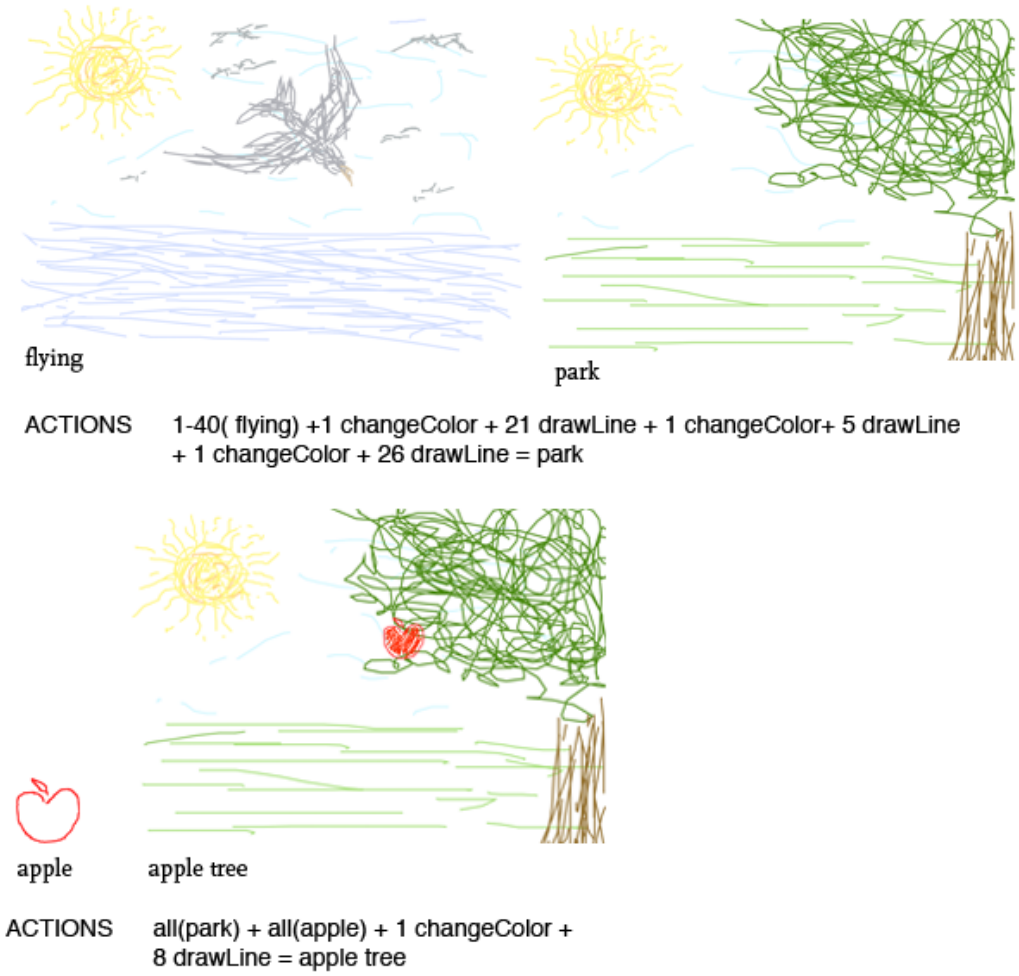

Figure 3-3: Actions from existing painting processes can be reused and combined with new actions to create new artwork.

to add to a work, you can just append steps to the end of the sequence. In order to undo something, the corresponding actions are removed from the sequence. By not having deletions, conflicts in the reordering of steps are avoided. Also, one can easily "replay" the process just by examining the state of the work after each action is executed. Another interesting side effect of this combinatorial property is the ability to take existing process action sequences and splice or mix them together as shown for digital paintings in Figure 3-3.

Figure 3-4: The difference between two digital paintings made can be seen in comparing the two action sequences and isolating the actions that are not found in both.

Process Search and Comparison

Another advantage of this representation is its application to search and comparison. Because all the metadata from the actions is preserved and transparent in the representation used, we are able to search at an action granularity. For example, we can search for all digital drawings with red circles by searching for all the circle-drawing actions that come after a red-color-change action. At a higher level, if we are able to determine which sequences of actions correspond to broader meta-actions, we can search for those as well. For example, if we know that a smiling face is comprised of a circle in the upper left, a circle in the upper right, and a concave arch in the middle below, we can look for sequences that would match this output. The annotations also serve an important role in search because they allow us to identify those higher-level actions and intentions.

In addition, the action-level representation enables easy comparisons between different processes. One can easily perform a difference operation between two processes just by aligning the two chronological sequences and seeing where actions do not match up. Then, because the actions can be easily isolated and recombined, once can pinpoint the exact difference. In the case of a digital painting, as shown in Figure 3-4, one can see the exact strokes that make the two paintings differ because these individual actions can be extracted from the process sequences. This same logic can be used to find the similarities between two processes.

Sharing and Multiple Users

This representation can be applied to a multi-user environment because actions are atomic. Thus, edits to processes can be done in a lock-free environment due to the lack of deletions. The action representations also allow collaborative design work. Asynchronous collaboration is easily achieved and recorded in this system since each additional author's work is just another set of actions appended to the end of an existing action sequence. In an open, shared process environment, this means that any author can extend any other's work. Moreover, synchronous collaboration is possible when each atomic action is at such a low time resolution that there typically is no overlap between action times.

Even in collaboration, all the steps of the process are preserved. Because each step can be attributed to an author, it is easy to attribute the work in the process to its rightful author. The action granularity assures that we can identify who makes which changes because actions are at fine enough a grain to only be done by one person and each deterministic action is relatively the same size.

3.1.3 Chronicler Framework Architecture

We can then use this representation in designing the Chronicler software system. Chronicler has a client-server architecture, with multiple networked, smart clients and one central server. The server acts only as centralized storage of the process information. The logic for process capture is contained within the clients, which can interact with the server via two actions (Figure 3-5):

1. Obtaining information for specified states and actions.

2. Send a state-action tuple that creates a new action change.

Throughout the design of Chronicler it was important to keep the generality and modularity of the system in mind. In order to allow any type of design application to be plugged into Chronicler, the dependencies between the application and the process capture/review parts of the system were kept to a minimum. This was achieved by

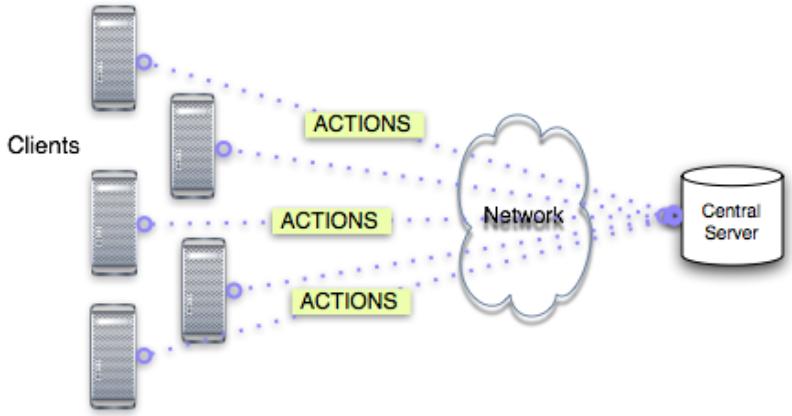

Figure 3-5: Multiple clients can make requests to the central server, obtaining information about states and actions, or adding new action tuples.

isolating all the data changing methods in the application, and interfacing only those with the process capture modules.

The benefit of such a design is that there are simple protocols across all application domains. This design also allows software designers to focus on the creation of the fundamental actions within a process.

3.1.4 Framework Implementation

Chronicler was implemented as a 4,945 line Java client with 982 lines from the painting application used to test the framework. The centralized data storage back-end was implemented using a Postgresql [36] database that involved an extensive schema for storing all user and process data.

The Chronicler client can be decomposed into two main parts: the underlying framework of the process capture and analysis modules, and then the actual design application as shown in Figure 3-6. The underlying framework consists of the specification for the process representation and composition. The framework also contains the mechanisms for centralized storage and data retrieval from the server as well as local caching. Lastly, it contains the UI for process analysis, which we describe in detail in Section 3.1.5.

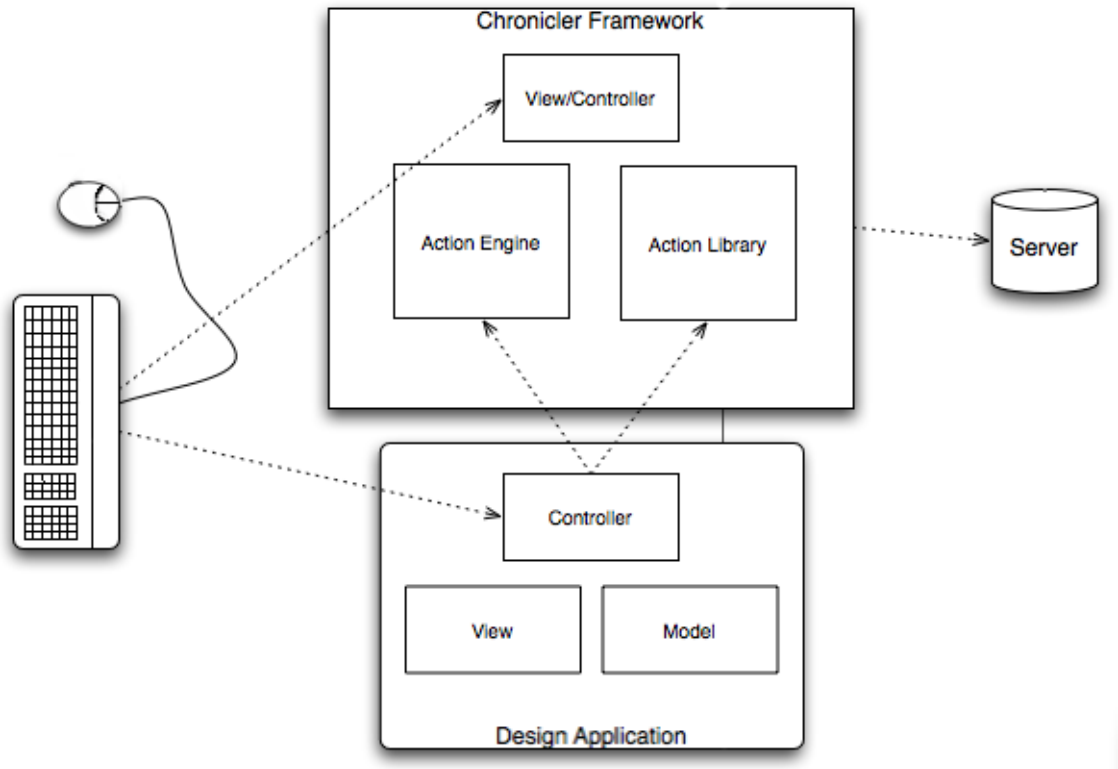

Figure 3-6: The general architecture of the Chronicler system, containing the action management framework and the design application.

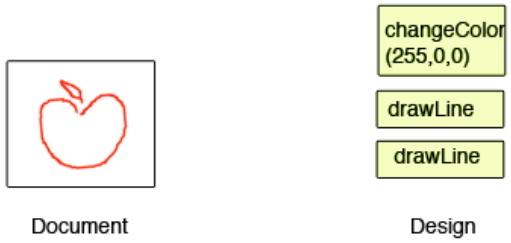

Figure 3-7: A comparison of the Document and Design representation.

Process Data

Every work in Chronicler is represented by two data objects: a Document and a Design as shown in Figure 3-7. The first, the design Document, holds the actual state of the design and represents what is actually seen and manipulated by the user as a design develops. It is analogous to the type of file format used in most design applications where only the most recent state is shown. For example, in a digital painting, it is a bitmap of pixel values. The second, the Design object, refers to the sequence of action events that generate the state of the Document. The Design is the persistent information saved across Chronicler sessions whereas the Document is regenerated by the Design every time it is opened. Each Design has a title, an originating author, the time of creation, and the name of the application by which the Design was made.

The process data for the Design is captured in the following set of classes:

1. Actions: discrete changes specific to an application that can be applied to a type of Design.

2. Transitions: parameterized changes made by users and applied to the Design.

3. Annotations: text descriptions associated with Transitions.

4. Users: individuals using the Chronicler system.

The Actions all implement the iAction interface that specifies a "run" method,

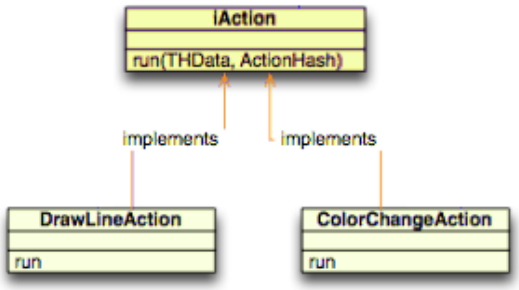

Figure 3-8: The iAction interface and example actions from the Paint program.

which takes in the data object (representing the design work) to perform the action upon as well as the parameters for the action(Figure 3-8). This run method contains the changes that this action performs on the design data. Some example actions from the painting program implemented in Chronicler were ones for drawing lines and changing colors Figure 3-8.

Each time an Action is used, a Transition object is saved into the sequence for the Design. A Transition contains the time of the action, the actual Action data, and a pointer to the parent Transition that came before the current Transition if this is not the first Transition in the sequence. The parent linkage allows Transitions to be chained in tree structures.

Annotation objects are stored as plain text with parameters for the associated Transition, time, and author. The mapping from Annotations to Transitions is many to one. This allows both the artist and other members using the system to add annotations while keeping the time and authorship of these annotations clear. For example an artist might make one annotation while creating the work and later after some reflection may want to make an additional comment. Other artists, after seeing the work may want to comment on interesting steps of the process as well.

To support multiple users of the system, each user of Chronicler has a unique ID to distinguish his or her work. The user also has a unique username and password for logging onto the system. This authentication was implemented using the SMPL

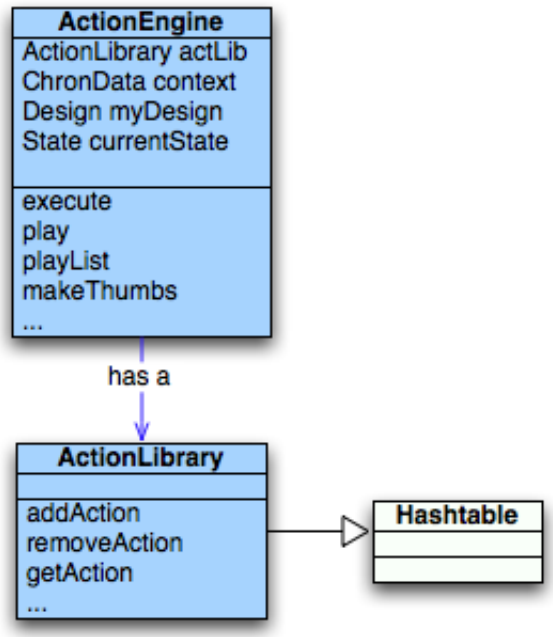

Figure 3-9: The ActionLibrary and ActionEngine. The actions contained in the ActionLibrary are used by the ActionEngine in creating documents.

authentication service [31].

Interfaces

The Chronicler client has sets of applications implemented as interfaces that consist of the following in addition to other domain specific classes (Figure 3-9):

1. ActionLibrary: A set of domain specific actions and execution logic.

2. ActionEngine: The mechanism that actually applies the actions to Documents.

All of the Actions that affect the design in progress are first pre-defined and registered in the ActionLibrary. The ActionLibrary is a hash table of string name and action-object key-value pairs. The Actions in the ActionLibrary are then fetched by the ActionEngine when they are actually executed during the course of the design. Figure 3-10 shows the ActionLibrary used in the example Paint program.

The ActionEngine is responsible for creating the sequence of events that take a blank design Document to a completed Document. It executes actions from the

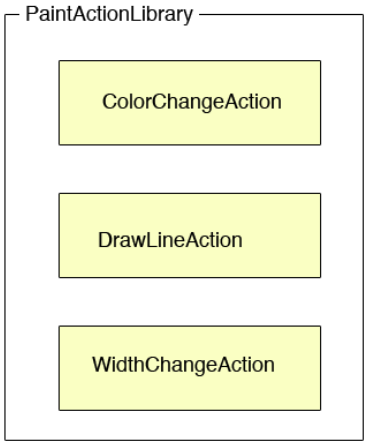

Figure 3-10: A sample ActionLibrary from the Paint program. The simple Paint program included three actions, one to change the stroke color, one to draw lines, and one to change the stroke width.

ActionLibrary as called for by the user by actually making the associated changes to the Document. Each Action execution corresponds to a Transition object that is saved into the sequence for the Design in the database. The ActionEngine is also responsible resetting the state of the Document. Given the tree-structure of the Design sequences, the ActionEngine can replay Actions from the root of a Document to regenerate any state represented by the path ending at any node in the tree.

Design Application

The second part of Chronicler implementation was the application itself. Chronicler supports design applications written in the Model View Controller [22] design pattern commonly used in user interface design. The benefit of this is that all the actions are isolated in the Controller section of the application. For example, in the Paint application, all the edits to the canvas and tool settings were placed in the Controller.

In order to maintain modularity and general application development in Chronicler, the only dependencies between the application classes and the process capture/review system were connections between the ActionEngine and the Controller

of the application. The ActionLibrary for an application could be defined and then each action made by the user via the Controller could fire an action execution in the ActionEngine.

Database Back-end

The Design, Transitions, Annotations, and Users are stored in a separate table in the PostgreSQL [16] database, indexed by unique ids. Each of these classes inherited from a general ChronicleObject class which managed connections to the database (Figure 3-11). The Java client interfaced with the database using the JDBC API [20] for performing SQL queries and inserting data.

The use of database transactions to record the state of designs also guaranteed atomic transactions within the multi-user environment. Furthermore, the design of the system dictated that there would be no deletions, only insertions because no data is removed. This also prevented any locks from occurring on the database. The initial design of the system used a constant stream of Transitions to the database as the end user executed Actions. The motivation behind this decision was to keep an up-to-date reflection of all the design work on the database, so that collaborators could get real-time updates. However, this soon proved to cause lags in performance because of the frequent database connections, possibly by many clients at once. As a result, the streaming of Transitions was converted from individual to batch streaming, at times dictated by the user as in CVS commits.

Finally, in order to facilitate text metadata queries on the database, the Action data within the Transitions was stored as xml using XStream [39] so as to provide transparent data objects in text searches of the database. The Postgresql full-text search feature, Tsearch2 [7] was used for searching through all the text annotations and xml data.

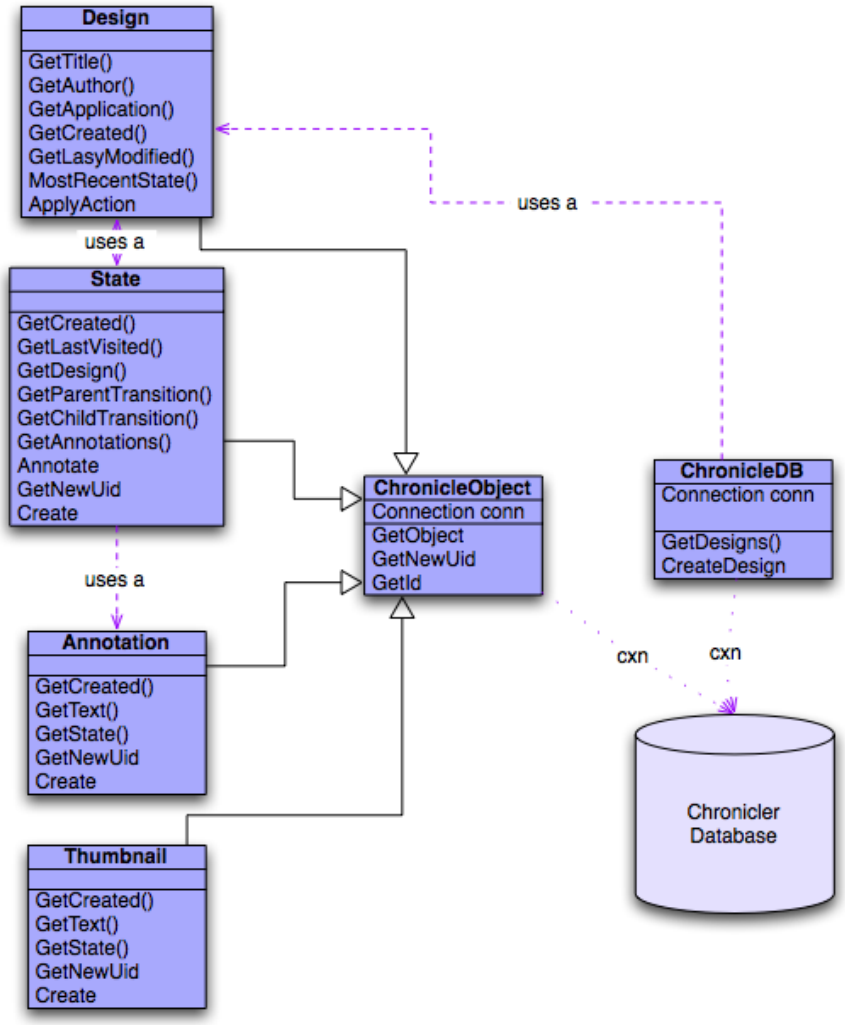

Figure 3-11: The Design, Transitions, Annotations, and Users are stored in separate tables on the central Chronicler database and indexed by unique ids. The ChronicleObject manages the database connections.

3.1.5 User Interface

Chronicler uses a general process capture and review user interface that applies to all applications built within the system. The user interface for Chronicler was designed with the intent that the functionality be as unobtrusive as possible, only aiding the design process, not interrupting it. Thus, in order to shield users from the complexity of the underlying system, the look, feel, and interactions were kept as simple as possible. It was also important to give real-time feedback to users of the capture system so that they could feel the impact of process capture directly. Furthermore, the design was driven by the desire to provide simple but informative process visualizations for the user. These goals motivated the design of the overall interface as well as the three individual process review modules.

The UI for Chronicler is in a separate window from that of the main design application, allowing the UI of the application itself to remain independent. This allows the designer to choose how much of Chronicler he or she would like to see during the design process. The designer can either have the process viewing window right beside the application while the design is in progress as shown in Figure 3-12, or have the Chronicler UI be in the periphery or even minimized so that it can provide no distraction. This also accounts for every design application having the same process capture and analysis UI, providing consistency. The UI design is described with respect to the simple raster painting program, Paint, that was written to test the Chronicler system.

The Chronicler UI consists of four major parts: key-frame capture and review, annotation, process replay, and search. Here we discuss the major design ideas and usability purposes behind each module in the context of the Paint application. Because of limited screen real-estate, the key-frame review, process replay, and search results were not all placed on the screen at once. Viewing each one at a time avoids confusion because each displays process images. Also, each view has a different purpose that does not necessarily need high visibility at all times. In contrast, the annotation viewer/editor and search input fields were made visible at the same location at

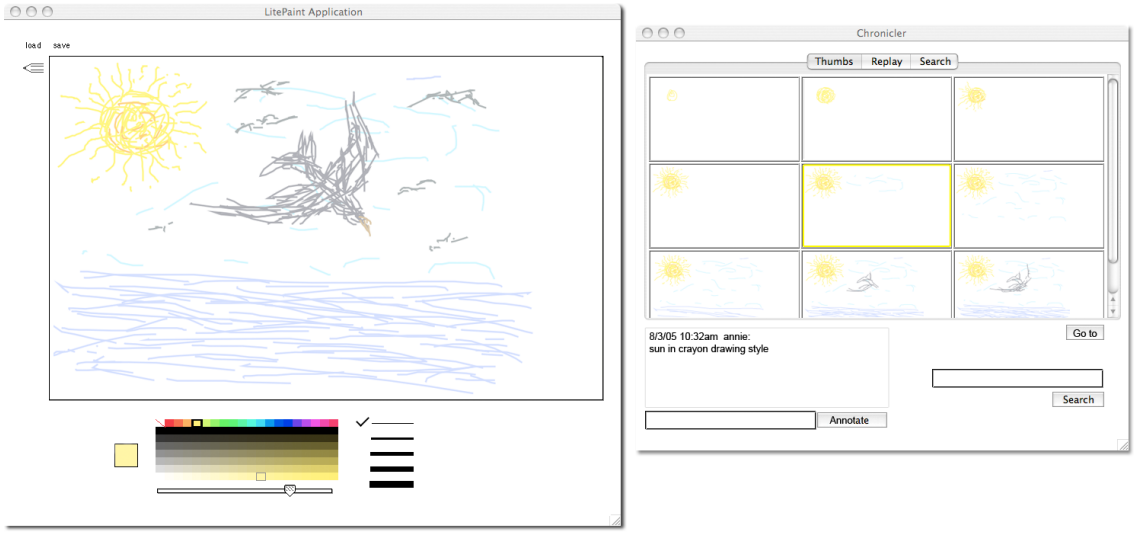

Figure 3-12: The Chronicler UI uses a separate window so that the user can review the process and add notes while using the design application.

all times. This not only gave consistency, but also accessibility so that users could perform searches or edit the annotations related to any of the views.

The final layout used tabbed viewing panes for the key-frame review, process replay, and search results. The annotation and search fields remain in a fixed position at the bottom of the window as shown in Figure 3-13.

Key-frames

As a user makes a creation in an application, it is important that he or she is given the choice to mark key-frames of importance in line with the application so as to minimize interruption of the design process. In this case, Chronicler lets the users do this while drawing simply by pressing the space bar. Alternatively, if this is too much of a burden, key-frames can be captured automatically at a set period of actions.

Each key-frame capture causes a thumbnail of the application's canvas to appear in the frame process viewer, showing that the state has been saved. Thus, this is a helpful view to use while capturing the process because it gives immediate feedback and confirms the capture. At this point, if the user would like, he or she can go into the Chronicler window and add an annotation for that state, otherwise, he or she can

come back later in the process and annotate if desired.

The key-frame feature allows simple browsing of all the major steps in the design process, showing a thumbnail for each key-frame and the time of the action and author. It will also show the associated annotation if one exists. It is a simple visualization for process review, isolating the steps and showing the state of the work at each. The user can navigate to a particular saved key-frame and revert to it, having it load in the creation application, simply by double clicking on the associated thumbnail or selecting it and pressing the "go to" button.

The layout for the key-frame viewer was comprised of a tiled thumbnail interface (Figure 3-13). As more key-frames are added, the thumbnails scale down to fit in the space until they reach a four-by-four grid, where a scroll bar appears to accommodate for the additional screen space needed. This prevents the thumbnails from getting to small to distinguish. The ordering of the thumbnails is chronological from top-down, left to right, complying with English reading and time conventions. Finally, a mouse press can select each thumbnail and the user feedback for this is a border highlight of the thumbnail. A mouse-over on the thumbnails would popup a tool tip indicating the author and time of creation for a particular frame. Lastly, each thumbnail had a beveled appearance to give the affordance of selecting and clicking.

Annotation

The annotations play an important role in the user interface because they allow the user to supply additional information that we could not just automatically capture. However, because the purpose is to explicitly ask for design information, the user interface for the annotations must be simple and easy to use. The annotations are placed in a consistently visible place so as to encourage users to contribute more information. Lastly, it is important that the annotations have a clear association with the writer and time of annotation.

The annotation box was made as a text display area, showing the annotations under an author and timestamp heading. The input area was a single line text box, with the affordance of editing, where annotations could be committed by pressing

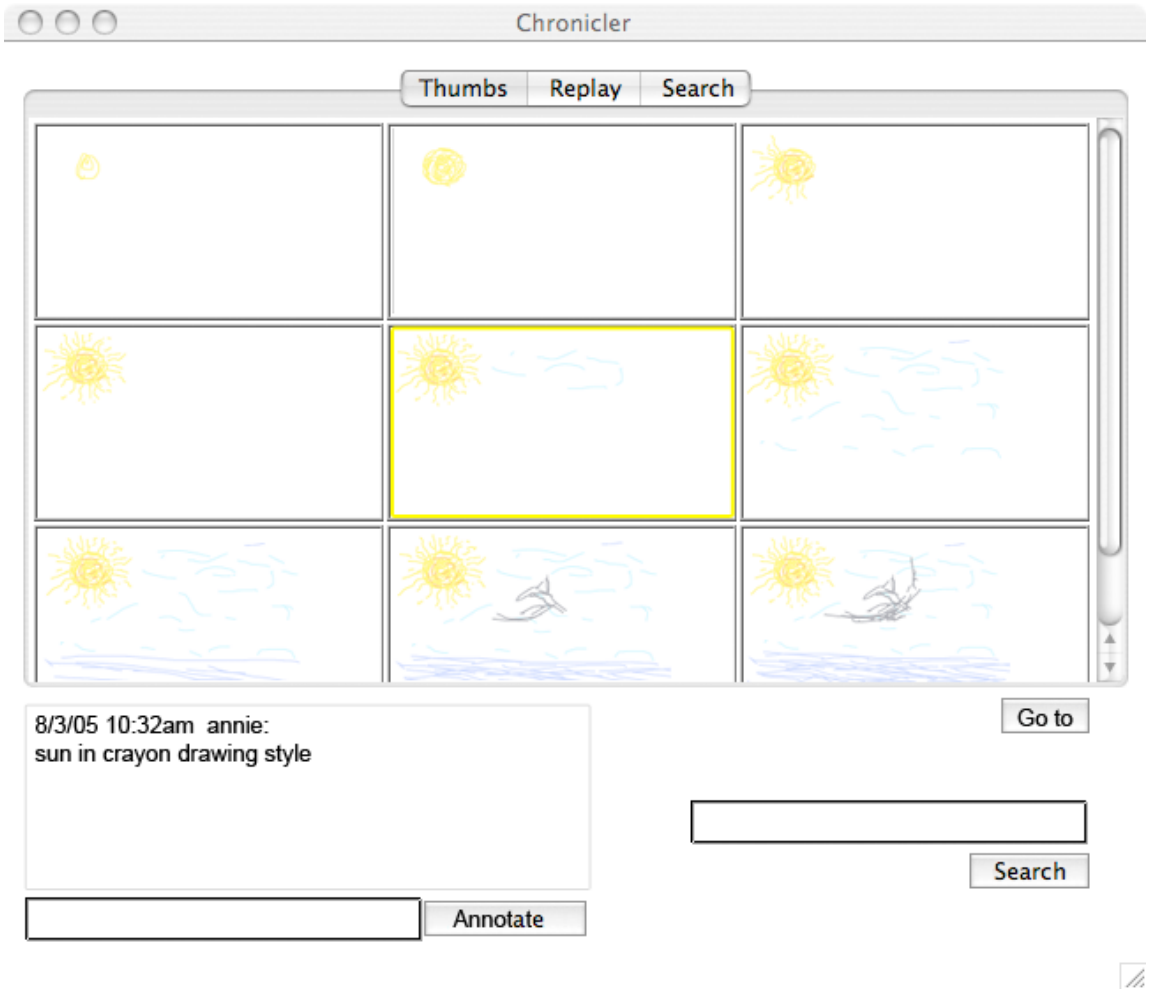

Figure 3-13: The Chronicler key-frame viewer. Each of the captured frames of the drawing is shown in one of thumbnail images. This provides a simple visualization of the process sequence and also a way to navigate the process. To revert to any state shown by a thumbnail, the user can double click on the thumbnail or select one and press the "go to button.

enter or the "annotate" button (Figure 3-13).

Replay

The replay feature allows users to see the process of all the changes being made to the raw data form in an animated movie form. This is a desirable feature to have in process viewing because of the success of video tutorials in education. In the case of the painting program, one can play an animation of the painting as it goes from blank canvas to finished piece. Seeing all the actions being done gives the user additional intuition on the design process beyond just the key-frames because the actual gestures and timing are captured. The user is also able to play, pause, stop, rewind, and fast-forward this viewing process consistent with the typical movie interface. As the individual frames for the actions are being played in the animation, if one of them has an associated annotation, the annotation will also appear down in the bottom annotation field until the next annotation comes up in the play sequence.

The actual layout of the replay feature consisted of one large viewing panel for the animation, with buttons alongside the panel for each of the play, pause, stop, rewind, and fast-forward features (Figure 3-14). The default playback rate was at approximately ten times the speed of the actual drawing. This kept the relative times for each change while speeding up the viewing process so that viewers would not get impatient while watching the process. At the same time, we wanted the animation to still be somewhat broken down so that each step was distinguishable, thus the frame rate was less than 10 fps to avoid perceptual fusion.

Search

The user is able to search for keywords in annotations or painting metadata such as shape and color by inputting search strings in a text box. The search box is accessible at all times in the Chronicler window, and when a search is executed, the search tab is automatically brought into focus with the search results. The search results then return in order of relevancy in the form of key-frame thumbnails (Figure 3-15). Like with the key-frame view, the user can select any thumbnail to see more detailed

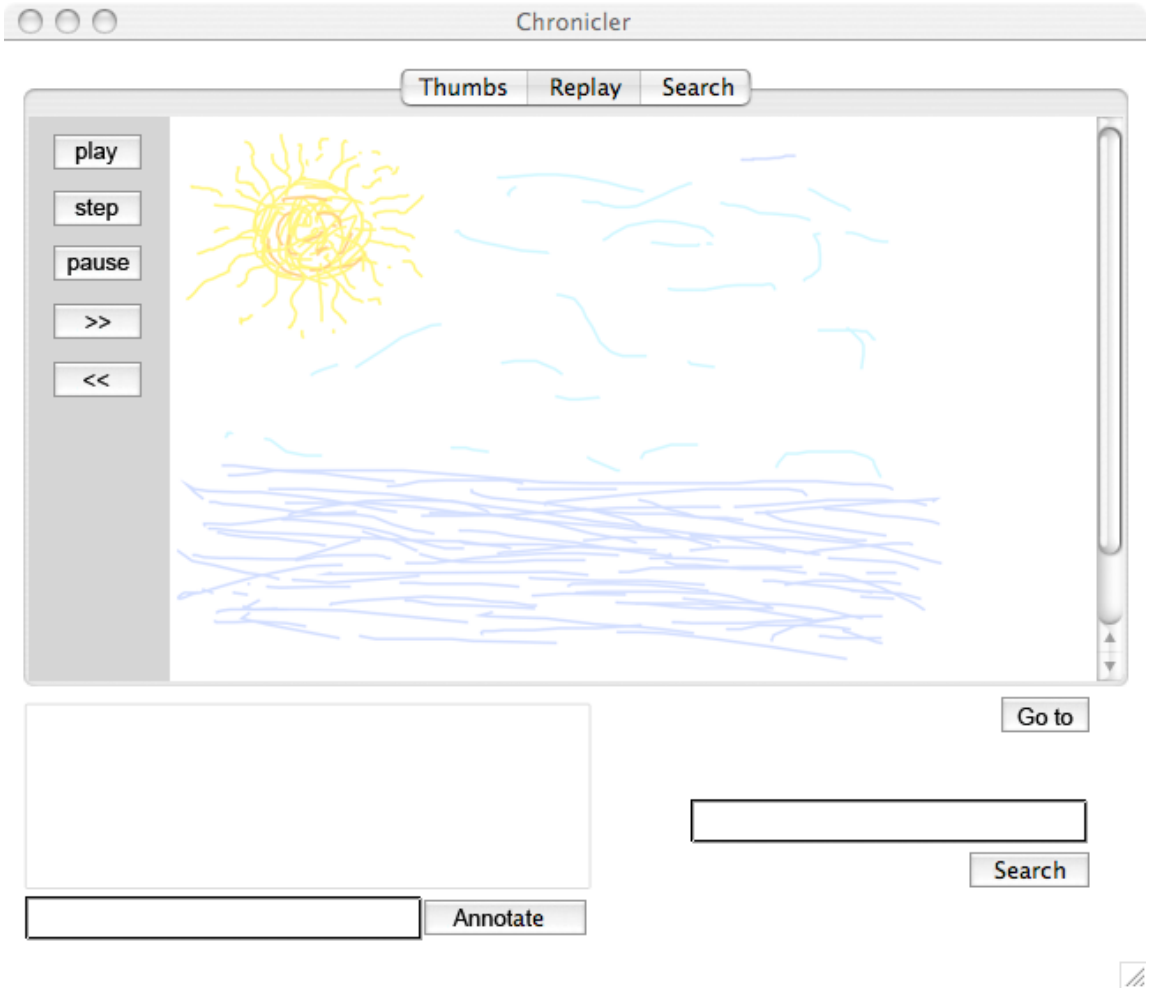

Figure 3-14: The replay feature allows users to replay the process in chronological order as one would a movie.

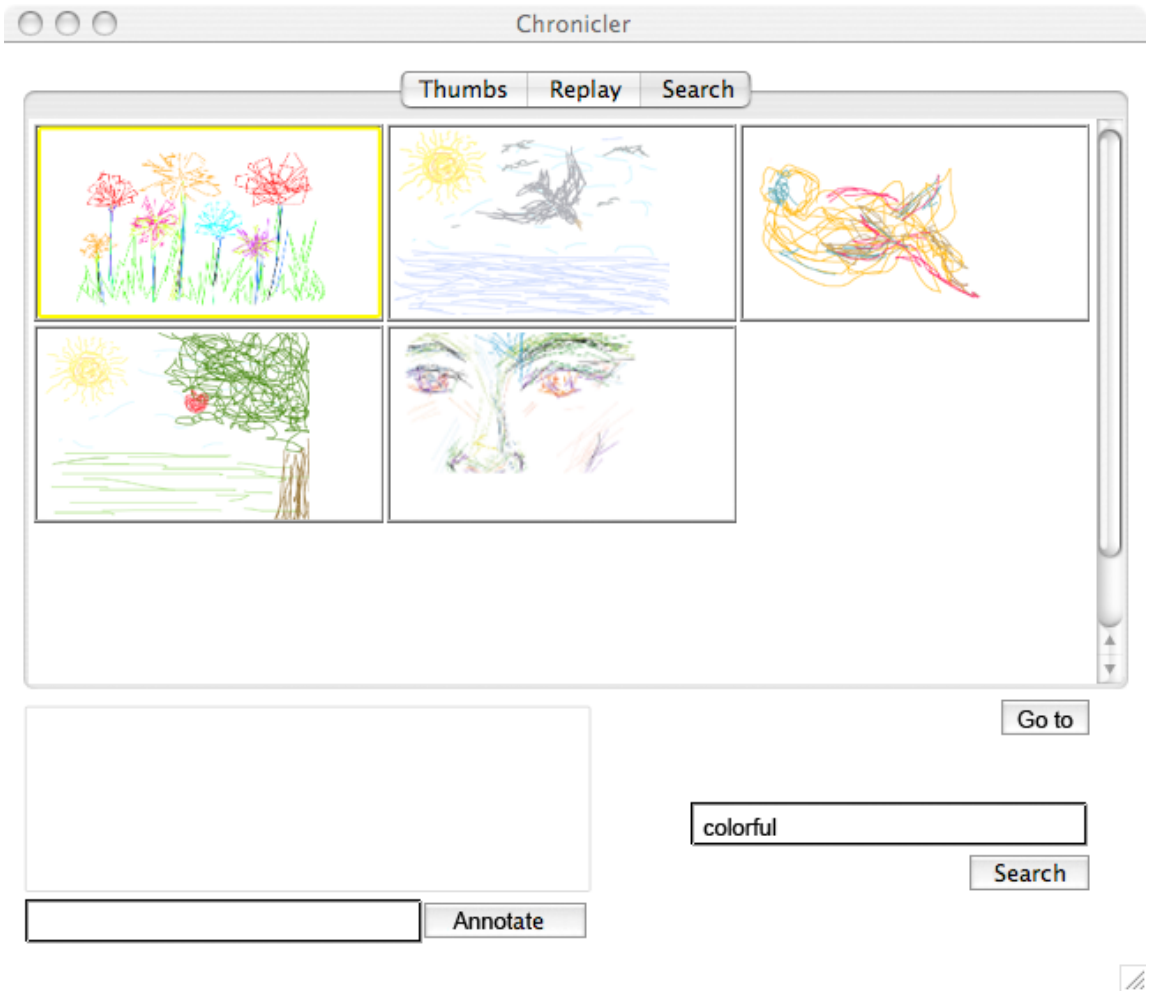

Figure 3-15: The search feature allows users to search through all the annotations and metadata collected from the processes stored in the repository.

information on the creation and annotations involved, or double click on a thumbnail to load that piece. For example, the user can search for the word "face" and find all the pieces that have been titled or have parts annotated as a face. The user can also search based on visual metadata such as "blue circle."

3.1.6 UI Implementation

Chronicler's user interface was built by extending the widgets provided by the Java Swing library. In order to keep simplicity in both the implementation and the experience of the UI, it was also designed as multiple interacting modules, with one

UI widget for each of the major features. Chronicler loosely follows the Model View Controller [22] design pattern. Each UI module uses data compiled from the Action Engine as a Model, with a combined View and Controller as we will describe.

The key-frame module used a Design object as the model. View and Controller were made with multiple Thumbnail widgets, one for each key-frame. Thumbnails were an extension of JButton widgets representing transition states. Each key-frame was captured from within the Paint application by adding key and mouse event listeners to Paint. A mouse-up event on the canvas signaled the end of a drawing action, a space-bar event signaled the recording of a key-frame.

The replay module also used a Design model, and used a series of view updates fired by an animation timer to provide the playback functionality. The annotation module was based on a Transition model, using standard Swing text widgets. Finally the search module also used the standard text widgets for input, and displayed results using the Thumbnail widgets.

In the case of the Paint application, each of the images used for the views were generated by using the ActionEngine to crawl the process tree and then creating bitmaps to represent the state of the paintings. An important back-end feature implemented to support the fluidity of the UI was image and data caching. The cache was necessary on the client end because the navigation of the process tree via the ActionEngine meant that many states needed to be created from scratch quite frequently. Thus, the state relevant information was cached on the client end using a first-in-first-out eviction policy.

Caching was also used on the database end as each major key-frame captured resulted in the creation of a Transition with an associated thumbnail image. This additional caching aided in browsing key-frames that were not in the cache, and also in generating the results in process searches.

3.2 Art Genealogy

The Art Genealogy system was bootstrapped off of the existing OPENSTUDIO project framework, and as such, many of the design decisions resulted from the integration with the existing system. It leverages the OPENSTUDIO database-backed web framework and extends it to include process capture without interfering with the original system. The motivation behind the process capture design in the Art Genealogy system is to capitalize on the important process elements discovered in Chronicler and take a more coarse-grained approach, saving only that information that is needed to make sense of the creative process behind a piece. It was also designed to be minimally invasive to OPENSTUDIO making the fewest underlying changes possible in order to attain the goals of domain independence, combinatorial time manipulation, process search and comparison, multiple user support, rationale capture, and user friendly interfaces.

The OPENSTUDIO project is a web-based environment for creating, buying, selling, and collecting creative works in a community (Figure 2-7). For the purposes of discussing design, this section will only describe those parts of OPENSTUDIO that impacted the design and usage of the process capture system. A description of the project on whole can be found in the Related Work section. Because the process representation was influenced to a large extent by the design of the existing OPENSTUDIO framework, this section first addresses the framework on whole and then the resulting process capture representation.

3.2.1 Framework Architecture

OPENSTUDIO follows a client-server architecture, with a two part client end consisting of the design application for creation and the web browsing environment for browsing, collecting and buying or selling. OPENSTUDIO has a centralized server for persistent storage of the created documents. This server stores all of the work created by the design applications, as well as all the users, buying and selling transactions, and user input tags. Because of this centralized storage, and the hundreds of

client requests from the application or web that can occur at any time, the atomicity of transactions is vital to the system.

OPENSTUDIO also includes a application development framework in which other programs such as a paint program, sound editor, spreadsheet maker, and text editor have been made. Each application is networked and fetches user and document data from the central server in the form of a completed document. Whenever a document is saved, it adds a new document or updates one on the server. The web clients make connections to the same server to retrieve and display data on the design works or add transactional data.

The applications made for OPENSTUDIO each use their own domain specific data models and only the raw byte data is saved on the server. Thus, unlike in Chronicler, each action made while using an OPENSTUDIO application may not follow the same form and specification. As of this writing, the only application publicly released on the OPENSTUDIO website is the Draw program which I originally implemented and then extended to support process capture.

3.2.2 Process Representation

The motivation behind the process capture design in the Art Genealogy system is to capitalize on the important process elements discovered in Chronicler and take a more coarse-grained approach, saving only that information that is needed to make sense of the creative process behind a piece. In order to accomplish this, the system uses two levels of representation, versions and then chronological sequences of data changes. This is demonstrated through the use of a simple drawing program.

Versions

The version control extension provides a coarse look at the design process, giving discrete samples of the process over large periods of time. This can be done by saving the entire state of a design work at a given time and then giving each a parent. If the work is the first, original version, it has no parent. However, if a piece is derivative

from another, that other work will be the parent. Each work version is created by a single author. This creates a chain of authorship or creation genealogy as shown in Figure 3-16.

I considered many alternatives for adding versioning capacity to the system. One alternative was emulating the design of CVS or SVN [9] [30]. However, this would require drastically restructuring the OPENSTUDIO document storage and services for additional functions such as difference calculations, much in the fashion of Chronicler. Furthermore, within the OPENSTUDIO system, different versions of the same course of work can be owned and modified by different users.

Timestamp Acts

The main motivation of the finer grained process capture was to get an overall intuition on the chronological steps of a specific version, a level of detail not captured by the versioning described above. This can be captured by embedding a timestamp to each discrete action captured by the representation of the design work captured. This motivated the use of a vector drawing application as a test application because a vector drawing in itself is already divided into discrete parts, where each vector shape could be considered a timestamp act. This preserves the time dimension that is usually lost in the final saved document. Also, these timestamps can be used to both order the actions chronologically and to estimate the time for steps by taking differences between timestamps (Figure 3-17).

Domain Independence

The goal of domain independence is fulfilled by the version tracking because each work only needs to have a parent attached to it, a change general enough to be applied to all works in the system. However, the detailed action process tracking with the timestamps involves tightly coupling the capturing system with the specific application and application data type used, such as vector shapes.

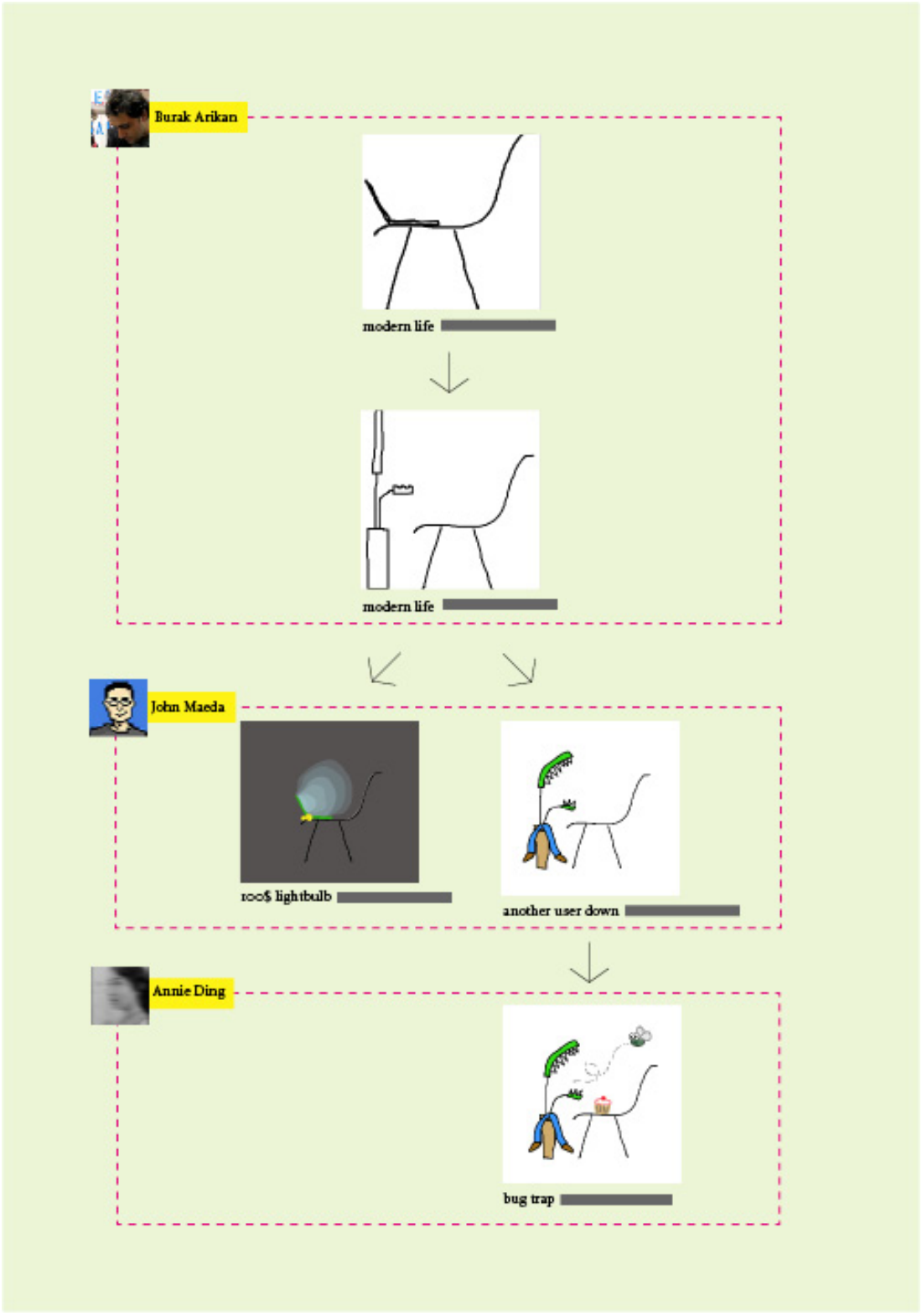

Figure 3-16: The versioning system allows derivative works to be tracked and creates a chain of authorship. Here the piece at the top is the root version.

Figure 3-17: Each action, in this case the drawing of a vector shape, is marked with a timestamp in milliseconds. These timestamps let us know the chronological order of the shapes as well as estimate the time to draw each shape.

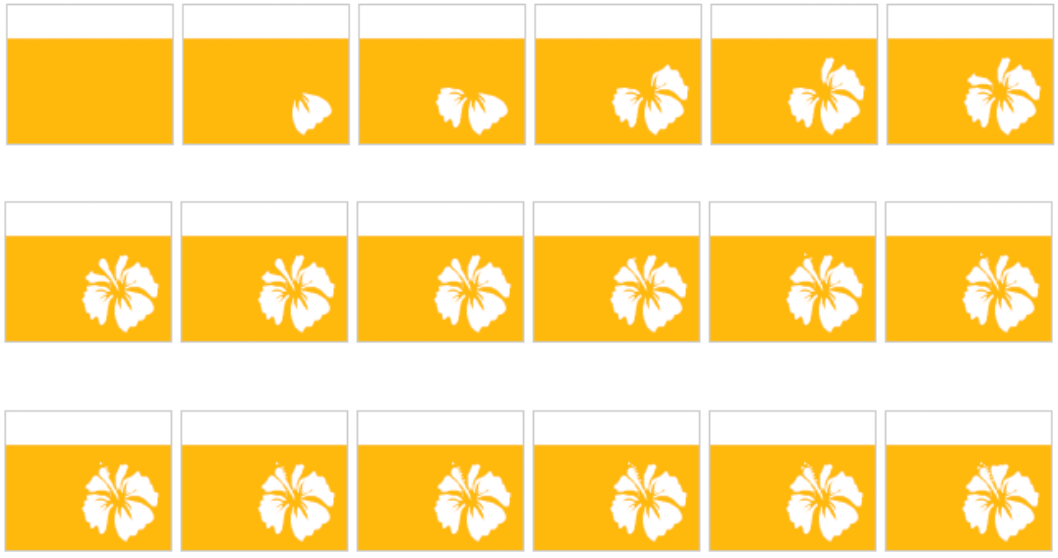

Figure 3-18: The timestamp information allows us to reconstruct a step-by-step sequence of events, shedding light on the design process.

Rationale Capture

The rationale captured by this system stems mostly from having a chronological estimation of actions. The timestamp acts allow us to regenerate the sequence of events behind a finished work as they were originally made. From the timestamps shown in Figure 3-17, we can infer the following process shown in Figure 3-18. Some additional rationale is captured by the tagging aspect of OPENSTUDIO because the tags add additional semantic information.

Combinatorial Time Manipulation

Because each timestamp act, or shape in the case of a vector document, can be thought of as a discrete step, this approach still allows combinatorial use of the steps because the shapes can be arranged in different ways to create new pictures. Like in Chronicler, the versioning steps can be combined in a tree-like manner. For example, if piece B was a second version built off of piece A, piece B would have A as a parent. The detailed timestamp steps from two different pieces can also be appended to one another in the case of a vector drawing application, to create a new piece placing the

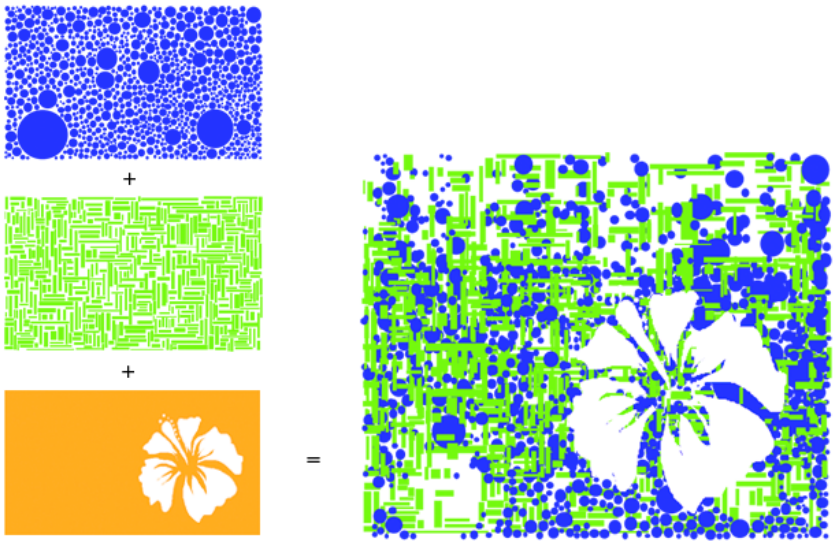

Figure 3-19: A random selection of timestamp acts are selected from existing drawings and then recombined to make a new drawing.

contents of the second in front of the first (Figure 3-19). Also, different groups of shapes can be reused across multiple pieces.

Sharing and Multiple Users

The simple versioning allows for asynchronous collaboration between multiple people on a piece. For example, Burak can draw a picture of a chair, John can then buy that drawing from Burak and modify it, and later Annie can buy the John's image and make additional changes as in Figure 3-16. Not only does this generate a trail of versions, but a trail of authorship as well. However, collaboration is not possible within one version because of the single author property.

Search and Comparison

The search functionality in Art Genealogy is more limited on a detailed scale because there are no action abstractions and the only information that each raw document gains is a timestamp. However on a larger scale, versions and version sequences in Art Genealogy can be compared as action sequences in Chronicler were compared.

3.2.3 Implementation

OPENSTUDIO is a framework written in Java, with Java application clients and a PostgreSQL database back-end. Versioning was implemented in OPENSTUDIO by making a simple extension of OPENSTUDIO's Document object structure. All applications save their data in the form of a Document object. This Document object includes the bytes of the actual design data such as all the shapes in a vector drawing. It also includes the application type that created it, the time of creation as well as the title, author, owner, and price. To accommodate versioning, each Document retains the same structure as described before, with an additional parameter for a Document parent.

The implementation of seamless version information capture is done in applications by tracking first to see whether or not a Document was opened from another piece. If this is not the case, then the Document is considered an original and has no parent assigned in the parent field. If the Document was opened from another, then a pointer to the original Document is saved as a potential parent. Upon saving, if the Document is to be saved as a new version, the original will be assigned as the new parent. However, if the Document is just saved over the original, then the parent assignments will not change. If at any point the Document is cleared by the user, then the parent field is set back to null.

The Draw application was implemented in the OPENSTUDIO framework as a 5,432 line Java web start application, using Hibernate [19] to retrieve the serialized Document objects from the same database. Again, the design of the applications follows the MVC design pattern [22], with the model data fetched from and saved to the server. Also, the isolation of all the data modification code in the Controller aids in tracking the series of actions/changes in the system.

The vector objects in Draw were created by extending the Shape Java object. Each saved Draw document is a list of Shape objects ordered by z-order. Although it is straightforward to see how to draw an ellipse with an ellipse tool or a rectangle with a rectangle tool, the same intuition is not trivial with a free-form shape. Thus,

each free-form polygon Shape also reveals chronological information by retaining the order of the points in the polygon. The timestamps on the Shapes are taken from the local client. This is sufficient because we are interested only in relative times within one version.

3.2.4 User Interface

Draw Application

Like the Chronicler Paint application, Draw was a simple, but powerful application, capable of capturing process information with minimal effort on the part of the user. Draw has a minimal look and feel and with only the minimum amount of tools necessary to create a drawing. Because there are so few tools, it was possible to keep all the tools and options visible to the user at all times without occupying too much screen real estate. However, the user also had the option to hit the minimized-view at the top left corner of the window to hide the tools and menus. The user could also hide the color picker by sliding the bottom scroll bar. Figure 3-20 shows screenshots of the Draw UI in both modes.

There were many reasons for the limitation on the number of tools. The original Draw tool had many more tools and features, however, we found that a limited number of tools could let users focus more on the drawings and not get side tracked by choosing the right tool. Furthermore, it makes process tracking simpler, limiting the combinations of actions to the general polygon domain in the vast field of artistic possibilities. Artists can push the limits of the existing tools, accomplishing many of the same tasks with the few tools provided as they would have with more complex tools. Fewer tools also resulted in interesting processes and sequences of actions that together could mimic a more complex tool. Still, this approach can scale to support even more tools and actions.

The detailed process capture occurs within the Art Genealogy system invisibly. An artist can use Draw and be completely unaware of the process data being saved because it is embedded in each element that he or she adds to the canvas. At each

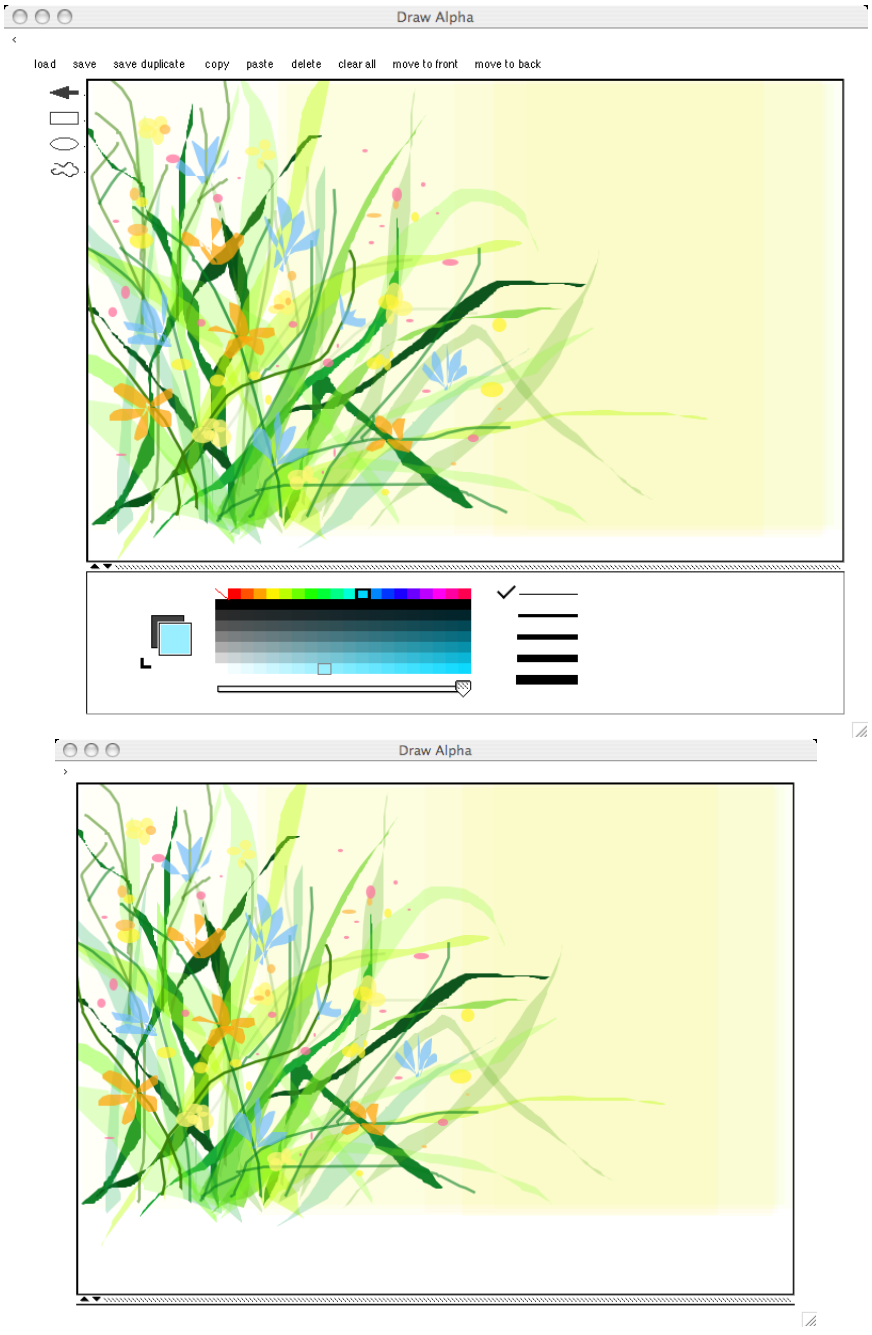

Figure 3-20: Draw, a simple vector drawing program, in it's default and minimal view.

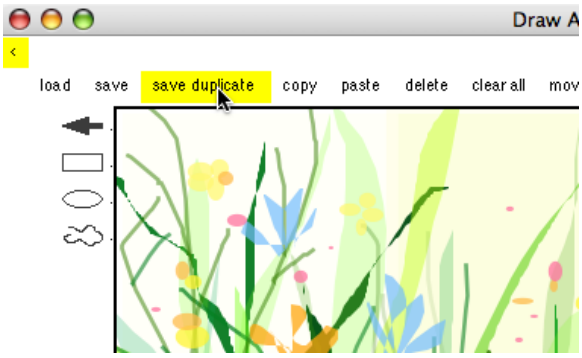

Figure 3-21: The "save duplicate" shortcut button in Draw for saving versions.

Shape drawn, the application takes a timestamp. The only burden on the user is the burden of explicitly saving his or her document. By doing this he or she is either editing a version or adding a new version to the end of an Art Genealogy tree. Any work that starts as a blank canvas, even if it was originally another piece and then cleared using the "clear all" function, is considered a root Document in the Art Genealogy. If an artist wants to edit an existing version, if she is the original creator, she can directly save over the existing version if desired. However, if she wants to save a new version of work, she must explicitly save the piece as a new version, which can be done with either the same name, or a new name. The process of saving a new version is expedited by the "save duplicate" function (Figure 3-21), which automatically saves a new copy. If someone who was not the originally creator of a version wants to edit it, he cannot directly overwrite the original. A new version is automatically created in this case.

Version Viewing

The versions in OPENSTUDIO can be browsed from the web interface. Each piece of artwork in the system has it's own web-page. The goal is for browsers of the site to get a feel for the "genealogy" of a piece at a glance, but be able to navigate it upon closer inspection. The genealogy of the piece can be viewed at the bottom of each page as shown in Figure 3-22). If the piece is a leaf in the version tree, then the genealogy will contain all the versions from this leaf back up to the original root in the form

of thumbnail image links. The user can follow the trail of version progression simply by following the thumbnail links in the genealogy. Arrows between the thumbnails indicate the direction of tree growth. Each version thumbnail is also accompanied by a link to its author's page, showing the name of the author. Also, mousing-over the author's name indicates the time at which the version was made.

Detailed Process Viewing

In addition to the Draw application, an additional simple application was implemented to support process viewing. This application allows users to compare processes by visualizing them in filmstrip, step-by-step views. It also provides playback of the creation process of a piece in movie form, and let users "mix" the processes of multiple pieces. The design of this application also reflects the minimalist look and feel of the Draw application it complements.

The filmstrip viewer, as shown in Figure 3-23, allows a user to open up any number of drawings he or she owns and see the chronological buildup of the piece, shape by shape. The user can add pieces from the list using the "Add Process" button, and the same file opening interface as used in the Draw application. Later, pieces can be removed from the list by using the "Remove" button. The frames of the pieces go in chronological order from left to right, as is the standard time direction convention, with each frame differing from the next by the next shape that was added chronologically. If there are too many steps for the window, scroll bars will appear, allowing the user to scroll along the process filmstrip. The strips are arranged so that each step of each piece takes the same amount of horizontal space. Then the user can compare the complexity of each piece by directly aligning the steps between different pieces. Additionally, because many pieces have very many steps and go far off screen, there are three set zooming levels for the thumbnails: small, medium, and large.

The playback feature in the Art Genealogy process viewer, as shown in Figure 3-24, works similarly to that of Chronicler. The user can choose a piece to watch the process of, and then play or pause the viewing. Each frame of the animation is essentially one of the frames in the filmstrip viewer. That is, the user watching the

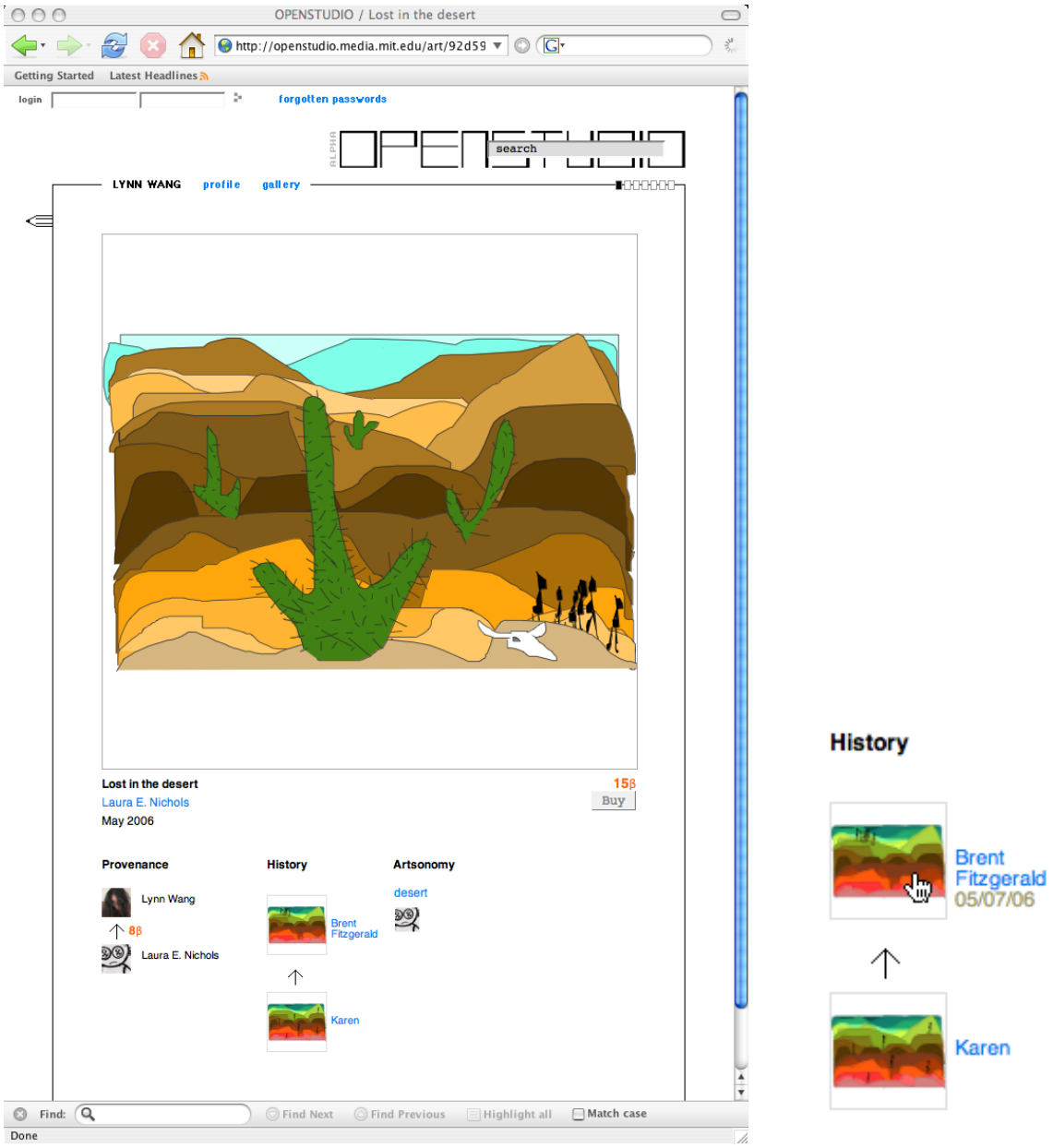

Figure 3-22: The versions in OPENSTUDIO are show in the History feature below each piece. The user can mouse over the images to see the date of creation and click on the image to go to the version, or the author's name to see the author's page.

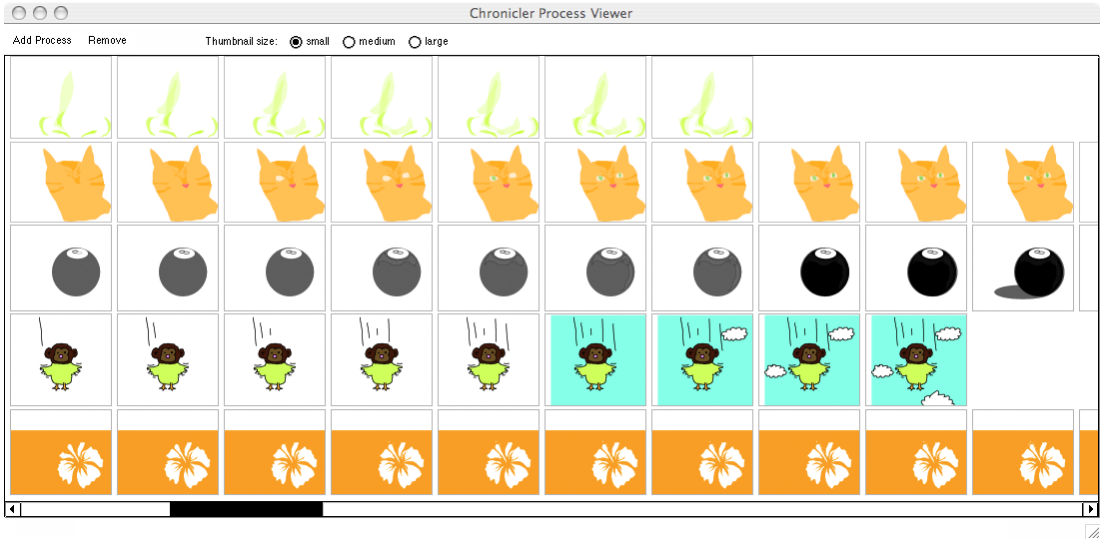

Figure 3-23: The filmstrip viewer allows users to open art pieces and see the step-by-step break down of processes. It also allows users to compare different processes by juxtaposing them on the screen. Lastly, the thumbnails are viewable at three sizes for close inspection as well as broad overviews.

process will see each shape show up over time as in it's creation chronology, finishing with a view of the final product. The time for each frame is proportional to the time it took to draw the shape added by that frame. That is, a frame that adds a shape that takes a long time to draw will be shown proportionally longer. In order to provide a compelling viewing experience, the playback occurs at ten times the speed of the original drawing, cutting out the delays that occur from long breaks taken by the artist.

Lastly, the process viewing suite has a novel feature that allows you to "mix" the processes of existing pieces. Similar to how a person can mix music, this lets users cut a drawing along the time (process) axis and mix, match, and reuse shapes or shape sequences to create a new sequence, resulting in a new drawing. This has been implemented in a proof of concept form by simplifying the cuts to randomly generated segments as shown in Figure 3-25. This puts the least burden on the user, allowing the user to choose finished pieces, by selecting their titles, and then randomly piecing together a new drawing using parts of the selected ones.

The full version of this feature still has limited usability, allowing users to indicate

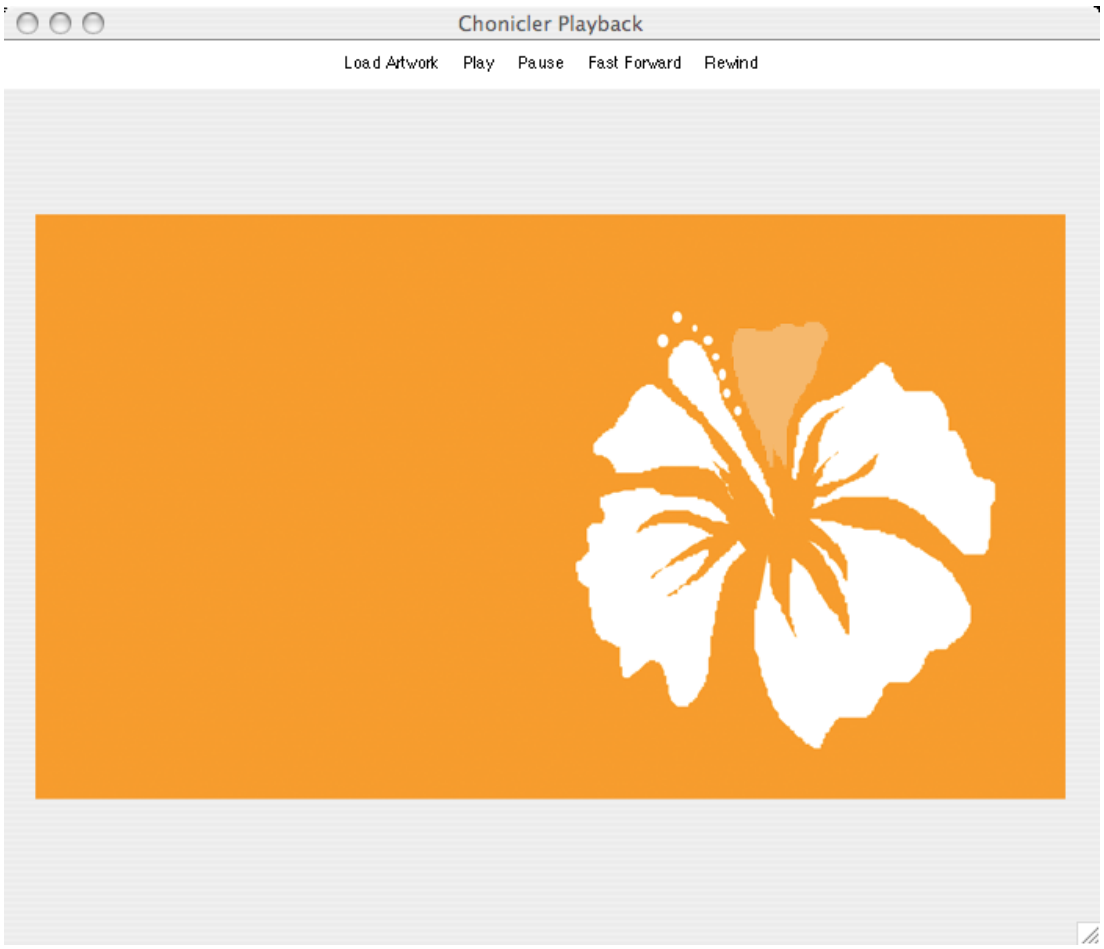

Figure 3-24: The playback feature allows users to watch an animation of the creation process, played in times proportional to the original creation process.

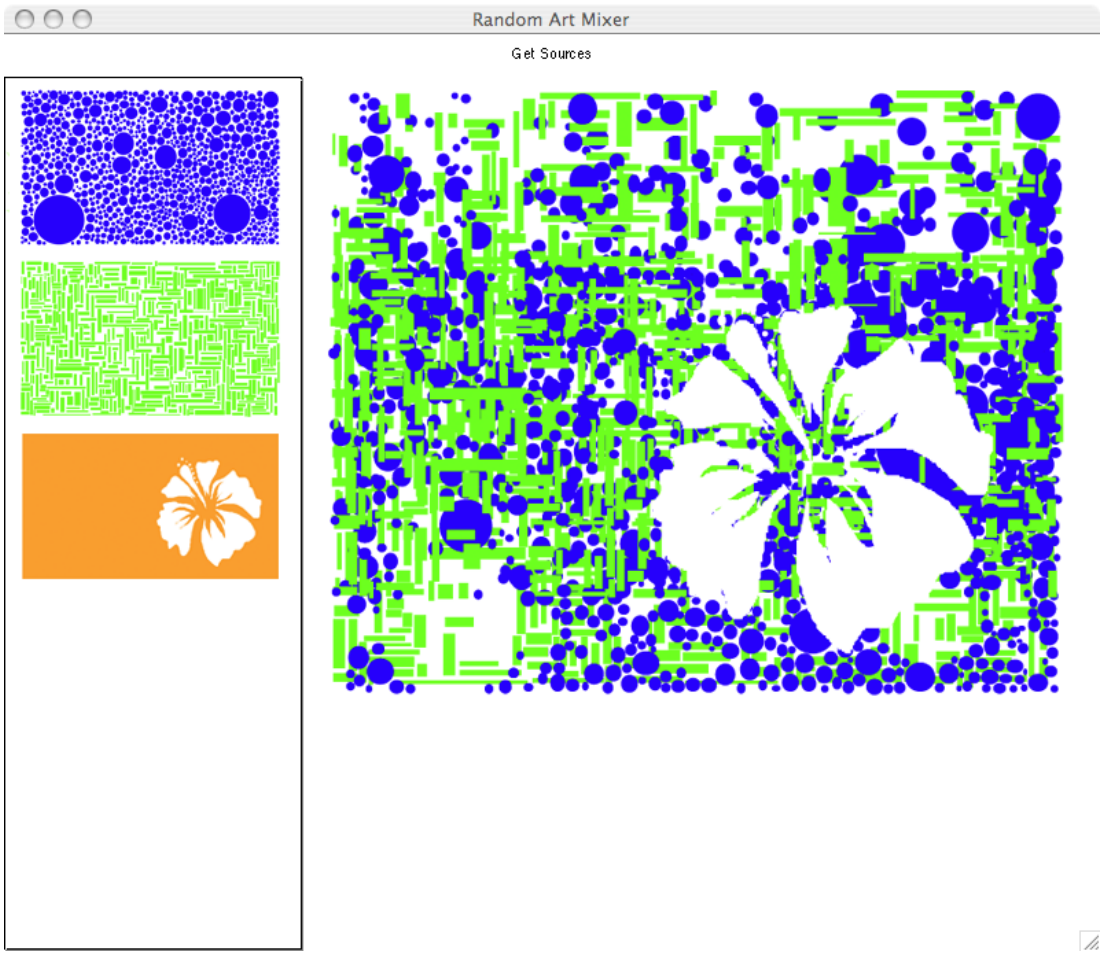

Figure 3-25: The Random Art Mixer lets users select existing source drawings and then randomly selects shapes to generate a new drawing.

how much of a piece to use and in which sequence through less user-friendly text, command line like inputs.

3.2.5 UI Implementation

Like Chronicler, the front end of the Draw application was built by extending Java Swing UI widgets. The separate process viewing application was also implemented using Swing in the form of a 1,233 line Java application. It also used the standard minimal OPENSTUDIO Java look and feel for consistency across applications.

The web interface of the Art Genealogy system was implemented using Ruby on Rails interfacing with the same PostreSQL database back-end used by the applications. This included factors such as the Ruby Document object retrieval as well as remote rendering of the Documents on the server end for displaying the images on the web pages.

Chapter 4

Results

The Chronicler and Art Genealogy frameworks give unique perspectives on the strengths and weaknesses of these differing approaches. This section presents an evaluation of both systems by comparison of the applicability of the systems for process capture and review, focusing most on the dimensions of representation, usability, and collaboration. It also presents the feedback resulting from in-person user studies from both systems. Additionally, it exposes the observed emergent behaviors that resulted from giving an online creative community this capture and review system. Finally, it presents the results from extensive data mining and statistical analysis of the large corpus of user generated data captured by the Art Genealogy system. These results and observations provide further evidence for the benefits of process capture in collective creativity, specifically in the realms of learning, recycling ideas, attribution, valuation, and expression.

4.1 System Design Comparison

The comparison of Chronicler and the Art Genealogy systems reveals the strengths and weaknesses of each in capturing and reviewing the art creation process. In terms of design, the biggest difference between Chronicler and Art Genealogy was in representation choices that then affected the usability and collaboration capabilities. However, some of the strengths and weaknesses of these two systems also result from

factors aside from the designs of the systems themselves. Most notably, the Art Genealogy system was capable of unveiling many interesting user phenomena described in the following sections because it was able to bootstrap off of an existing, active creative community.

4.1.1 Representation

Generalizability

One important dimension of representation is its generalizability, i.e. whether or not the representations can be generalized to capture any process. In this respect, Chronicler's representation has an advantage. The base framework provides a specification for defining the individual action building blocks that can create a process. Furthermore, all of these action types are defined in accordance with a general interface and then registered in a library. Because almost all digital design applications work via menu actions and tool applications, almost any application can be broken down into discrete actions that fit into this general mold. Thus, Chronicler's action sequences can be used for painting and drawing applications, but also for text editing, or spreadsheet applications.

In contrast, the timestamp process capture representation used in the Art Genealogy system allows for little generalization. The representation used to capture the Draw process is embedded in the specific data format used for making shapes. As a result, this process capture mechanism would not be able to extend beyond process that require more than creating shapes. However, the versioning representation, by virtue of its simplicity is easily generalizable. The simple addition of a parent parameter per work could apply to all domains. In fact, this versioning ability applies to all documents within the OPENSTUDIO system, including those created by applications such as a sound editor, pixel editor, and spreadsheet maker.

Combinatorial Use

In addition, both systems focused on utilizing combinatorial properties of representations. The combinatorial use of the actions in Chronicler was essential to its generalizability. By using actions combinatorially, one can in essence capture the entire process space through a small set of actions. This representation can then mimic the step-by-step characterization of the creation process in which many tools are used over and over again in different ways. Although defining these base actions adds some initial overhead to the design of the application itself (finding and isolating the elementary atomic actions), the benefits from combination outweigh that overhead. The combination of actions in Chronicler or shapes in Art Genealogy lets us define work through sequences of discrete steps. Thus, we can generate new sequences or mix processes merely by recombining or reusing pieces of old sequences.

Using combinations of basic building blocks also allows us to search the process space more easily. Similarities and differences between processes are also easy to spot in the structure of the combination sequence. For example, we can see when the same actions are used in similar sequences to determine similarity. In this respect, Chronicler holds an advantage as the action-level tree allows for the application of graph algorithms which make the differences easy to extract.

A side-effect of the ability to use the representation combinatorially is the ability to create branches in the work process, rather than just keeping the process linear. This worked especially well in the case of Chronicler because one Transition could have any number of transition children and all children were stored uniquely. Similarly, the parent-child document relationships were able to capture the tree structure of the processes in Art Genealogy. However, in the case of the individual Draw documents with time-stamped shapes, because the only the final product of the drawing was saved, only one linear branch of the actual draw tree could be preserved within one document.

Granularity

The granularity of representation also had a major effect on the systems. In the case of Chronicler, the representation was intermediate but still relatively fine-grained on the spectrum between capturing each and every user gesture and just recording the finished product. The action level representation was useful because it was the closest to how a person would describe the steps of a process. Furthermore, capturing work at an action level in a tree-like manner meant that deletions were unnecessary and that all information could be captured. However, capture at even the action level had performance ramifications in the regeneration of pieces from the underlying action representation, as described later.

Art Genealogy approached the issue of representation granularity from two directions by having a dual layer version and process timestamp representation. While each version is a very thick time-slice of the process, reflecting a finished product of some sort, the timestamp metadata embedded in each version lets us unravel the fine grain parts of the work. The major weakness of the timestamp approach used in Art Genealogy was that it only reflected work that was reflected in the final state of the work because the timestamps were embedded in the Shapes. Thus creation steps that were deleted would not be preserved. Unlike in Chronicler applications, the time stamped Shapes in Draw were relatively fast to unpack, because they added very little overhead to the generation of the underlying vector drawing.

Although space was not a major concern in the design of both systems, the different granularities of representation also affected the space requirements of the systems. In Chronicler, the space occupation grows linearly with the amount of transitions or actions per document. The drawback with this approach is that the data cannot be compressed to smaller than the size of the action itself. An alternate approach used was compressing sequences of transitions together. This was shown in the example of compressing each sequence of lines between every mouse-up action in the Paint application. A major contributor to the space used in Chronicler is from the thumbnails associated with the key frames marked by the user. However, these cached

thumbnails were necessary to aid performance.

In contrast, in Art Genealogy, the space for each document could potentially be compressed by a much greater factor because the only information necessary per Draw document was a list of shapes, each of which was defined only by coordinates, color and time. However, many documents contained redundant information across different versions because the full state rather than just the difference in changes is saved. Lastly, both systems sacrificed space for the sake of transparency. Both systems used an xml storage format to facilitate searching, which could have been made more compact with an alternative representation.

4.1.2 Collaboration

Chronicler's support for both asynchronous and synchronous collaborative design is powerful, but the full potential remains unrealized because there is no established user base for the system. In contrast, one of Art Genealogy's biggest strengths was the large voluntary user base consisting of over 200 people across the world as of this writing. Multiple factors contributed to this difference. First, the Art Genealogy project was web based, making accessibility easy for most users. Most importantly, Art Genealogy was bootstrapped off of OPENSTUDIO, which maintains user interest for a large part because of the strong community that has developed behind the project. Because the users feel that they are part of a community, they are more likely to interact and collaborate especially as they see other members of the community doing so. The community aspect of the project and the open display of information and interaction also helps propel the viral spread of emergent behaviors, which are discussed in a later section.

4.2 User Studies

To further assess the usability differences between the two systems, a series of in-person user tests were conducted on both Chronicler and Art Genealogy. The studies consisted of approximately ten users with relatively strong technical backgrounds and

some art or design experience. The data collected from the hundreds of remote users of the Art Genealogy system is discussed in the following section.

Both systems demonstrated that a very simple design application was not only sufficient, but actually advantageous for testing the art capture process. In some cases, because of the limited tool set, artists accustomed to massive, feature heavy suites of tools were forced to rethink how they drew. For example, at first the Draw creations were very simple and polygonal, but as artists matured and time went on, the power of Draw's simplicity was made clear. Simplicity meant newcomers could easily pick up the Paint or Draw skills. Artists began to think out of the box and were able to accomplish and surpass many of the same tasks as those they would do with complex tools. This resulted in a large spectrum of styles, and thus the collection of a wide range of art processes.

Both Chronicler and Art Genealogy were successful in providing simple visualizations that could help the user understand the process behind a piece. The combined key-frame and animation approach gave the users both a static and dynamic look at the original creation process over time. One aspect of process viewing that this research did not fully address was the ability to visualize branching information in linear space. That is, given the small browsing space that is available in process capture interfaces meant to complement an existing application, it is difficult to show the whole tree of process development in a meaningful way. This is a problem that much UI research, especially in domains such as web browsing, continues to try to solve.

4.2.1 Chronicler

The biggest advantage of the Chronicler user interface was the ability to view process and create concurrently. Because both aspects of the framework could be viewed simultaneously, processes could directly influence the painting. Users did not seem to find the process of marking key-frames through pressing the space bar distracting while drawing. Also, the user was able to watch the pieces of the process added to the timeline as she painted, giving her much more feedback on what to capture or

annotate.

However, the usability of Chronicler was most limited by the performance speed of the system. The action level representation chosen resulted in long generation times for pieces, because they were not stored in an immediately viewable and editable form. For example, if a person wanted to edit a very large document full of many painting action strokes, the user would have to wait as each action was re-applied to a new blank canvas, bringing the piece back up to the correct state. During the initial design, when every action was immediately sent back to the central server to enable synchronous collaboration feedback, the lags in visual updates while painting were sometimes long enough to cause the users to become very impatient.

In order to counter this, I added image caching, so that thumbnails of images would not have to be generated on the fly in the visualizations. Additionally, the user interface was changed so that updates to the central server would only occur when the user wanted to set a keyframe, not for every action. These measures greatly sped up the performance while drawing, to the point where users were not too bothered by the less frequent small delays. However, users still felt a noticeable delay while waiting for the paintings to regenerate when opening a work.

4.2.2 Art Genealogy

Unlike Chronicler, Art Genealogy did not suffer from performance troubles because the documents could be generated and unpacked quickly due to the representation characteristics we have discussed. However, because the Art Genealogy worked across two platforms, the Java application platform and the Ruby on Rails web-platform, the system parts could not be as tightly integrated and the feedback in some situations was slower. It would have been advantageous in some respects to have the process reviewing functionality in the same space as the search and browsing done on the web. However, the difficulty in unpacking the Java drawing data for the web resulted in implementing the process viewing functionality in a Java application.

On the flip side, this abstraction truly made the process capture invisible to the user, achieving the goal of making the application as unobtrusive as possible. In

Draw, the user does not have to do any additional work for process capture to occur while drawing. Initially the process capture was built into the Draw application without any user interfaces for version browsing or process reviewing. Thus, prior to the launch of the user interfaces for Art Genealogy process reviewing, none of the users of the Draw application had any idea that the process capture was occurring.

When the process reviewing application was shown to users, they initially felt somewhat exposed, not only because the "mystery" behind their process was revealed, but also because the mistakes they made were visible through the process viewer. However, they were very intrigued and sometimes to surprised by their own processes and were eager to view the processes of other artists in the community. They felt that the application was simple and straightforward to use, however as predicted, some users were curious to see some of the deleted process steps that were not captured by the final Shapes in Draw. Some users also expressed interest in a more interactive process reviewer that allowed editing in addition to viewing processes.

The versioning system had a very big impact on the dynamics of OPENSTUDIO as we describe in the next section. In terms of usability, most users understood the concept of the version genealogy of a piece which tracked authorship. However, in the UI, this feature was juxtaposed with the "provenance" of a piece, or ownership history, and some users confused the two at some points because the two trails overlapped in ownership at places. Prior to using the versioning, many of the users asked to clarify whether each version was a distinct work that could be bought/sold and displayed or if the versions were all lumped under one piece of artwork for display. The idea that each version was it's own piece of artwork was clarified once they saw pieces with multiple authors in the version tree. Finally, after using the version saving feature in Draw themselves, and seeing the impact on the display of the version history of a piece, they felt that they understood the system much better.

4.3 Emergent Behaviors

As of this writing, Art Genealogy draws upon over 250 users from OPENSTUDIO who have created over 5,000 drawings using the Draw tool. The members of OPENSTUDIO span countries across the world and many age groups. Even the skill level of the artists covers a wide range, from elementary school children to world renowned designers. The experimental environment provided by OPENSTUDIO allows members to be free to create and manipulate artwork in ways that real world models for art creation and ownership do not allow. As a result, many interesting behaviors have emerged from the project. This section details the emergent behaviors stemming from the Art Genealogy aspect of OPENSTUDIO. They are not only interesting from the perspective of social interaction, but also provide further evidence of the five benefits process capture in collective environments provides (described in Chapter 1): learning, recycling ideas, attribution, valuation, and expression.

4.3.1 Learning and Understanding

The creation process served as a source for learning how to draw as well as a source of inspiration for drawings. The process reviewer in essence provides step-by-step visual tutorials for how to draw. If an artist on OPENSTUDIO is interested in learning how to draw a cat, he or she can simply buy a picture of a cat and see the drawing broken down into small steps that are easy to replicate(Figure 4-1). It also lets a user view multiple ways of approaching one subject. For example, people can compare different drawings to learn approaches to drawing a face (Figure 4-2).

Learning from the process viewer also helps to dispel some of the mystery behind the making of a piece. For example, "Self portrait," by Francis Lam (Figure 4-3), an amazingly complex pixel style work, stunned many of us working on the OPENSTUDIO project when we first saw it. Some members of the team suspected that the drawing was generated by a software bot, however the process analysis gave us insight into how he created the piece. The analysis indicated that the drawing was made in a non-systematic manner, characteristic of how an artist might think about a

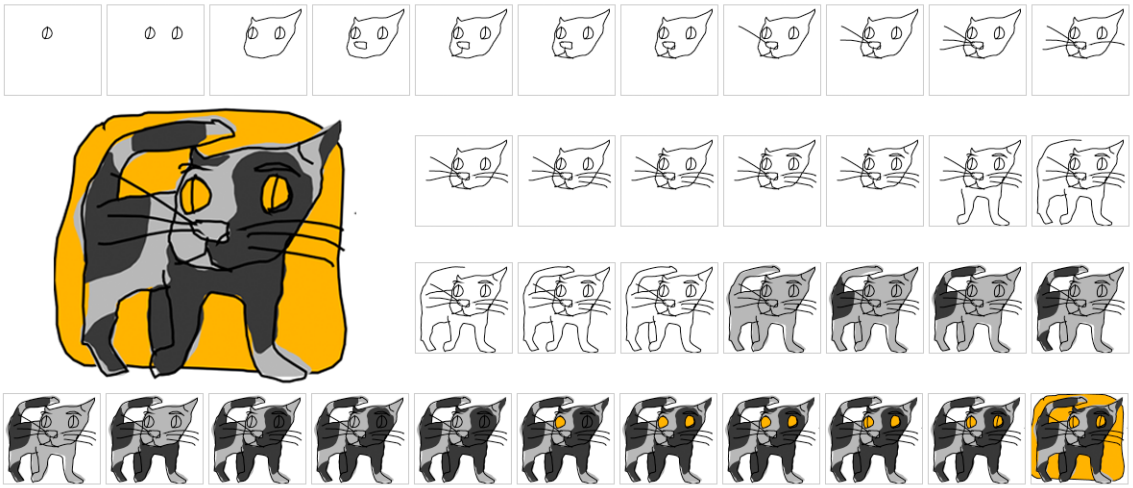

Figure 4-1: To learn how to draw a cat, one can look at OPENSTUDIO member Luis Blackaller's cat piece "felino1".

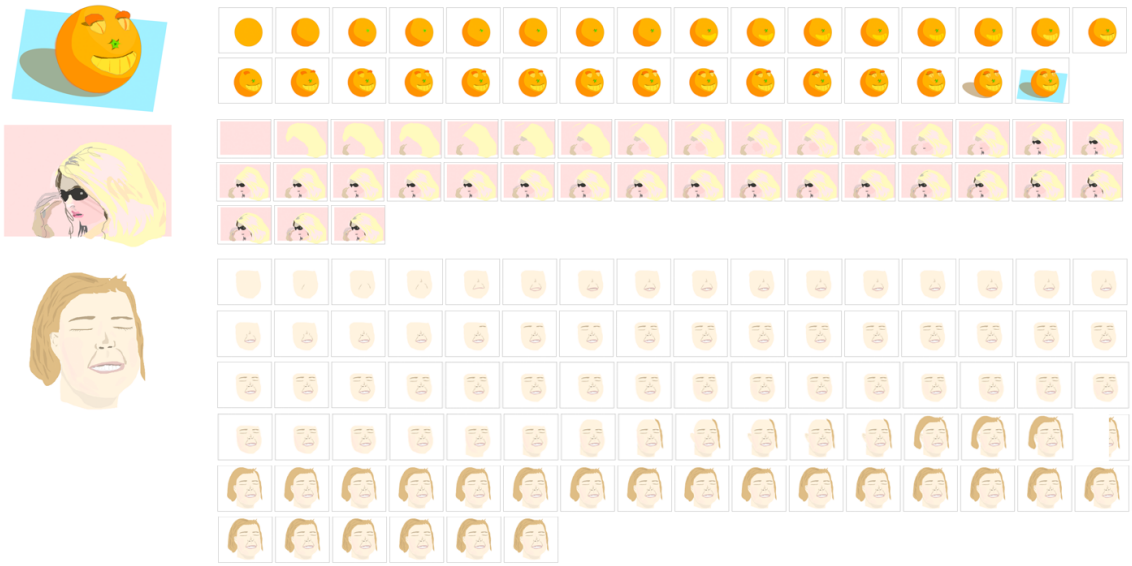

Figure 4-2: To learn how to draw a face, one can compare different processes for drawing a face. For example we can compare C. Connie Yeh's "I didn't know they had faces 2" (top) with Amber Frid-Jimenez's "susan in the sun" (middle), and John Maeda's "nobody" (bottom).

work, but not a machine. Furthermore, the analysis showed that the work contained slightly imprecise pixels, mistakes that a machine would not make.

In contrast, the work by OPENSTUDIO's "TheShepherd" does not seem extraordinary at first glance (Figure 4-4). However, if you look at the process, you can see that the work was actually created with a bunch of tiny strokes at a very fast speed. Although we do not know TheShepherd and we were not able to see how he made his drawings, by just looking at the process (Figure 4-5), we were able to gain enough insight to conclude that TheShepherd actually commissions drawings of sheep from outside of the OPENSTUDIO environment and then transcribes them in the Draw tool, stroke by stroke through using a computer program.

4.3.2 Design Reuse

The idea of mimicking work is taken to the next level when you consider the ability to recycle parts of previous artwork and create derivative works. This is demonstrated through many works of art within OPENSTUDIO. After the unveiling of Art Genealogy, many more people began to buy artwork strictly with the intent of reusing parts of the work or making extensions to the work. In some cases, different work was reused by the original author, as the basis for new artwork, such as in the penguin and calligraphy pieces shown in Figure 4-6. The ability to buy and edit artwork in OPENSTUDIO also allows other authors to edit a work. For example, The "Hawaiian Flower" piece was bought from the original owner and then used almost like a stamp (Figure 4-7).

Other artists choose to extend a work along a new and interesting direction rather than just using a small piece. The extensions range from very elaborate as in the case with the development of the "chair" pieces, to small additions such as in the "turtle" pieces shown in Figure 4-8. The trails of editing often extended across many authors, with each purchasing a version of the piece to tack on an additional version to the version genealogy tree. Since the launch of Art Genealogy in February 2006, over 706 derivative works have been made in OPENSTUDIO.

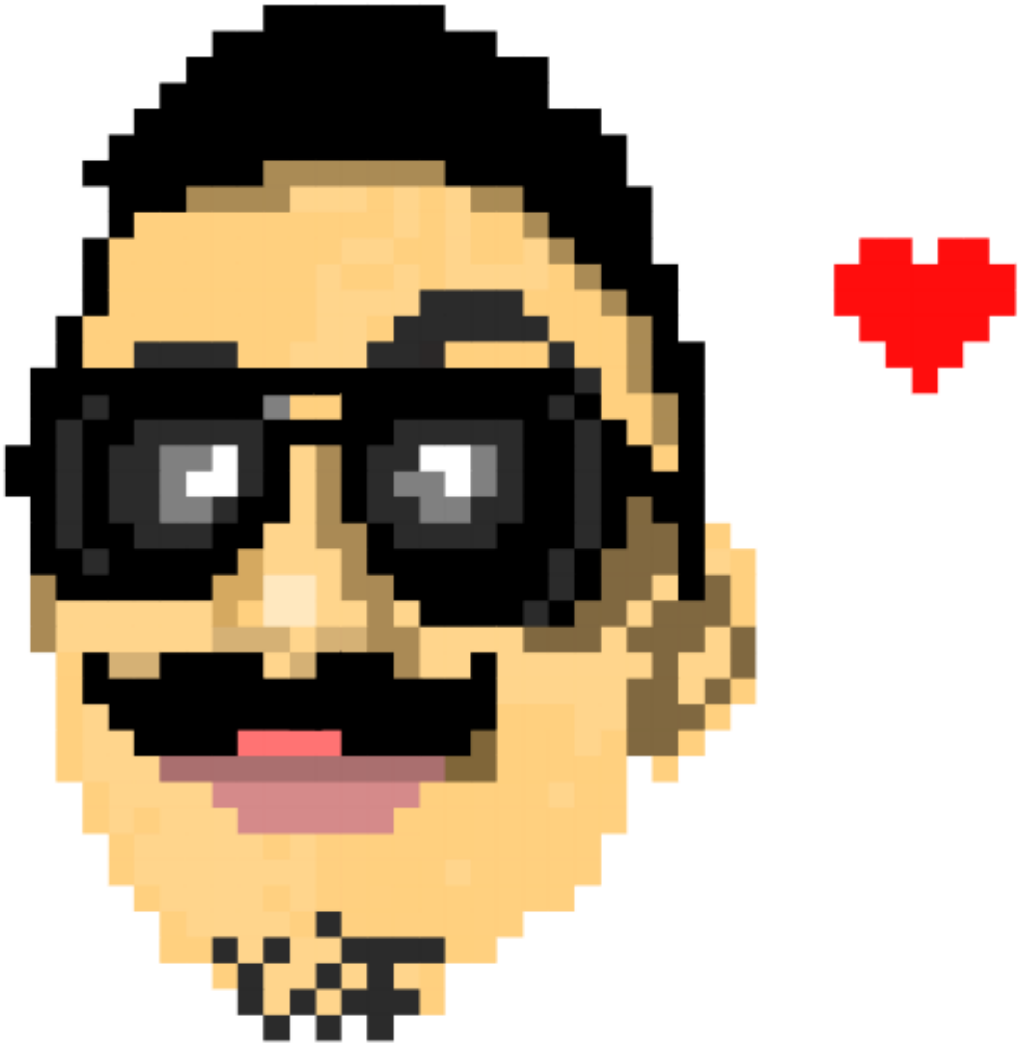

Figure 4-3: Francis Lam's "Self portrait" is one of the most complex pieces in OPENSTUDIO, with over 1,500 shapes. The amazing work piqued the curiosity of many people in OPENSTUDIO. Viewing the process behind the piece allowed us to uncover some of the mystery behind how the work was created.

Figure 4-4: A sample of the many sheep sold in TheShepherd's OPENSTUDIO gallery.

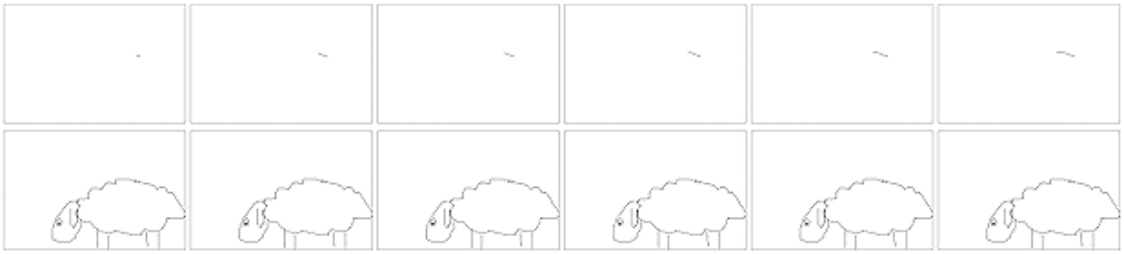

Figure 4-5: The first and last six steps of the 280 step process behind one of TheShepherd's sheep drawings. The fact that these drawings were all done at a very fast, and consistent rate, with incremental changes allows us to conclude that the drawings were made using software bot, not a person.

4.3.3 Identification and Attribution

However, this ability to mimic, copy and extend work brings about questions of how to appropriately attribute authorship. Derivative works can range anywhere from blatant copies, where it seems that the original author should retain all the credit, to creative extensions, where successive editors ought to deserve credit for their contributions as well. Prior to the release of Art Genealogy, copiers of work were sometimes able to make blatant copies just by re-saving a work and passing themselves off as the authors. Furthermore, people who made creative extensions off of work could not indicate which works they were working from.

The versioning system provided by Art Genealogy is a solution to many of these problems. By clearly indicating the chain of authorship on a work, it guarantees that anyone who copies a piece will be revealed as having done so because the community can see that the previous piece by a different author was exactly the same (Figure 4-9). It has made community members who are buyers more aware if they are not getting the original value of a work and artists are ensured that others cannot take credit for their work. Furthermore, it has encouraged more users to collaborate because derivative works become considered as tributes rather than infringements.

In addition, the process capture helps us in cases of pure forgeries, where one person has copied the likeness of another work from scratch and thus there is no authorship trail to follow. In trying to determine which of two similar pieces is the

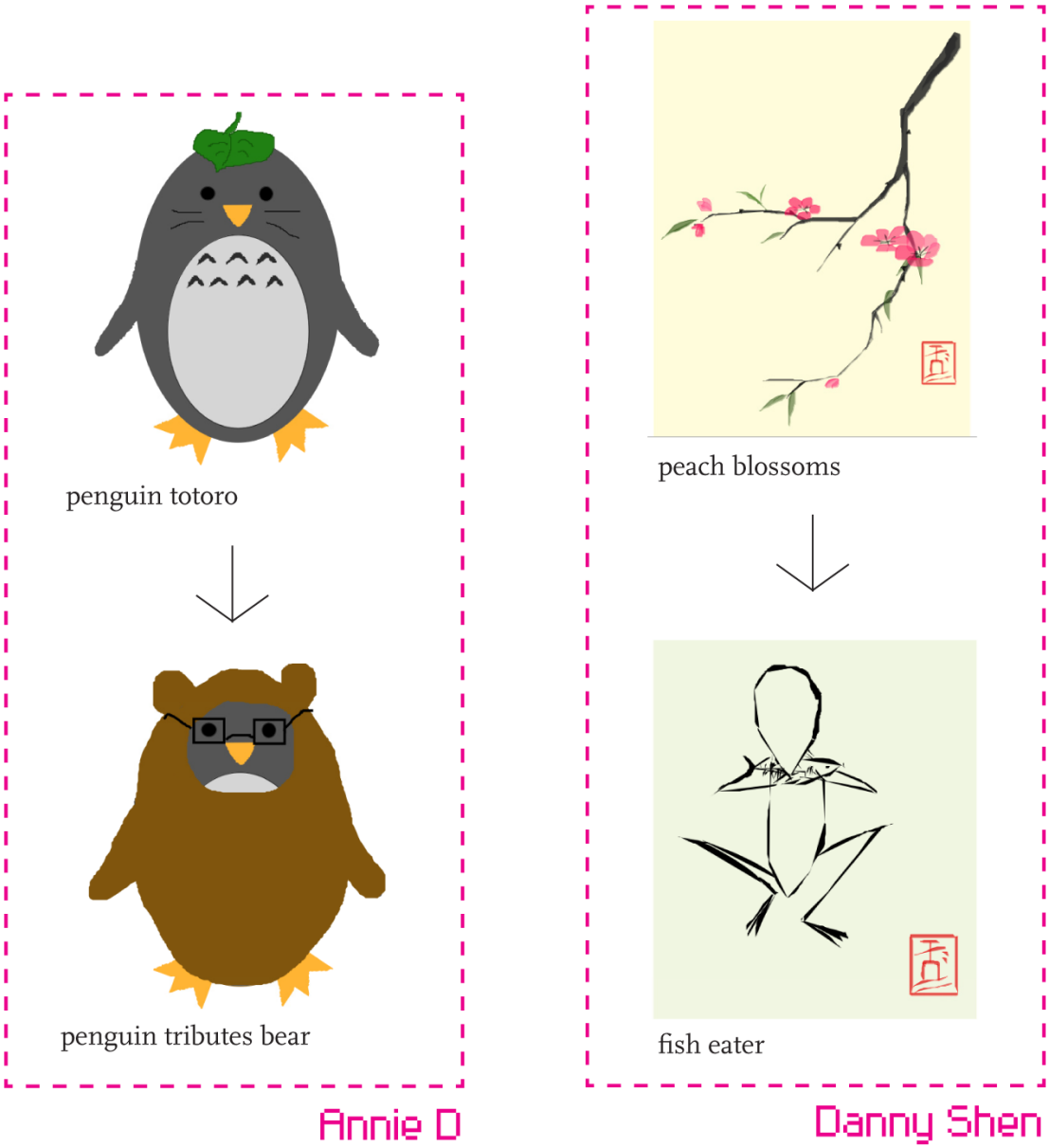

Figure 4-6: Example pieces where parts are reused by the original artist.

Figure 4-7: The Hawaiian flower piece is reused in a stamp-like manner across multiple pieces.

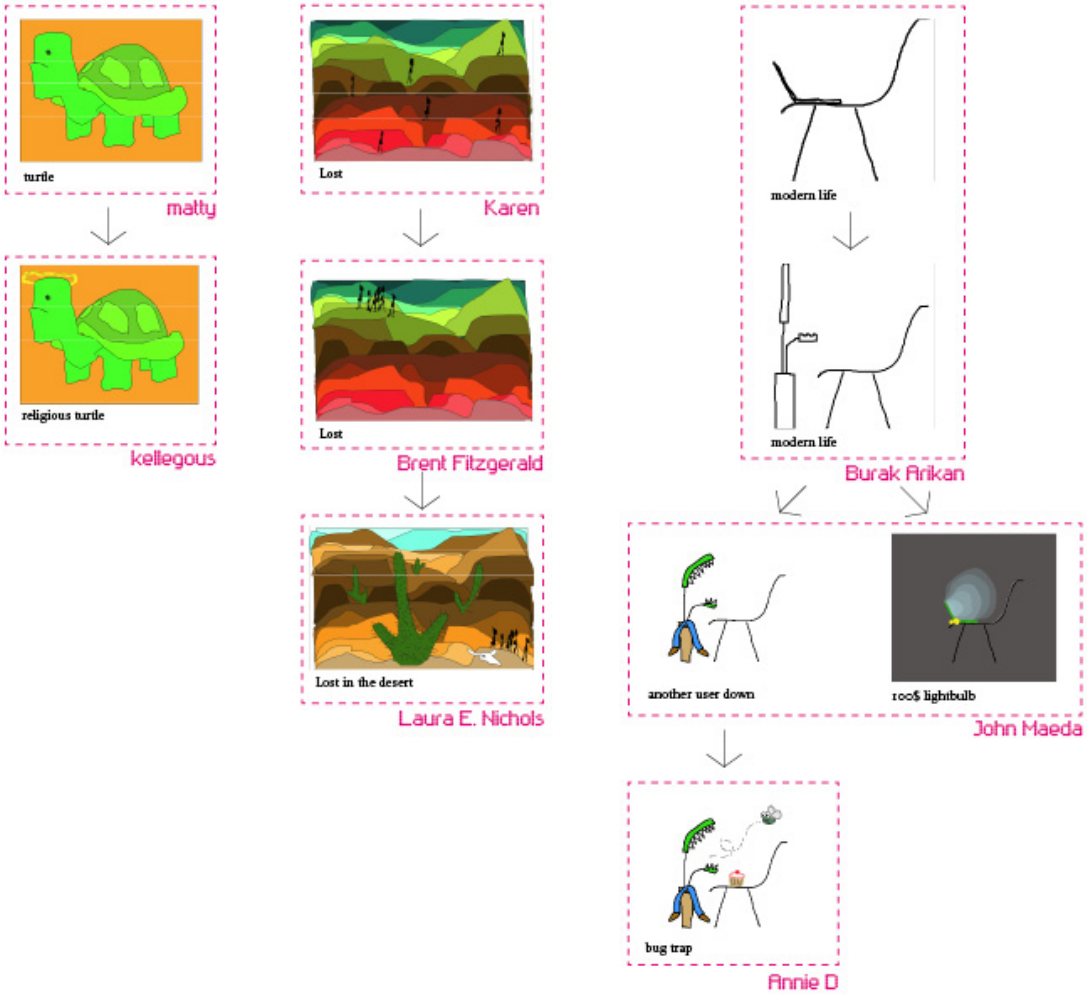

Figure 4-8: Multi-user collaborations through derivative work, ranging from slight additions, to a branching tree of versions.

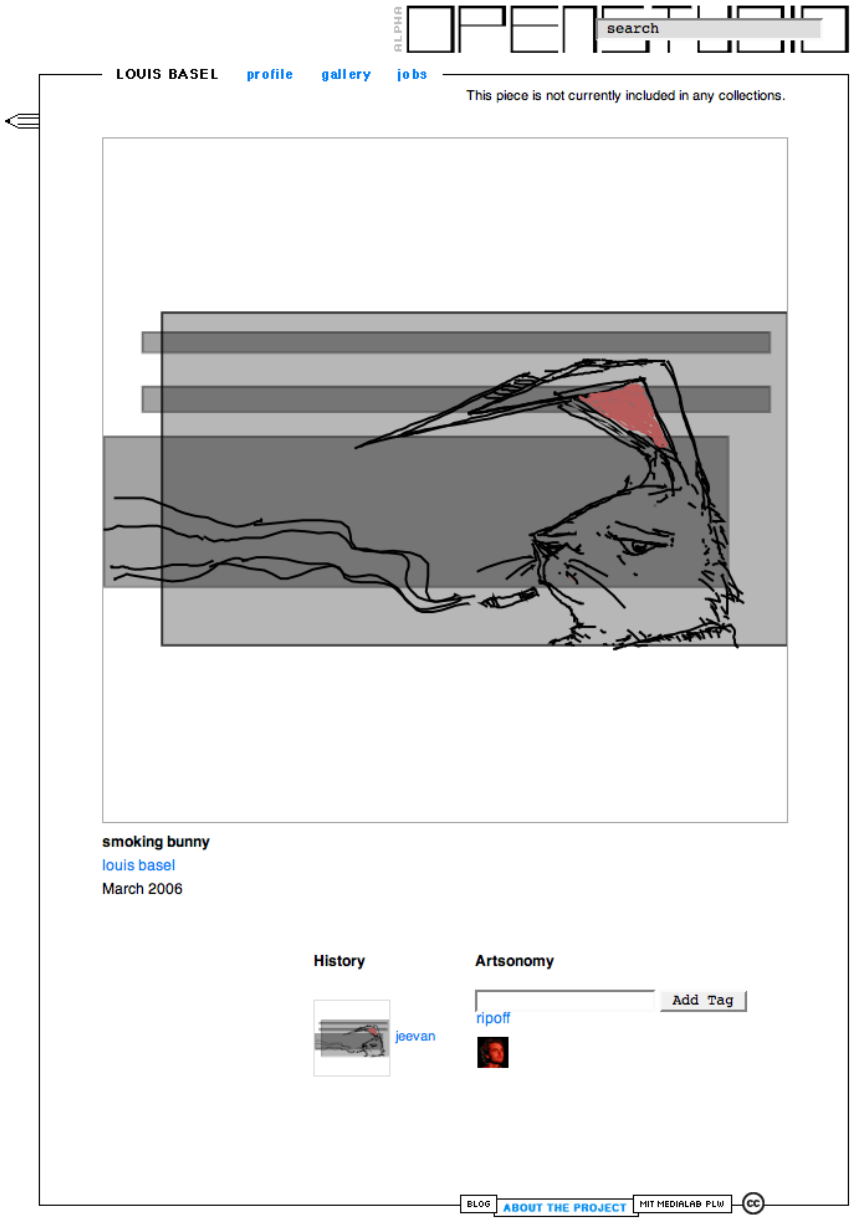

Figure 4-9: Louis basel's exact copy of jeevan's original work can be seen clearly through the History of the piece.

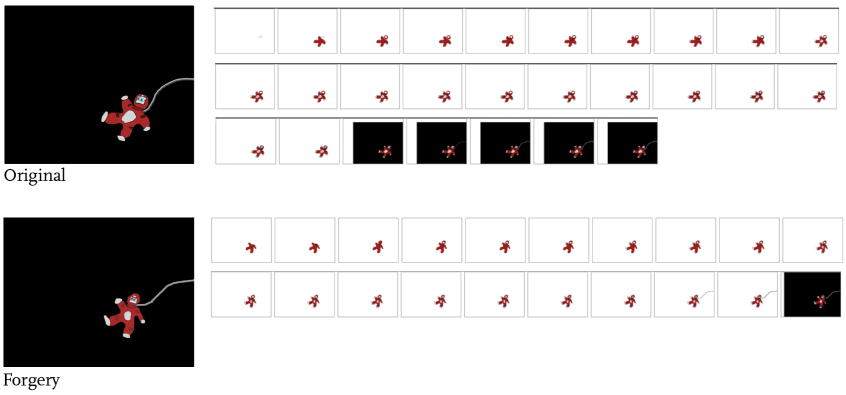

Figure 4-10: The difference between the forgery and original "floating" piece is obvious through looking at the process. The original, by Brent Fitzgerald, shows many more steps, including some mistakes that were later covered up, whereas the forgery by John Maeda is much shorter because it is a deliberate copy.

original, we can view the step-by-step process behind the two. Often, the original contains many extra steps, indicating the work in progress like where the artist changed her mind, or made some mistakes that aren't obvious in the final work. In contrast, the forged work has an almost minimal number of steps to create the likeness of the original. Because the process itself is not creative, there are few reasons to make big mistakes when the final goal is already explicit. This contrast is shown in the two "floating" pieces (Figure 4-10).

4.3.4 Valuation

Because the version genealogy is made open to the viewers of works in OPENSTUDIO, the process also provides a good basis for judging the value of a work. We can analyze this directly from the prices members of OPENSTUDIO are willing to pay for works created in Draw. In general, each derivative work created in the evolutionary chain of a piece of art adds value to a piece because value in the system is correlated with creativity and effort. Often the original work has the greatest share of the value because in artwork originality is highly valued. Likewise, if a person makes a trivial addition in the evolution of a work, there is hardly any value added (Figure 4-8) and if someone makes a particularly creative or clever addition, the value of a piece

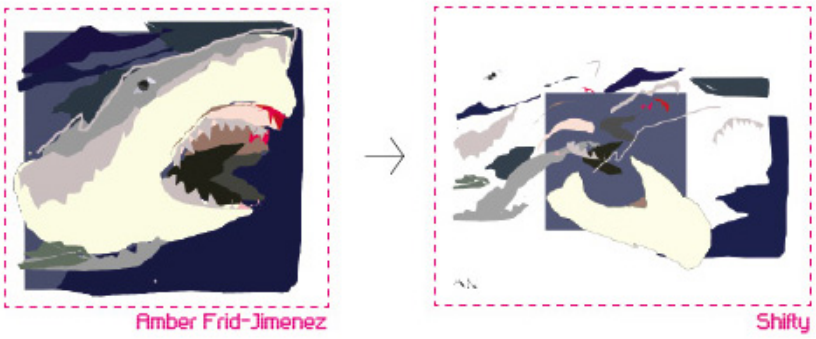

Figure 4-11: Shifty bought the original work,"shark attack" by Amber Frid-Jimenez and then completely changed the work by taking all the shapes in the piece and mixing them into a nonsensical jumble.

increases more.

Also, if a work is merely a direct copy of another work, indicating that no additional effort was put in. The work is not only valued less, but often disrespected by other members of the community and tagged as a "ripoff" (Figure 4-9). For example, Louis Basel in OPENSTUDIO originally made a series of exact copies and tried to sell them for lower prices, but after having all his work marked as ripoffs, he has since taken his copies off from display and sale. OPENSTUDIO member, kellegous, was only able to sell his copies for one tenth of the prices of the originals, and while the originals have appreciated in value, the copies have depreciated.

4.3.5 Expression

Surprisingly, process can even become a medium of expression. One interesting example is the case where Shifty bought "shark attack" from Amber Frid-Jimenez and then completely changed the work by taking all the shapes in the piece and mixing them into a nonsensical jumble as shown in Figure 4-11.

In addition, the preservation of the time dimension of the creative process allows artists to take discrete works and weave them together into something continuous, as in animations. Like Clouzot's capture of Picasso's process, the process itself can become part of a performance. A great example of this kind of purposeful, open

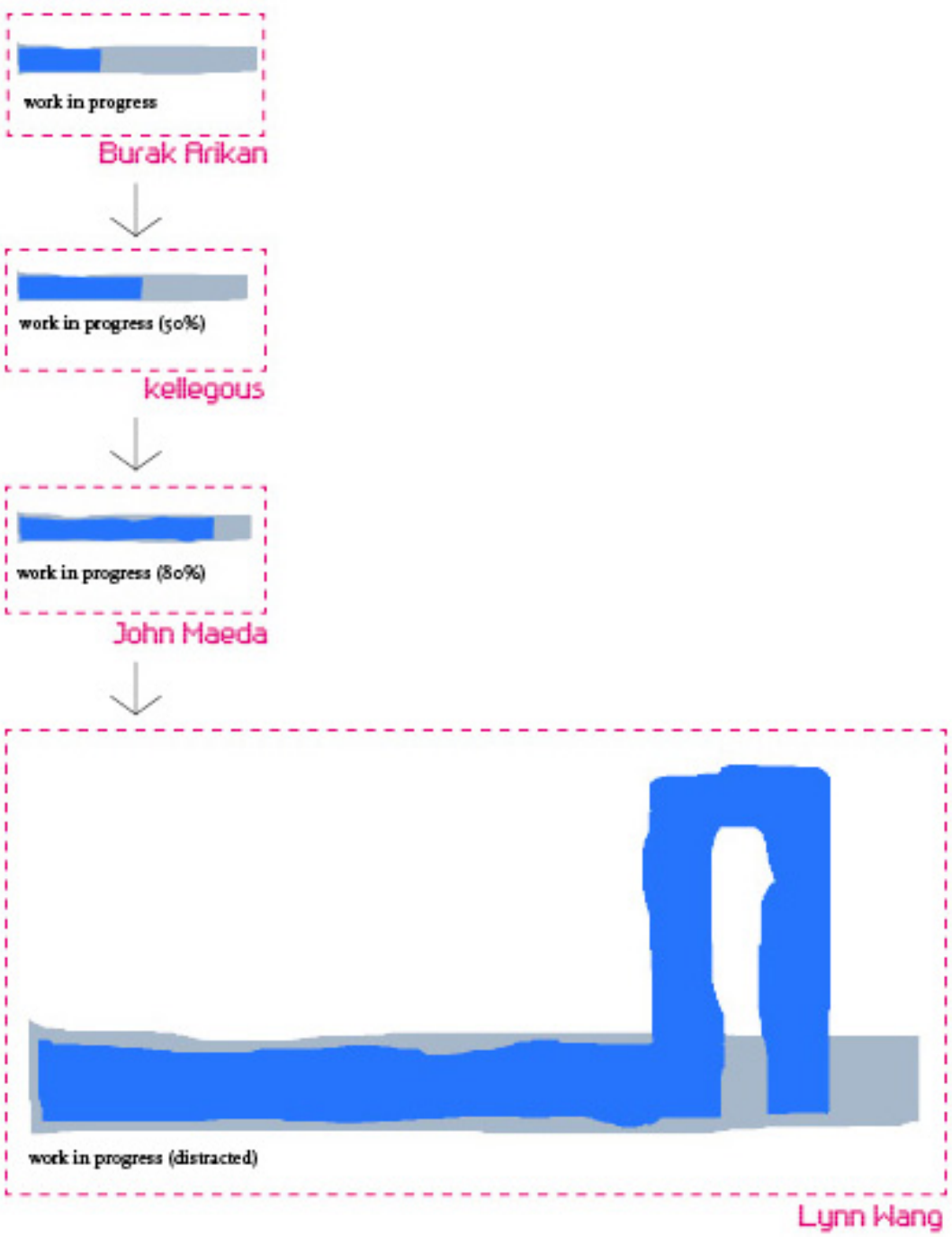

Figure 4-12: This piece, which started as a simple progress bar by Burak Arikan was slowly incremented to 50 percent, and then 80 percent by other artists who bought and extended the work. The most interesting turn occurred when Lynn Wang cleverly created a detour for the progress bar. All of these pieces together, utilized versioning to show a passage of time, creating a storyboard for the work.

collaboration, was "progress" by Burak Arikan. This piece, which started as a simple progress bar was slowly incremented to 50 percent, and then 80 percent by other artists who bought and extended the work. The most interesting turn occurred when Lynn Wang cleverly created a detour for the progress bar (Figure 4-12). All of these pieces together, utilized versioning to show a passage of time, creating a storyboard for the work.

4.4 Data Analysis

With over 3,000 art pieces generated with the use of the Art Genealogy system in OPENSTUDIO, the system provides a rich corpus of art creation process data from which we can extract process regularities in artwork. To do this, I used in intermediate representation aimed at finding correlations of artwork with the original artist or in a more broad sense, an artistic style. This section describes a general analysis of the artwork processes created with Draw, as well as the metrics and results from extracting meaningful regularities in drawing processes.

4.4.1 Fingerprints

Ullman's work [36] has shown the importance of intermediate representations, and in this case, the goal was to find a set of features unique enough so that you could distinguish artwork, but not so detailed that you would not be able find relationships. Because we have so little information about how individuals draw, I chose to take a broad approach, encompassing both visual and temporal information, taking advantage of all the process information. The visual features include characteristics like color, transparency, drawing size, shapes and shape sizes. The time-based features include information like drawing gesture directions, draw frequencies, z-orders, and times for drawings. This way if an artist or some artists have a very unique process behind their work, the regularity can be found.

These features are all grouped together into a "fingerprint". This fingerprint is essentially a feature vector, comprised of 27 features, 11 which are time based and

[92d59718074de7cd0107632954b3007d |
12551798000edbc84356000165a50001 |
210.87357 | 241.91954 | 0.0 |
0.041859377 | 0.42154616 |
0.86206895 | 0.13793103 |
0.047000486 | 0.25128916 |
0.43687177 | 792.35 | 848.69995 |
0.9246939 | 0.6067804 | 0.093333334
| -0.06666667 | 0.39968345 |
0.15578558 | 0.0 | 43.49425 |
2411.4695 | 1257.4122 | 18.787924 |
87.0 | 0.3537007 | 43.0 | 0.0]

Figure 4-13: The "candle" by Danny Shen, and its associated fingerprint of features.

16 which are visual. Figure 4-13 shows an example of the feature vector for the displayed drawing. Two additional pieces of information, a unique id for the drawing and the id of the author, are added to the front of each vector. These allow us to later analyze the fingerprints with feature information only, and then judge accuracy by comparison to the original drawing and author.

I analyzed the distribution of each feature across a snapshot of the OPENSTUDIO database, including 1,860 drawings made by 126 unique artists in OPENSTUDIO. Figure 4-14 shows some example visualizations for drawing shape count and frequency. In each visualization, the x-axis shows the scale for some feature being measured in the pieces. Each row in the visualization is a different artist, and each dot represents one drawing. This visualization makes it apparent that each person has a different distribution, and thus there must be something distinct that can be extracted.

4.4.2 Clustering for Regularity Emergence

There are many ways to approach the problem of extracting regularities from the collection of fingerprints. First, I will describe the results of common clustering techniques including EM and K-means. Then, I will describe how Coen's Cross-Modal Clustering [12] technique applies to the problem. Finally, I will briefly describe the results of some classification techniques I applied to the fingerprints.

Figure 4-14: The distribution visualizations for shapes/drawing and seconds/shape/drawing of the works in OPENSTUDIO. Each row in the visualization is a different artist, and each dot represents one drawing.

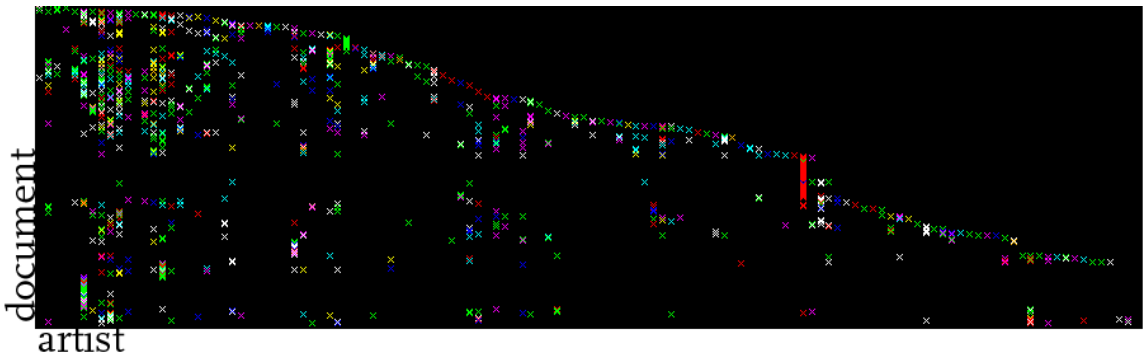

Figure 4-15: The resulting EM plot, generated with the Weka tools, of artists against individual document. The clusters are shown by color, but because Weka is limited to only 10 colors, some colors are repeated.

EM Clustering

Expectation maximization (EM) works well in cases where you have some idea of the data distribution, but you do not necessarily need to know the number of clusters to find. I was able to run this technique using the Weka Library [38]. of machine learning tools. In this case, I set the EM parameters in Weka so as to find a result with approximately 100 clusters. The reasoning behind choosing 100 clusters was that, given 127 artists, I wanted to find a clustering that could roughly correspond with authorship, while still allowing for some overlap.

The image above shows the resulting plot (provided by Weka), of artists against individual document(Figure 4-15). The clusters are shown by color, but Weka is limited to only 10 colors, so some colors are repeated. The purpose for this image is to give a feel for the distributions found. In general, if the work was actually all clustered by artist, we would see one predominant cluster color in each column.

Looking at the actual drawings contained in the clusters, we are able to get a much better sense of why artwork was clustered. Figure 4-16 shows two randomly selected clusters and the resulting images. The dotted lines show groups of work by author. Interestingly, the work seems to be grouped more by style than by just one unique artist. In the top cluster, each of these works were created with a similar process, using a pixel like approach to drawing, where each artist painstakingly placed each

individual rectangle over a long period of time to create an image. The one outlier in this cluster is the text piece, which was not made by rectangles, but was still made using similar stark colors and the long process of making many small polygons. The bottom cluster shows work that, from a visual perspective, it seems to be grouped by the use of amorphous transparent polygons. In this cluster, half of the work was made by one artist.

K-means Clustering

In the case of K-means clustering, it is necessary to provide the number of clusters you want to find. Once again, for the same reasons and for consistency, I used 100 clusters. Figure 4-17 shows the artist to drawing distribution again. At a rough glance, there seems to be even less grouping of artists with their artwork. Upon looking at the actual clusters shown in Figure 4-18, there is still a little grouping by artist. However once again, it seems to be that the work is grouped by a general artistic style, which multiple artists could have. In the top cluster, we can see that the work is grouped very much by the visual feature of using highly saturated colors, and by using the same ellipse tool to a large extent. The bottom cluster shows that you cannot always recognize clusters by only visual information. Although the works share the quality of a large number of polygons, they also share similar time-based information which we cannot extract just by looking at the pictures.

The Shepherd

A particularly interesting cluster that was identified very clearly in both the EM and K-means analysis, was a large cluster of work created solely by an artist in OPENSTUDIO named "The Shepherd". The Shepherd is unique because he solicits artwork of sheep from people online, outside of the OPENSTUDIO environment. Thus, each of the sheep you see above was originally drawn by a different person. However, he then transcribed all the sheep drawings he received into OPENSTUDIO using a bot. This program was able to draw each picture pixel-by-pixel, at a rate far faster than any human could draw. Furthermore, all of his works were drawn at

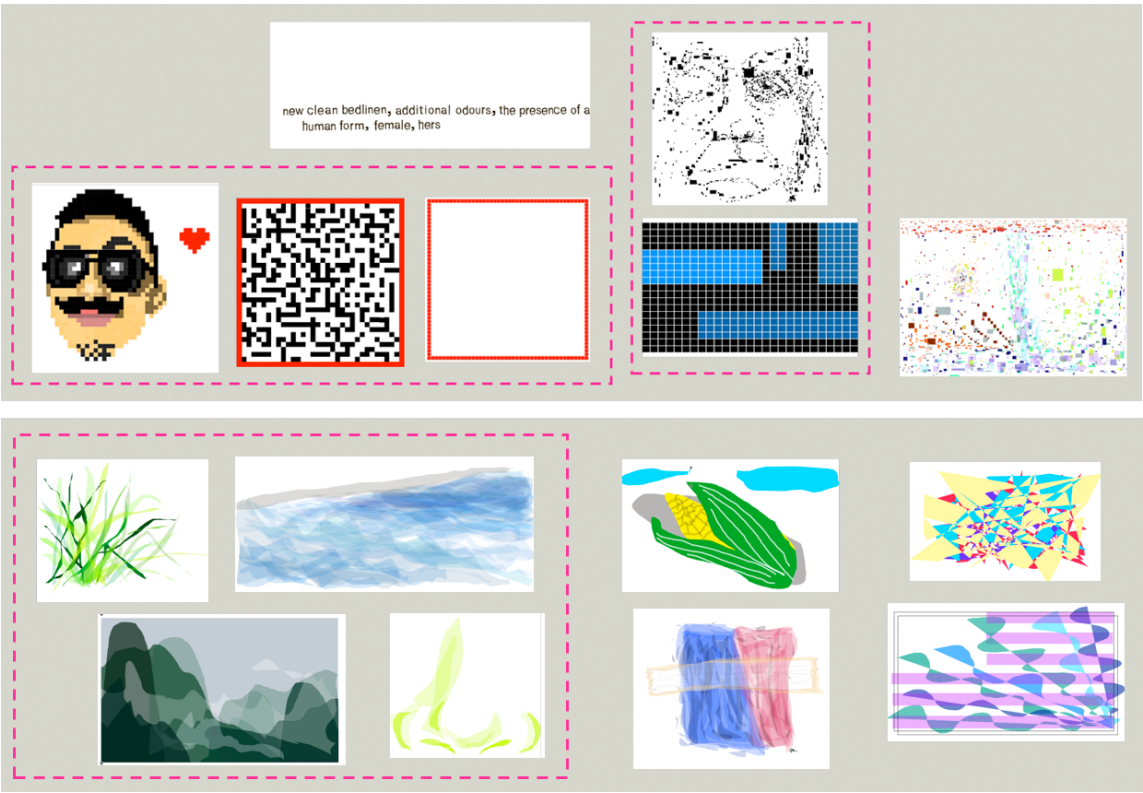

Figure 4-16: Two randomly selected EM clusters. The dotted lines show groups of work by author. Interestingly, the work seems to be grouped more by style than by just one unique artist. In the top cluster (95), each of these works were created with a similar process, using a pixel like approach to drawing, where each artist painstakingly placed each individual rectangle over a long period of time to create an image. The one outlier in this cluster is the text piece, which was not made by rectangles, but was still made using similar stark colors and the long process of making many small polygons. The bottom cluster (28) shows work that, from a visual perspective, it seems to be grouped by the use of amorphous transparent polygons. In this cluster, half of the work was made by one artist.

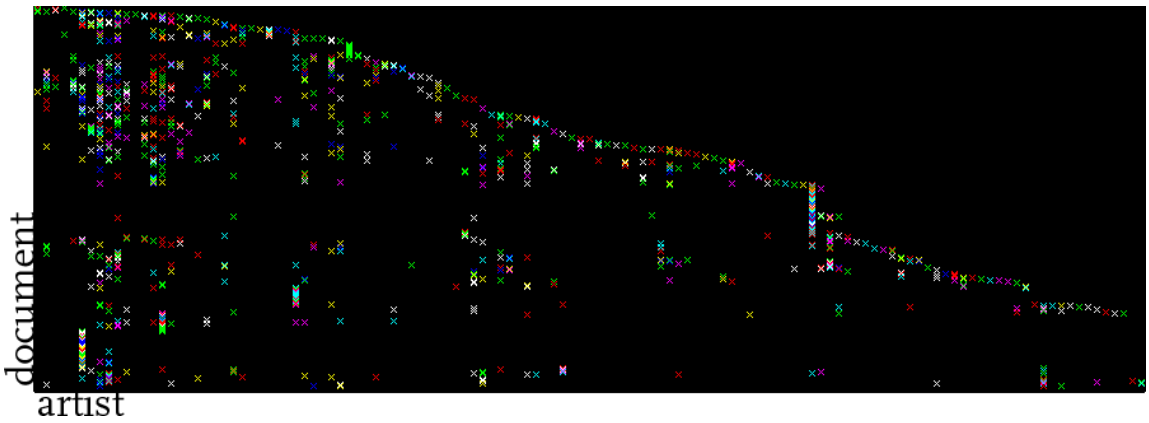

Figure 4-17: The resulting k-means plot, generated with the Weka tools, of artists against individual document. The clusters are shown by color, but because Weka is limited to only 10 colors, some colors are repeated.

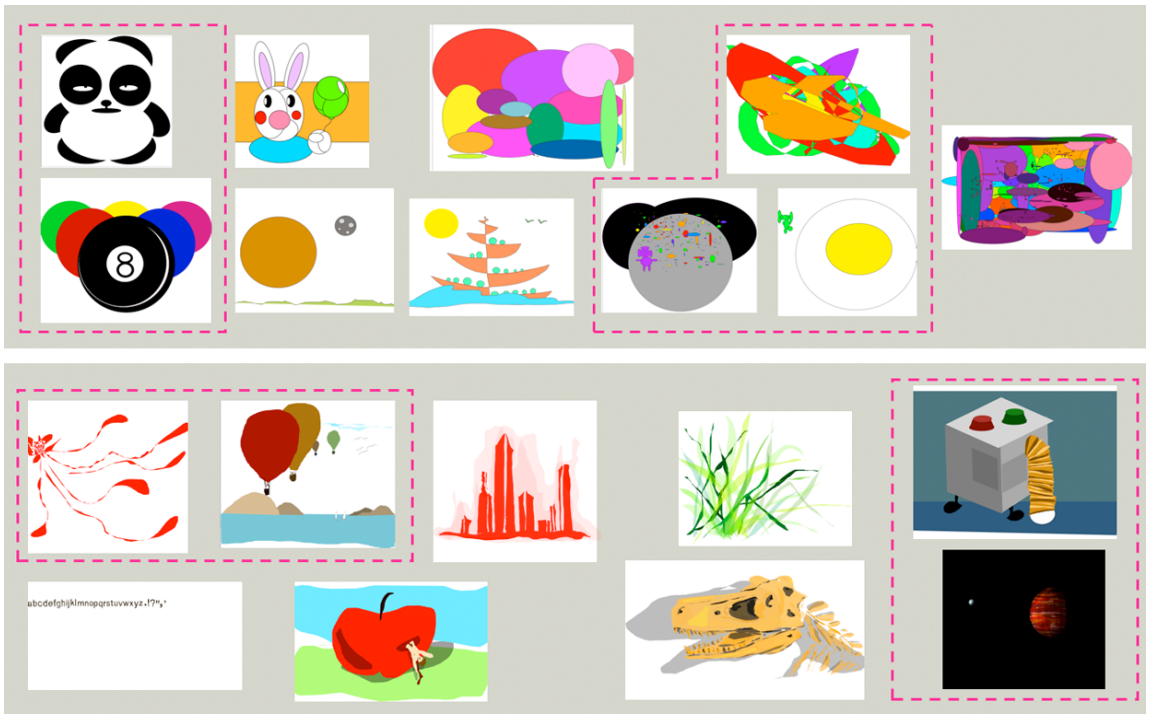

Figure 4-18: Two randomly selected k-means clusters. In the top cluster, we can see that the work is grouped very much by the visual feature of using highly saturated colors, and by using the same ellipse tool to a large extent. The bottom cluster shows that you cannot always recognize clusters by only visual information. Although the works share the quality of a large number of polygons, they also share similar time-based information which we cannot extract just by looking at the pictures.

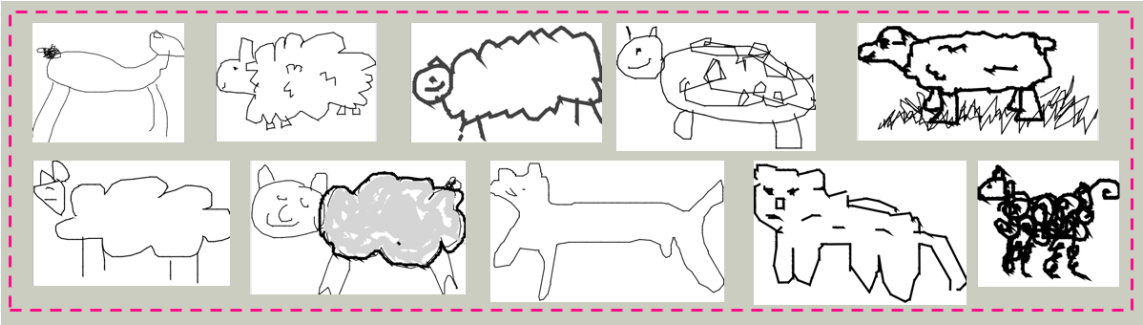

Figure 4-19: The works made by TheSheperd are clustered by both EM and K-means analysis. Because his works are transcribed by a software bot, his process is so unique that all his pieces are clustered together with him as the sole author of works in the cluster.

exactly the same rate because of the program. This incredibly unique process groups all of the work by The Shepherd (Figure 4-19).

Cross-Modal Clustering

A different clustering technique, Cross-Modal Clustering, works well in cases where you have little knowledge about the data you are analyzing. In this case, it seemed like an ideal fit because we have very little understanding about how all the different features in the fingerprint relate to each other. That is, the fingerprint has a diverse set of attributes, all measured in different scales. Although I was not able to implement the Cross-Modal Clustering algorithm due to time constraints, I will describe how the problem can be solved using this technique. The first step of the technique is gathering data on each of the different modes in the form of a Slice. In Coen's phoneme clustering example, he had a visual and audio mode. In the problem of fingerprint clustering, there would be 27 modes, one for each feature. The next step of the algorithm involved comparing all of the modes and seeing probabilistic correlations by activity. In the phoneme example activity was judged by matching times whereas in this example, activity would be judged by matching authorship. Finally, after running the recursive clustering technique, while the phoneme example clustered speech by phoneme, the goal would be to find that the fingerprint analysis would result in clustering artwork by author.

Classification

Additionally, I ran some classification algorithms on the fingerprints using the Weka toolkit [38]. These classifiers included some that were Bayesian, decision trees based, formula based, and rule based. The top performing classifiers in classifying each work to its respective author using cross validation were NNge and K*. NNge is a nearest neighbor based algorithm and had 40 percent accuracy while K* is an instance based algorithm that had 38 percent accuracy in matching authorship. These results seem very encouraging given the 1,860 drawings and 126 authors analyzed.

Forgeries

Finally, one interesting application of clustering and classifying fingerprints is the ability to understand and identify forgeries. The images in Figure 4-10 show one case of forgery in OPENSTUDIO. The images look almost identical and it is difficult, just from a visual standpoint, to figure out which is which.

However, upon looking at the process information in the fingerprints, the forgery becomes obvious. In general the original took a much longer time, with a much lower draw frequency because the artist was still planning as he drew and had to correct mistakes. In contrast the forgery was done very fast because the artist was able to deliberately copy. Furthermore, classification techniques can be used to show who the author of each was because we can compare the fingerprints and see which artist the work gets classified with.

Chapter 5

Future Work

The systems in this thesis present a model for collective process capture that can be taken to the next level when combined with other methods of knowledge capture or applied to other domains of media creation. The user analysis shows that process capture and review, which is often overlooked nowadays, has great potential to influence and aid design work.

5.1 AI

One of the most promising areas for further exploration is in the field of artificial intelligence. With the vast collection of process metadata that both systems provide, along with the capability to annotate or tag the process, the data can be used with machine learning to infer deeper rationale and semantic information. Additional analysis methods can be used to extract regularities in processes, recognizing when a sequence is common-place or unique for a particular type of drawing or a specific author. Work can be done in combining raw process sequences into meta-sequences with semantic meaning, such as grouping the individual processes for two eyes, a nose, and a mouth into a face process.

Furthermore, the ability to draw semantic meaning from processes also opens the door to process recognition. Currently, we are familiar with the idea of image recognition, where the source data is often a two dimensional bitmap or even possibly

a three dimensional model. Process recognition adds the additional dimension of time, for even richer information. Thus, a person may be able to recognize a drawing of a face, not only by the final image it creates, but by the sequence of gestures used by an artist to create it.

5.2 UI

Additional work must also be done in the user interface domain. Process capture and review can be immensely complex from an engineering perspective, but the user must still be shielded from this complexity. As discussed, the UI must be as unobtrusive as possible while still remaining helpful so that people will willingly contribute to the design knowledge repository. Thus, more work needs to be done in finding ways to implicitly solicit user data. Additionally, visualizing the branching processes along the time dimension is still an unsolved problem. We must find ways of being able to view this non-linear information in an easily browsed way. Visualizations can also be used for viewing regularities between processes in process comparison.

5.3 Open API

In order to allow more analysis of the data collected in OPENSTUDIO through Art Genealogy, an open api must be developed so that others can benefit from the massive amounts of creation data we have collected. This would create a rich data resource for AI analyses as well as visualization analyses.

5.4 Digital Rights

The Art Genealogy system also has many implications in authorship and attribution. Current licensing and copyright models, especially in the realm of digital media, only account for authorship at one point in time and give all credit to the original authors. However, with the influx of media and information made available via the

internet, more and more derivative works are being formed. In such scenarios, the balance of authorship between the original author and successive authors is unknown. Original authors may feel violated when someone creates a derivative work. Similarly, successive authors may not get deserved credit for making innovative extensions. This is clearly demonstrated in the music domain where music is frequently remixed. For example, in the case of *The Grey Album* by Danger Mouse, he remixed The Beatles' *White Album*, with Jay-Z's *Black Album*, but could not legally release the CD because of copyright infringement. Art Genealogy's approach would allow derivative works to be sold, but point attribution to other authors that contributed to the work at different times.

5.5 Social benefits

Finally, we have only touched the tip of the iceberg in terms of recognizing the applications for process information. Prior to this work, it was widely agreed upon that process capture aided in learning skills through emulation as well as information reuse. However, the short period in which Art Genealogy has given users open access to the creation process has shown the impact of process capture on encouraging artistic reuse of work, valuation, expression, and attribution. We must continue to analyze the effects of shared design process information, whether it be by numeric analysis, user tests, or social surveys, to best understand how people can benefit from it. As users are able to access more process information in more effective ways in highly interactive communities, the list of process capture benefits will continue to grow.

Chapter 6

Contributions

In summary, this thesis has made the following contributions:

- Identified five major applications of design process information: learning, reuse, attribution, evaluation, and expression.

- Explained the importance and challenges of using representations that allow for combinatorial process building, making process and rationale capture systems easy to use, and sharing process information in order to obtain all five applications in a design capture system.

- Explained the benefits of capturing and preserving design processes and rationale in the domain of digital art creation.

- Enumerated the six major goals for design process capture systems: domain independence, combinatorial time manipulation, process search and comparison, multiple user support, embedding design rationale, and user friendliness.

- Designed and implemented two art process capture systems, Chronicler and Art Genealogy, which capture, preserve, and share art processes in collective environments in an easily reviewable and reusable way. Today, the Art Genealogy system is in use by over 250 online users and has collected over 5,000 drawing processes.

- Implemented a simple raster art painting and a simple vector drawing tool for capturing art processes.

- Created the Chronicler *action sequence* representation, a relatively fine grained intermediate process representation that was generalizable to different domains and multiple users. Action sequences are made of atomic steps, providing combinatorial use of steps and easy comparison.

- Created the double layer process representation used in Art Genealogy, of using document *versions* as a coarse grained representation and embedding *timestamp acts* in the drawing representation of each version to capture more detailed process information. This representation was a simple but powerful extension of the Draw application in OPENSTUDIO allowing combinatorial use and collaboration.

- Created the process review user interfaces for Chronicler and Art Genealogy, that allowed users to interact with simple visualizations of their art processes while designing and seamlessly captured major key points of the process.

- Compared and contrasted the strengths and weaknesses of Chronicler and Art Genealogy across the representation, usability, and collective sharing axes.

- Conducted user studies for both systems.

- Presented the many interesting emergent behaviors in OPENSTUDIO that have resulted from the Art Genealogy and have illustrated the five major process applications.

- Implemented a process feature analysis system, visualizing feature distributions.

- Created the *fingerprint* representation for showing regularities in features across the processes captured in Art Genealogy.

- Clustered drawings using the fingerprints using EM and K-means, showing the correlations of cluster to artistic styles and authors.

- Showed how fingerprint clustering and classification can be applied towards forgery identification.
- Presented future applications and extensions of this work.

This thesis has provided systems and models for capturing, reviewing, and sharing the design process and design rationale. As evidenced through the active use of process knowledge resulting from these systems, it is apparent that as more people preserve and share processes, we will continue to discover additional applications of this often untapped resource.

Bibliography

[1] 37signals. Ruby on rails screen casts. www.rubyonrails.org/screencasts.

[2] Adobe. www.adobe.com/products/creativesuite/versioncue.html.

[3] Adobe. Photoshop. www.adobe.com/products/photoshop/.

[4] Guillermo Arango, Eric Schoen, and Robert Pettengill. A process for consolidating and reusing design knowledge. In *ICSE '93: Proceedings of the 15th international conference on Software Engineering*, pages 231–242, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[5] Burak Arikan, Annie Ding, Brent Fitzgerald, Amber Frid-Jimenez, and Kelly Norton. Openstudio. openstudio.media.mit.edu.

[6] Autodesk. Maya mentor. www.alias.com/eng/community/tutorials/mayamentor/index.jhtml.

[7] Oleg Bartunov and Teodor Sigaev. Tsearch2: Full text extension for postgresql. www.sai.msu.su/ megera/postgres/gist/tsearch/V2/.

[8] Thomas Berlage and Andreas Genau. A framework for shared applications with a replicated architecture. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 249–257, New York, NY, USA, 1993. ACM Press.

[9] B. Berliner. CVS II: Parallelizing software development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 341–352, Berkeley, CA, 1990. USENIX Association.

[10] John M. Carroll, Sherman R. Alpert, John Karat, Mary Van Deusen, and Mary Beth Rosson. Raison d'etre: capturing design history and rationale in multimedia narratives. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 192–197, New York, NY, USA, 1994. ACM Press.

[11] Henri-Georges Clouzot. The mystery of picasso, 1956.

[12] Michael H. Coen. *Multimodal Dynamics: Self-Supervised Learning in Perceptual and Motor Systems*. PhD thesis, Massachusetts Institute of Technology, 2006.

[13] Creative Commons. Creative commons. creativecommons.org.

[14] Annie Ding, Katherine Hollenbach, and Kelly Norton. Save your design, 2005. MIT User Interface Design Final Project.

[15] W. Keith Edwards, Takeo Igarashi, Anthony LaMarca, and Elizabeth D. Mynatt. A temporal model for multi-level undo and redo. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 31–40, New York, NY, USA, 2000. ACM Press.

[16] PostgreSQL Global Development Group. Postgresql. www.postgresql.org/.

[17] T. R. Gruber and D. M. Russell. Generative design rationale: beyond the record and replay paradigm. In Thomas Moran and John H. Carroll, editors, *Design rationale: concepts, techniques and use*. Lawrence Erlbaum Associates, 1995.

[18] Ambrosia Software Inc. Snapz pro. www.ambrosiasw.com/utilities/snapzprox/.

[19] JBoss Inc. Hibernate. www.hibernate.org/.

[20] Sun Microsystems Inc. Jdbc. java.sun.com/products/jdbc/.

[21] Yahoo! Inc. Flickr. www.flickr.com.

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

[23] Satish Kumar. Wink. www.debugmode.com/wink/.

[24] SQUID labs. Instructables: Step-by-step collaboration. www.instructables.com.

[25] B. Leuf and W. Cunningham. *The Wiki Way: Collaboration and Sharing on the Internet*. Addison-Wesley, 2001.

[26] Scott Arthur Moody. The stars process engine: language and architecture to support process capture and multi-user execution. In *TRI-Ada '94: Proceedings of the conference on TRI-Ada '94*, pages 4–15, New York, NY, USA, 1994. ACM Press.

[27] B. Myers and D. Kosbie. Reusable hierarchical command objects. In *Proceedings of CHI '96*, 1996.

[28] Karen L. Myers, Nina B. Zumel, and Pablo Garcia. Automated capture of rationale for the detailed design process. In *AAAI/IAAI*, pages 876–883, 1999.

[29] Kumiyo Nakakoji, Yasuhiro Yamamoto, and Masao Ohira. A framework that supports collective creativity in design using visual images. In *C&C '99: Proceedings of the 3rd conference on Creativity & cognition*, pages 166–173, New York, NY, USA, 1999. ACM Press.

[30] C. Michael Pilato, Ben Collins-Sussman, and Brian W. Fitzpatrick. *Version Control with Subversion*. O'Reilly, 2004. http://svnbook.red-bean.com/.

[31] Carlos Rocha. Smpl: A network architecture for collaborative distributed services. Master's thesis, Massachusetts Institute of Technology, 2005.

[32] Marc Schwartz. Process makes perfect. Master's thesis, Massachusetts Institute of Technology, 2005.

[33] S. Shum. Cognitive dimensions of design rationale. Technical Report EPC-91-114, 1991.

[34] P. Singh. The public acquisition of commonsense knowledge, 2001.

[35] ThinkCycle. Thinkcycle: Open collaborative design. www.thinkcycle.org.

[36] Shimon Ullman. *High-level Vision*. MIT Press, 1996.

[37] Inc. Wikimedia Foundataion. Wikipedia. www.wikipedia.org.

[38] Ian H. Witten and Eibe Frank. Data mining: Practical machine learning tools and techniques, 2005. www.cs.waikato.ac.nz/ml/weka/.

[39] Xstream. xstream.codehaus.org.