PILPUL: A 2D Graphical Virtual Forum for Exploring Identity and Values

by

Aaron Arakawa

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

May 21, 1999

© Copyright 1999 Aaron Arakawa. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

Author: ____________________________

Department of Electrical Engineering and Computer Science. May 15, 1999

Certified by: ________________________

Michel Resnick, Thesis Supervisor

Accepted by: ________________________

Arthur C. Smith, Chairman, Department Committee on Graduate Theses
PILPUL: A 2D Graphical Virtual Forum for Exploring Identity and Values
by
Aaron Arakawa

Submitted to the Department of Electrical Engineering and Computer Science
May 21, 1999
In Partial Fulfillment of the Requirements for the Degrees of Bachelor of Science in
Computer Science and Master of Engineering in Computer Science at the Massachusetts
Institute of Technology

Abstract

Pilpul is an attempt at using the computer as a collaborative on-line learning tool for
exploring identity and values. Users may walk from room to room in Pilpul, talking to
other users, creating artifacts and role models they admire. In addition, users may write
stories and biographies about the role models they create, and stories and descriptions
about the artifacts they create. This thesis will discuss the ideas behind Pilpul, the
technical challenges, and the implementation details. It reflects the work of Marina Bers
of the Epistemology and Learning Group at the M.I.T. Media Laboratory on the concepts
behind Pilpul and the technical accomplishments in attempting to actualize Bers’ vision.

Thesis supervisor: Mitchel Resnick
Title: Professor of the Epistemology and Learning Group, M.I.T. Media Laboratory
# Table of Contents

Abstract ........................................................................................................................................... 2  
Table of Contents .......................................................................................................................... 3  
1. Introduction .................................................................................................................................. 4  
2. Background .................................................................................................................................. 5  
   2.1 Theory Behind Pilpul .................................................................................................................. 5  
      2.1.1 Constructionism .................................................................................................................. 6  
      2.1.2 Multi-User Domains (MUD’s) ............................................................................................... 6  
   2.2 Related work ............................................................................................................................. 7  
      2.2.1 Logo .................................................................................................................................... 7  
      2.2.2 Sage .................................................................................................................................. 8  
      2.2.3 Kaleidostories ...................................................................................................................... 8  
      2.2.4 Pet Park ............................................................................................................................... 9  
3. Vision of Pilpul ............................................................................................................................... 11  
   3.1 A Sample Scenario: .................................................................................................................. 11  
   3.2 Borrowed Features .................................................................................................................... 14  
      3.2.1 Navigation .......................................................................................................................... 14  
      3.2.2 Interaction with Characters and Objects .............................................................................. 15  
      3.2.3 Storytelling in Two Forms: Narrative and Interactive ....................................................... 15  
      3.2.4 Collaborative Biographies for all Avatars .......................................................................... 15  
   3.3 New Concepts ........................................................................................................................... 16  
      3.3.1 Customizing graphics ......................................................................................................... 16  
      3.3.2 E-mail system: ................................................................................................................... 17  
      3.3.3 Conversational Programming ............................................................................................. 17  
4. Functional Goals of Pilpul .............................................................................................................. 17  
   4.1 The World ................................................................................................................................ 17  
   4.2 Graphical Representation Capability ....................................................................................... 18  
   4.3 Supporting Multiple Users ....................................................................................................... 18  
   4.4 Homerooms .............................................................................................................................. 18  
   4.5 Interaction Between Users ........................................................................................................ 18  
   4.6 Interaction Between Users and Objects ................................................................................... 18  
   4.7 Object Creation, Editing and Copying ....................................................................................... 19  
5. Choosing a Platform for Development ......................................................................................... 19  
   5.1 Objectives of the Platform of Implementation ........................................................................... 19  
   5.2 Options Considered ................................................................................................................... 20  
      5.2.1 Intel 1DMOO SDK ................................................................................................................ 20  
      5.2.2 Java Socket Servers ............................................................................................................ 20  
      5.2.3 The Chosen Platform: Java RMI .......................................................................................... 21  
   5.3 Achieving Object Persistence .................................................................................................... 21  
      5.3.1 Database vs. File System ................................................................................................... 22  
6. Implementation ............................................................................................................................... 22  
   6.1 Data Structures ......................................................................................................................... 22  
   6.2 The Server ................................................................................................................................ 25  
      6.2.1 The Server Object ............................................................................................................... 25  
      6.2.2 Initialization of the Server ................................................................................................... 26  
      6.2.3 World Persistence ................................................................................................................. 26  
      6.2.4 Changes to Objects .............................................................................................................. 26  
      6.2.5 Providing Clients with Information ...................................................................................... 27  
      6.2.6 Database ............................................................................................................................. 27  
      6.2.7 Re-creation of the World ..................................................................................................... 28
1. Introduction
The vision of Pilpul is an interactive, graphical multi-user domain for children to explore values and identity through collaboration and programming. Users can walk from room to room in Pilpul, talking to other users, creating artifacts and role models they admire. In addition, Pilpul would be populated with facsimiles of role models programmed to respond to situations in a way appropriate for the role model. Children may also construct their own rooms, create their own objects and role models, and program these with descriptions and behaviors using a conversational scripting language. The conversational programming aspect of Pilpul has a two-fold function, both to foster logical thinking and to help children explore their value and identity by forcing them to think about their role models' situational behavior. Children may also collaborate in writing biographies for the role model characters on the system, and communicate with each other on-line. This collaborative aspect of Pilpul is designed to create a “snowball” effect where ideas are built up collaboratively rather than individually.

The Pilpul project began as a combination of the collaborative role model exploration of Kaleidostories (Bers, 1998) and the 2D virtual environment of Pet Park (DeBonte, 1998). Pilpul is the idea of Marina Bers of the M.I.T. Media Laboratory. Originally, the Intel IDMOO 2D software development kit that powered Pet Park was to be the platform for
Pilpul. However, after three months trying unsuccessfully to get the abandoned and unsupported Intel project to work, this plan was scrapped. In struggling with the IDMOO code, however, enough knowledge was gained to construct a simpler 2D multi-user platform from scratch.

A good portion of the vision of Pilpul was implemented. From a technical standpoint, the major technological challenge of creating a multi-client interactive platform was met and conquered. The efforts of Grace Bae and myself resulted in a working client-server application accessible through an ordinary Netscape web browser. Users were able to explore the rooms and create and interact with objects, a great success from a software development perspective. Some of the conceptual features were left for future work on the project. These included conversational programming and collaborative biographies for role models.

The thesis will begin with discussion of the theory behind Pilpul and related software projects that influenced it. The next section will present the Pilpul concept in its entirety. Then the platform of development and the actual implementation details will be discussed, followed by a synopsis of technical accomplishments and future work to be done.

2. Background

This section details the theory and previous work that contributed to the Pilpul vision. Pilpul is the vision of Marina Bers of the Epistemology and Learning Group of the MIT Media Laboratory. The first subsection describes constructionism and Multi User Domains, and the second cites previous works involving one or both of these concepts.

2.1 Theory Behind Pilpul

This subsection divides the theory behind Pilpul into educational and technical aspects. Constructionism is the educational theory underlying Pilpul and the Multi-User Domain (MUD) is the technical basis for the implementation. These two concepts are described in this section.
2.1.1 Constructionism

The vision of Pilpul is an environment where children can explore identity and values by building characters and objects that embody ideals and values (or lack thereof) and create distinct behaviors for them. This is what educational theorists may call a constructionist learning environment, where the learner is able to explore and proactively make discoveries about a particular subject, whether it be geometry, group behavior, or in this case values and identity. Constructionism is a theory of education that departs from the traditional techniques of teaching by drills and repetition (Papert 1980). The constructionist approach to education “consists of providing opportunities for children to engage in creative activities that fuel the constructive process.” The premise of constructionism is that learning is best done by students constructing knowledge based on their own experimentation and experiences, rather than teachers instructing them by rote learning. Constructionism correlates constructing objects in the real world with constructing knowledge inside the learner’s heads. Pilpul aims to be a constructionist material that allows children to simultaneously construct values and culture in the virtual world as well as in their own heads. Values and identity are important issues to pre-teens and teenagers, as they become self-aware and must learn to act autonomously and responsibly (Covey, 1993). However, Bers feels they are not emphasized enough in schools. Pilpul is a virtual forum where these concepts can be explored. In particular, Pilpul focuses on programming behavior into virtual characters as a constructionist tool to make the user’s own behavioral motives more tangible and subject to reflection. Some of the previous constructionist materials include Logo, Sage, and Kaleidostories.

2.1.2 Multi-User Domains (MUD’s)

A Multi-User Domain (MUD, also known as distributed virtual environment) is a virtual world on the Internet that users may enter and interact with other users in real time (Braham, Comerford 1997). The virtual space of a MUD is represented by interconnected rooms analogous to sectioned off regions of physical space. Text descriptions or 2D or 3D graphics represent the environment surrounding the user, which includes their
physical surroundings and other users, computer controlled characters, and inanimate objects in the virtual world. Every user is also represented on the screen by a graphical or text facsimile of a person called an avatar. Users interact through their avatars with objects, people and other characters in the same room in real time by inputting commands to control their avatars from the keyboard, mouse, or other input devices.

2.2 Related work
This section will discuss other pieces of software whose ideas or concepts have been borrowed or expanded on in the creation of the Pilpul vision: Logo, Sage, Kaleidostories and Pet Park. Logo is discussed as the classic example of constructionist software. Sage is an interactive storytelling environment, and the goal of Pilpul is to incorporate this conversational storytelling aspect. Kaleidostories is an on-line text-based forum for exploring identity, values, and role models through storytelling. The concept for Pilpul builds on Kaleidostories, adding the dimension of interaction between users and role model characters. Pilpul would like to facilitate these interactions through displaying graphics and animation and gathering user input on any number of remote terminals. Pet Park is an earlier work done at the Media Lab and provided a conceptual basis for the design of the Pilpul interface.

2.2.1 Logo
Logo is an educational programming language where the children can move a graphical “turtle” around a two dimensional graphical world by giving it geometric instructions like “turn right 35 degrees” and “move forward 30 steps.” The language also lets the learner explore algorithmic reasoning through procedure, sequences of instructions that can be strung together to control the turtle. The act of writing down a sequence in procedure form allows the child tangible access to his thought process which can be reflected on and improved. This facilitating of reflection and improvement in reasoning about planar geometry was adapted into the Pilpul concept by Bers to apply to conversation, storytelling and behavior.
2.2.2 Sage

Sage was Bers’ first experiment with storytelling as a constructionist activity. Since then Bers’ Kaleidostories and now Pilpul built upon this theme. The interactive storyteller Sage provided a means for self-reflection through construction of interactive storytellers (Bers & Cassell 1997). The children would interact with storytellers built by others then build on their experience by constructing their own wise storyteller. They created new storytellers by building a database of stories and programming in a framework that simulated a conversational flow of interaction through turn taking states and communicative actions in a conversation. This framework forced children to think about what situations certain acts of conversation were appropriate in. This helped them think about their own social relations in the world. In addition, the end product contained clues to the children’s own culture and interests, in the stories and behaviors they chose to incorporate into their storyteller.

The premise behind Sage was that our personal values, experiences and culture are all embodied in the stories we tell. Stories are a primitive form of communication from which the listener may extrapolate ideas and feelings of personal relevance.

Sage is one example of a constructionist tool. Constructionism asserts that learners are likely to have access to more and different kinds of new ideas when they are building things that they can reflect upon and share with others in their learning community. In Sage, children shared their creations with other children.

2.2.3 Kaleidostories

Another of Bers’ projects, Kaleidostories, also used storytelling as a constructionist activity (Figure 1). Kaleidostories, however, did not have a conversational model of storytelling. Rather, it was a one-dimensional model where children would tell stories about their role models. The goal of Kaleidostories was to help children explore values and identity through the constructionist activity of building a web page featuring their favorite role models and stories about them. After entering each story about a particular role model of theirs, the user would be presented with a list of values to choose from that
the story embodied. The user was prompted to tell one or more stories about actions their role model took that the user admired. In connecting a story to a value, the user made a connection between the abstract value and the concrete action. This has implications on how the child will categorize future real life situations and the actions they will choose. Bers contends that as long as role models are in a context removed from every day experiences, they remain “empty vessels, with no invested meaning from the part of the child” (Bers, 1998). NBA coach Rick Pitino put it, “a man who buys a Brioni suit because he wants to be like Pierce Brosnan won’t be Pierce Brosnan, he’ll be the same guy in a better suit” (Pitino, 1996).

![A Kaleidostories screen shot. Users may build biographies of role models collaboratively.](image)

**Figure 1:** A Kaleidostories screen shot. Users may build biographies of role models collaboratively.

### 2.2.4 Pet Park

Much of the Pilpul concept was derived from Pet Park. Pet Park was a 2D graphical, constructionist medium for programming. Pilpul departed from this concept in that it
involves construction of identity and values while Pet Park constructed dances and animated behaviors. However, the desired look and feel of Pilpul was largely formed from Pet Park’s graphical interface.

Pet Park is a graphical, virtual MUD (Multi-User Domain) populated by pets, created by Austina DeBonte (DeBonte, 1997). DeBonte’s work continued experimentation with MUD’s as a constructionist medium in the MOOSE Crossing text based MUD project (Bruckman, 1996). In Pet Park, the pets are programmable using Yo Yo, a programming language based on Java and Logo (Begel, 1997).

Pet Park uses programming as a constructionist learning activity. In Pet Park, children create pets and program them with scripts to control their animated behaviors. DeBonte asserts that programming as a constructionist activity helps children isolate problems and reflect and improve on thought processes about algorithms and procedures.

The Pet Park world consisted of many interconnected rooms. The interface of Pet Park at any given time displayed the current room graphically, as well as pets and objects occupying the current room. Properties of rooms and objects were accessible through mouse and keyboard input, prompting pop-up windows and dialog boxes to appear for the user to alter the pets’ properties. These properties included name, description and scripts controlling behavior. Users could chat with one another using a small text window at the bottom of the screen. This interface would become the model for the Pilpul interface design. (Figure 2)
3. Vision of Pilpul

This section sketches out the vision of Pilpul the way it was in the minds of the creator, Marina Bers, with contributions from the development team, Aaron Arakawa and Grace Bae. The Pilpul concept is an approximate hybrid of Pet Park and Kaleidostories, borrowing the interface and MUD concepts of Pet Park and the storytelling and biography writing activities of Kaleidostories. In addition, there are many new concepts introduced. Each aspect of the Pilpul vision is analyzed. The section will begin with a sample scenario conceived during the design of Pilpul.

3.1 A Sample Scenario:

Aaron is a 12-year-old boy. He logs on to Pilpul and sees an animated character with his face standing in his room. His friend Skylar, another user, is sitting in his room and his role model Garrison Hearst is there as well. He talks to Skylar using the chat feature and reads Garrison Hearst’s biography just for fun. (Figure 3)
Figure 3: A Pilpul mock-up screen shot.

He exits his room and decides to go to the sports temple. There he finds many sports heroes: Steve Young, Michael Jordan, and Jerry Rice among others. Scotty’s friend Mikey is in the museum as well. They say hi to each other. Scotty then notices in the museum a facsimile of football player Bryant Young, who he has been a fan of for his work ethic. He decides that he will make a copy of Bryant to take home and program a personality into him. He gets home with his brand new Bryant Young and thinks about how he will program his personality. He then realizes that he doesn’t know anything about him except for his football prowess, which doesn’t give much insight into how he interacts with people. Scotty looks him up on Yahoo.com and finds a number of articles. He reads about Young and how he always encourages his teammates. He decides to program Bryant to greet people by complimenting them. Below is a hypothetical script that would accomplish this.
to greet (person)

if (person.gender == male)
{
    say “Hey” + person.name + “, have you been lifting weights?”

if (person.gender == female)
{
    say “Hey” + person.name + “, nice shoes.”
}

In the next article, Aaron learns that Young had two bones in his right leg broken by teammate Ken Norton in an accident during last week’s football game and he will be out for a year. He reads that Young reassured Norton, “Kenny, things happen and they happen for a reason.” Aaron was impressed that Young was not embittered towards his teammate. He enters this story, and attaches the value of responsibility to it. He has Bryant Young tell this story to everyone who clicks on him as a lesson in life. On the following page is a hypothetical conversation between a role model and a user.

Person 1:
“Tell me a story, Bryant Young”

Bryant Young
“I broke my tibia and fibula in last Sunday’s game when my teammate Ken Norton ran into me. He felt terribly guilty. I told him ‘Kenny, things happen for a reason.’ I try not to place blame on people or circumstances for my situation. Value: responsibility”

He is starting to read more about Bryant Young so he can program more personality into him when an alarm goes off on his screen- he has a new e-mail message. He clicks on the mailbox and the message pops up on his screen- “intercultural marriages discussion at the city forum, 3:00 P.M.” It’s 2:57 now so he starts out for the city forum.
3.2 Borrowed Features
The following are features of the Pilpul vision borrowed from related projects.

3.2.1 Navigation
The Pilpul world is envisioned as a system of interconnected rooms. Each room has a right and a left exit to adjoining rooms. The room and its contents are displayed as 2D graphics. Navigation between rooms can be achieved by clicking on the exits on the screen. This method of navigation is borrowed from the Pet Park interface.

Bers envisioned the Pilpul world being divided into three distinct regions representing different aspects of community. There is a residential area, where each user has a room where they can express themselves by populating it with objects and artifacts. In addition, there is a museum area containing several museums and temples where users can go learn about ideas and different cultural symbols. Finally, there is a community area where users can meet and discuss issues. In Pet Park rooms are connected in a linear fashion. If Pilpul were laid out this way, all the rooms in each area would be connected in a line, and the last room in one area would be connected to the first room in the next area. This is not an ideal design for Pilpul, as it would be inconvenient for a user in the last residential room wanting to visit the community discussion room to have to travel through all the residential rooms and all the museums and temples. The ideal Pilpul navigation interface conveys the idea of the three distinct areas and facilitates speedy transportation between the areas.

One way to do this is to have always visible a small map with a visual representation of each of the areas. Each of the areas is laid out in the same fashion as Pet Park, with rooms connected in a one-dimensional line. When any of the areas is clicked on, the user is transported to a room welcoming the user to that area. This room will have an exit to the first room in the area. The user will still have to travel through many rooms to get from the first residential room to the last residential room, but travel from any residential room to the community center is instantaneous. Another possibility is to create a map for each area where users could click on the room they wanted to visit and be sent there.
3.2.2 Interaction with Characters and Objects
The goal is for actions to be performed on characters and objects onscreen by using the mouse. When a character is clicked on, a menu of options appears. The user may choose a number of editing or interactive options performed upon the release of the mouse button. Additionally, the user may click on the background for a menu of options for creating rooms, role models, and objects.

3.2.3 Storytelling in Two Forms: Narrative and Interactive
Storytelling is an aspect of the Pilpul design taken from Kaleidostories. There are two forms of storytelling in Pilpul, narrative and interactive. The narrative form is one-dimensional and the storyteller and listener are distinct. Interactive storytelling occurs when the storyteller's story depends on responses at certain junctions, made by the listening party, which can be a user or computer controlled avatar. Storytelling behavior needs to have a distinct interface for creation and the MUD presentation of conversation needs to indicate when storytelling is taking place.

3.2.4 Collaborative Biographies for all Avatars.
One of the goals of Pilpul is to allow collaborative biographies for the role models and users in the world. This is also a goal of Pilpul borrowed from Kaleidostories. All user characters and role models would have a biography. For role models, to have multiple perspectives on the role model's life and accomplishments is a valuable thing since one person seldom knows all there is to know about a particular person they admire. It is also possible that different people admire a particular role model for different reasons, and what they include in the biography will reflect that. This will be referred to as a collaborative biography. With many people involved in writing a biography it becomes well rounded and complete, with many different aspects of the role model's life represented. Users who admire a role model for one reason will be made aware of different perspectives on his or her role model and may change the way that he sees it himself and will also provide insight into his or her own reasons for liking the role model.
There are two types of characters that populate the Pilpul world, user characters and role model characters. Role model characters should decidedly have collaborative biographies. It is also desirable to have users enter their own biography, but it is not clear if it should be collaborative. If it is collaborative, however, it will help the user introspect by giving him outside perspectives on behavior patterns that he may be oblivious to.

### 3.3 New Concepts

There are also a few envisioned aspects that were not found in Pet Park or Kaleidostories, namely the ability to customize animated characters and for users to be able to send e-mail to each other.

#### 3.3.1 Customizing graphics.

It is desirable to have avatar graphics have at the very least actual digitized faces of the users and role models. This would make it that much easier for the avatars to be connected to something in the real world. In addition, having the user and role models represented by generic graphics on the screen would probably decrease the user's personal attachment to his or her creation. There is less of a sense that the object on the screen represents a real life person. This was demonstrated in Kaleidostories, where there was a stronger sense of humanity in the entries where kids included pictures of themselves than in those with no pictures.

In the ideal Pilpul application, at the time of creation of a new avatar the user would be prompted to place a picture of the character's face on the scanner and Pilpul would automatically generate a body for the character. This body would be able to walk around with the digitized face of the character.

Similarly, if a user wishes to introduce an object to the Pilpul world he or she would ideally scan the object in or take a digital picture and it would automatically be incorporated into Pilpul. It is desirable for an object representation to be scanned in because to have the user choose a graphical representation of an object from a preset
library of graphics would severely restrict the range of objects the user can effectively create in Pilpul.

3.3.2 E-mail system:
An e-mail system is desirable to disperse information about an event in the Pilpul world, such as virtual meetings or discussion groups. Also, this provides a medium for feedback about a user’s contributions to Pilpul, creating the opportunity for agreement, criticism and suggestion.

3.3.3 Conversational Programming
This concept of identity manifested as behavior, not just knowledge, is central to Bers’ vision of Pilpul.
Ideally the child would be able to program situational responses into their role models.
The act of expressing the conversational behavior of the role models as a script hopes to accomplish three things. First, it will lead kids to reflect on conversational patterns, and what values these patterns embody. Second, writing conversation in script form creates opportunity for reflection and modification. Third, the personal attachment that results from building the role model and its conversational patterns helps the children internalize the values and conversational techniques they externally observe.

4. Functional Goals of Pilpul
This section defines the goals of the Pilpul project from a high-level implementation perspective. It covers the concepts Pilpul should convey to the users and actions users should be able to perform.

4.1 The World
The vision of the world is a system of interconnected rooms that contain objects, role model characters, and user characters. The first goal is to create a world where users are able to roam around the rooms and interact with other users, and with the objects and role model characters that populate the world. In addition, users are able to pick up and drop objects, and create new objects, role models and rooms.
4.2 Graphical Representation Capability
The basic goal is to create a 2D still-life graphic representation of the room and its characters. Each user should have his own point of view. A 2D animated world or even a 3D animated world may be later possibilities (At the time of this paper a 3D animated version of Pilpul is under construction using the Microsoft 3D Virtual Worlds SDK).

4.3 Supporting Multiple Users
Many users should be able to explore the world at the same time. The users would be able to log on and off asynchronously, that is at different times. Users would be able to come and go as they please, interact with each other, and not disrupt each other’s sessions when they log in and out.

4.4 Homerooms
Each user would have their own homeroom they can return to and populate with objects they create and copy and bring home from elsewhere in the world. When users log on, they would be initially placed in their homeroom. Other users may visit these homerooms but not modify them.

4.5 Interaction Between Users
The vision of Pilpul would have users communicate with each other while on-line to exchange pleasantries and comments about the world. The idea is to use a “chat room” style command line window at the bottom of the screen. A more sophisticated feature that would be more complicated to implement is to have cartoon balloons pop up above the speaker’s head.

4.6 Interaction Between Users and Objects
An essential part of the Pilpul concept is interaction between users and computer-controlled characters and objects. This interaction may be in the form of telling stories and biographies, a form of one-sided conversation. Ultimately however, rolemodels and objects would be programmed to create scripted two-way conversations with the user.
This is one of the key constructionist aspects of Pilpul, as users explore the dynamics of conversation and its relation to values through creating conversational templates. The role model or object would initiate the conversation. The user may type his response, and the role model would process and use the information to determine what to say next. The command window in which the conversation unfolds allows scrolling back in the conversation to view earlier exchanges. It would buffer however many lines necessary, perhaps 50 or more.

4.7 Object Creation, Editing and Copying

Another constructionist aspect of the Pilpul concept is allowing the user to create objects and role models and give them tangible personalities that can be reflected upon. It is essential, therefore, to give the user the ability to create new objects. The construction of these objects will unleash the users' creativity and imagination. These objects can be edited and moved by their owners, or creators, and may not moved or otherwise modified by other users. Other users may wish to take a copy of the object with them and input their own stories and biographies into them and should be allowed to do so.

5. Choosing a Platform for Development

This section describes Java RMI, the development platform chosen for implementing Pilpul, and how the decision to use RMI was reached.

5.1 Objectives of the Platform of Implementation.

We needed a platform on which to create a networked client-server application. In this application model, the server stores the data and remote client machines runs the interface, displays the data, and facilitates manipulation of the data. In addition the following are the goals of the client-server application.

- The state of the world needs to persist after the client program terminates so the user may return to the world in the same state.
• The state of the world needs to be the same on all clients. The most sensible way to do this is to store the state of the world on the server and have the clients retrieve the information from the server.

• The state of the world needs to persist after the server program terminates so the world returns to the same state when the server is shut down and restarted.

5.2 Options Considered.
This section briefly describes the three platforms considered for implementation: Intel IDMOO SDK, Java socket servers, and Java RMI. Java RMI was eventually chosen as the platform.

5.2.1 Intel IDMOO SDK
The first option was Intel’s IDMOO SDK, the engine that powered Pet Park. This was the chosen platform for implementation initially. It was chosen because it already provided the functionality we were looking for, namely it was a networked client-server multi user domain with persistent world state and 2d graphical display capability. After months of attempts trying to get IDMOO to run in any capacity without any substantial support available either on the Intel side or the Media Lab side, it became clear that it was a waste of time trying to get it running. However, in these attempts the Pet Park source code was studied in depth and it was discovered that it had a nice framework for keeping objects on a network of clients. The Pet Park object model was valuable in formulating the eventual design.

5.2.2 Java Socket Servers
When it was clear that Intel IDMOO SDK was no longer a realistic option, the next option became to build a client server model from scratch using socket servers, and using the framework to support an object model similar to Intel’s. This would have involved establishing an extensive protocol between the server and clients for broadcasting events. When an event would occur on a client, the protocol would communicate the change to the server. The server would then broadcast a message to all the other clients so they could update their local copy of the world. This would have required giving each client
its own model of the world and asynchronous messaging between the clients and the
server. This would be very, very complicated to do with a large number of points of
failure.

The third option was Java RMI, which in the end was chosen as the distributed object
framework for Pilpul.
Java RMI is described in the next subsection.

5.2.3 The Chosen Platform: Java RMI
Sun Microsystems has implemented a distributed object management system in its Java
language specially designed for the development of client-server applications. Java’s
Remote Method Invocation (RMI) package allows for objects shared among many
clients, and more importantly, provides consistency of object data across all clients. The
way RMI works is that clients have virtual pointers to actual objects on the server
machine. These objects reside on a process called the RMI registry. The RMI registry can
accept requests over the network to access methods of any object residing on it.
Whenever these remote method calls are made, they are executed on the machine on
which the RMI registry is running, and the results are passed back over the network to the
caller. The caller never has an actual copy of the object. The only actual implementation
of an object lies on the RMI registry machine. Changes are observable from all remote
processes connected to the RMI registry. Therefore, object state consistency is not an
issue, eliminating a huge engineering problem! Performance is mediocre, however,
because each access to an object, no matter how trivial, requires two time expensive
network calls, from the client to the server, and the server back to the client, in addition to
any processing time needed to execute the method.

5.3 Achieving Object Persistence
One of the goals not addressed yet is the goal of having objects that persist after all
programs are closed and the server machine is shut down. Pet Park had a built in
mechanism to do this, using files. One option would have been to use files in conjunction
with RMI to store the state of all objects. Based on my previous experience with
Kaleidostories, I found that it would be much easier to implement a data storage and retrieval system using SQL to write to and read from a database. The following subsection briefly outlines the advantages of using a database over a file system.

5.3.1 Database vs. File System
Using a database over a file system provides:
- Fewer lines of code to access and parse data.
- Organized data that can be read, re-organized, altered, and deleted using the database user interface.
- Data is organized in a clear, understandable form that is easily manipulated by an administrator who doesn’t know how to program.
- Modularity. If the platform changes, the SQL code doesn’t need to be changed. The database connection code may need to be changed in but one place.

6. Implementation
This section describes the actual implementation of the Pilpul application. First the data structures designed to represent the world are described. The remainder of this section is divided into analyses of the major stand-alone software components that comprise Pilpul, the client application and the server application. A third component, a web server, serves as an accessory to the other two processes and is also briefly covered.

6.1 Data Structures
All objects in the Pilpul world are implemented in Java RMI. This allows a single implementation of each object to be shared among all clients. The single implementation is stored on the server. Each client who needs to access an object on the server is given a remote pointer to the object (Figure 4). When the client makes a method call, the call is passed over the network. The method is on the server, and the return value is passed back over the network to the client. These return values can take the form of Strings and more importantly pointers to other remote objects on the server machine. Thus the client has indirect access to any number of objects on the server. Clients may mutate objects on the server by calling mutator methods remotely. Having a single instance of each object
rather than giving each client their own copy eliminates the need to make local changes on the client and inform the server and all other clients of these changes.

![Diagram of the Pilpul client-server model]

Figure 4: The Pilpul client-server model.

The Thing base class of Intel's IDMOO system inspired Actor, the base class of Pilpul. An Actor is anything in the world that is represented graphically on the screen, including stage objects such as Rooms. The Actor base class contains information essential to all Actors in Pilpul.

The defining characteristics of an Actor are that it has a description, an image URL, an owner (another Actor), a value, and a story. The Actor also has a slot for a single possession, of type Actor. This allows users to carry Rolemodels, Objs, and other Users, although carrying other Users is disallowed.

The following are the essential slots of the Actor class:

String name;
The owner slot of an Actor is important for distinguishing who can pick up or modify the Actor. Only the Actor’s owner may change its description, add a story, or pick it up. Other Actors may access these properties but not edit them.

The database slot is used by the ActorImpl to write to the database. Whenever the user makes a change, the ActorImpl updates the Actor’s database record to reflect this.

The script attribute of the Actors will in the future be a template for conversations between the user and his or her role models.

Users, Rolemodels, Objs, and Rooms extend the Actor class. Room extends Actor to inherit the description and imageurl. Other than that, Room uses none of the other slots. In this implementation, the amount of space wasted by slots with no values grows with the number of rooms. An alternative would have been to have a BasicActor class, off which Room and Actor branch. BasicActor would have only the most essential slots, being the name, description, and imageurl.

A Room is an Actor with pointers to other Rooms it is connected to. The Room has pointers to Exit objects, which in turn point to Rooms. Exits contain no other information, but they could contain later the location and size of the exit to facilitate arbitrarily placed exits and “teleporters” to non-adjacent rooms. A Room also has a list of Actors that reside in it.
Actors do not store x, y coordinates. Rather, these coordinates are computed as the Actors are drawn. The Actors are not placed in any particular order on the screen. This is not the most elegant solution but the fastest to implement given the short amount of time allotted for development. The algorithm for placing actors is to start from the left and draw to the right until the edge of the screen is hit, at which point the following actors are drawn scaled down (using the built in Java image scaling function) and drawn a row back.

One problem encountered was that Images could not be passed from the server to the client within the Java RMI framework. Because Actors could not have pointers to Images, they need an alternate means to refer to their images. The solution was to place the actual Images on a web server and have the Actors refer to them by URL. When the client need to draw an Actor, it looks up the URL and grab the image. Then, it places the image in a hashtable with the Actor's name as the key. The next time the Actor is drawn, the graphics engine first looks in the hashtable to see if the Image was cached there, and if so draws it. If not, it visits the URL and fetches the image.

The actorlist is implemented as a Vector of Actor objects. The Vector was the most obvious choice because it can hold Java Objects of any type, and therefore can accommodate Actor objects. To draw the Actors in the room, an Enumeration is made from the elements of the actorlist Vector. All the elements are enumerated through and drawn in arbitrary order.

6.2 The Server
This section describes the implementation of the server and all of the functionality it provides.

6.2.1 The Server Object
The server object is the core of the distributed object network that implements the Pilpul world. It stores all objects in the world and provides clients access to these objects through remote methods. Implemented in Java RMI, the server object is instantiated and
placed on the RMI registry process, where its services can be used from remote
machines. All objects residing on the server object are also remote objects, and one of the
server object's functions is to distribute remote pointers to these objects. These objects
may then in turn be accessed remotely (see Figure 4). Copies of the objects in the Pilpul
world are stored in string format in database tables. The server object saves and restores
objects to and from the database by executing SQL statements. From this point forth the
server object will be referred to as simply the server.

6.2.2 Initialization of the Server
The server application instantiates the server object, registers it on the RMI registry, and
exits. From then on the server sits on the object RMI registry process and services
method calls from clients. Clients gain a pointer to the server object via a network request
to the RMI registry.

6.2.3 World Persistence
The server keeps two copies of the world object and database record form
simultaneously. The object form can only persist while the server is running. Once the
server is killed, the object representation is lost forever. This is why a database
representation is maintained to mirror the state of all objects. When the server dies, the
database representation persists. When the server is restarted, the objects can be restored
from information retrieved from the database.

6.2.4 Changes to Objects
Objects are changed through calls to remote mutators. Changes to object states are
designed to propagate to the database file as well. Every mutator method implemented on
the server side makes a SQL call to perform the same change in the database. This makes
each object mutation expensive. An alternative would have been to have objects scanned
every few minutes or so and the database updated. Perhaps each object changed could be
flagged, and these objects could be written to the database to avoid having to rewrite
every single object. Even this would be expensive, as even fields that don't need to be
modified would have to be to ensure that all changes would be reflected. An even more
undesirable option would be to have the server administrator hit a save button every so often.

6.2.5 Providing Clients with Information
When a user logs in, the client he is on connects to the RMI registry and fetches a pointer to the server. The server object provides the client with access to all the information necessary to display the world to the user except for the actual graphics. The graphics are stored on a separate web server separate from the server residing in the RMI registry. Then the client requests from the server a pointer to the user and a pointer to the room the user is in. The server looks up the user in the database, verifies the password, and instantiates a new object for that user. Users are instantiated as needed rather than in advance.

6.2.6 Database
Each object is represented in a database by translating each instance of an object into a record, in a database table. A table exists for every type of object: Rooms, Users, Objs, and Rolemodels. The columns of each table contain text representations of each defining property of the object being stored. For instance a Room stores a name, a graphic URL, a description, the name of the left adjacent room, and the name of the right adjacent room.

As objects in the world are modified, the database record is updated using a SQL UPDATE command, which changes the contents of an existing record. As the Java representations of the objects are updated, the database is updated to reflect the changes in real time. This eliminates the need for a “save” function. In addition, when a user closes the RMI registry or shuts down the server machine, no changes are lost. This is important also in that when another user logs in, he sees the world as the same as every other user sees it. If changes were only saved periodically, there could be inconsistent world states on different clients.
6.2.7 Re-creation of the World

When the server process is started, the objects that constitute the world are re-created from the saved information in the database. The doubly linked list of rooms is created. Adjacent rooms point to each other. Recreating rooms is done in a certain sequence. All the rooms are loaded before any linkage is possible. First, the Rooms are instantiated from Room table records and loaded into a Vector, with null pointers for the left and right exits. This Vector is enumerated through and for each Room, the left and right exits are set to point to the appropriate Room objects. The pointers are subsequently set to the now existing Room object with that name.

6.3 The Client

This subsection describes the client and its implementation.

6.3.1 Overview

The client process can be run on a remote machine with respect to the server. The client is responsible for connecting to the server and displaying the world to the user. It also lets users interact with the world through the keyboard and mouse. It is implemented as a Java applet and as such may be run using the Java Appletviewer or from any Netscape browser running Java 1.2. Problems were encountered running Pilpul in Internet Explorer.

The Java Applet is called the ClientEngine. It is named this because it performs every I/O function required: displaying the world, gathering input from the keyboard and mouse, and communicating with the server.

6.3.2 ClientEngine/User Interface

The client engine is responsible for exchanging world information with the server and displaying the current world state to the user. The interface is the process of the client engine that displays two-dimensional representations of the current room and all the characters and objects in the room. In addition, it gathers user input from the mouse and keyboard to interface the user to the world. The client engine is implemented as a Java
applet so Pilpul may be run from a web browser. The Java 1.2 API has several useful tools for displaying graphics and gathering user input, making it ideal for Pilpul. The Java AWT package provides techniques for displaying graphics and detecting mouse clicks and keyboard entries.

The client engine has slots for the user and the current room the user is in, and a remote pointer to the server. Any login or update requests are executed by invoking server methods remotely.

The client engine detects user actions such as mouse-clicks by listening for Java AWT 1.1 Events (Java 1.2 uses the Java 1.1 event model). The event listener model automatically calls methods registered to listen for certain events. When the mouse is clicked, the method registered for listening to mouse-clicks is invoked. This method then extracts information about where the mouse was clicked, cycles through the Actors and exit positions to extrapolate the object clicked on, and calls the appropriate method to carry out the appropriate action.

The following actions are possible: exiting a room, talking to an Actor, editing and viewing the Actor’s story, editing and viewing the Actor’s value, creating a new Actor, creating a new Role Model.

6.3.3 Login
The login process is rather involved. On the client side the username and password is gathered in a pop-up dialog window. These are fed to the server, which returns pointers to the user and his room, or an error code if the user does not exist or an incorrect password is entered. The room and user objects contain information needed to draw the world. This procedure is described in the next section. The paint() method of the client engine was then called for the first time.
6.3.4 Displaying objects
Displaying objects was done by overriding the Applet's paint() method. Java has built in insurance that the paint() method is called every time the canvas needs to be redrawn, for instance if an obstructing window is removed.

6.3.5 Editing Actor attributes
Actor attributes can be edited by clicking on the Actor and selecting "Edit Story," "Edit Description" from the pull-down menu. Only users who are owners of the Actor may edit these properties. This client checks to see if the owner slot of the Actor is the same as the user slot of the ClientEngine object.

6.3.6 Adding a new object/role model
The client allows the user to add new objects and role models to the world. Pilpul takes the user through several dialog boxes to determine what name, image URL, description, story, and value the new Actor will have. Once all the data is gathered, a server method is called to create a new object. The server method takes these parameters and instantiates a new Obj implementation. It also creates a new database record for the new Actor. Then it adds the Actor to the actorlist of the room it was created in.

6.3.7 Adding a new role model
Adding a new role model follows virtually the same procedure as adding a new object.

6.3.8 Navigation
The client interface allows the user to move from room to room exploring Pilpul. The client listens for mouse clicks on "hot spots" on the screen over the doorways of the room being displayed. The graphics of each room are designed to have each exit in the same sets of x, y coordinates in every room. The client checks for mouse clicks in these areas. When these hot spots are clicked on, the client engine is informed that a room switch is in order.

The room switch is carried out by replacing the current room pointer on the client (name
the actual property) with the pointer to the room at the destination of the exit. This is easily facilitated with pointers to the adjoining rooms in each room data structure.

Each character and objects (excluding user avatars) has stories attached to them. These stories and are implemented as String properties in each Actor. The description and biography slots are similarly implemented.

6.3.9 The Graphics Engine

The graphics engine draws the background and all of the Actors in the Room’s actorlist. Actors are placed on the screen left to right and front to back, in that order. A more sophisticated algorithm may replace this simple algorithm in the future. For instance, user characters could be placed on center stage and objects drawn in the periphery. Perhaps objects could be assigned fixed locations. Users would be able to click and drag objects to new locations on the screen. In this case computation would be required to ensure objects do not completely obstruct the view of other objects.

6.3.10 Refresh daemon

A refresh daemon runs in the background and refreshes the pointer to the actorlist from the server object. The entrance and exiting of any users or creation of objects is reflected in the updated actorlist. After the actorlist was refreshed, the contents of the room are redrawn.

The refresh daemon was implemented as a thread that was spawned off the graphics engine. It’s function was to sleep for a specified length of time (in this case, 5 seconds so the client wasn’t bogged down by refreshes- a refresh takes about a second and no user input is possible during this time), update the actorlist, and call the graphics engine’s repaint() method.

6.4 Difficulties Encountered in Development

There were some issues developing in Java RMI. These are described in this subsection.
Java RMI makes running the server a two step process. The system administrator needs to run the RMI registry, then the server application. This sequential process adds one more point of failure and makes running the server complicated because it needs to be done sequentially. To get around this, a batch file was written to automate the process.

In addition, development was inefficient because after the Java files are compiled, Java RMI requires that stubs and skeleton classes be compiled. This makes compiling a three-step process. Also, two separate files needed to be maintained for each remote object class. Thus when one was changed, often the other needed to be changed as well. This was no more complicated than maintaining a header file for each C++ file but is more complicated than maintaining an ordinary Java file. To top things off, the stub/skeleton compiler was on the slow side.

Testing was tedious because in order for any changes to an object to take effect, a long sequence of compiles and restarting of software components was required. These processes were automated as much as possible using batch files. However, because the RMI skeleton/stub compiler is so slow, it didn’t make sense to recompile every remote object class every time a small change was made, as it would take close to 2 minutes. Thus it was undesirable to automate the whole process.

6.5 The Web Server
A third software component, the web server, was necessary to assist the client and the server. The web server serves to distribute graphics and applet code to the client. The client wishing to run Pilpul loads the applet code from the web server by visiting a URL. Images used by the client applet are also loaded from the web server. We chose to run the web server on the same machine as the server. However, the only restriction is that the images and the web server reside on the same machine. The web server could run on a completely different machine than the server. Perhaps this would improve performance by not tying up server resources running web browser services.
7. Technical Accomplishments:
This section recounts the technical accomplishments of the Pilpul implementation. The main network and world representation infrastructure was successfully implemented, and most of the essential interface functionality was in place at the end of development. The greatest success was constructing a working server that supported multiple clients logged in simultaneously. The following subsections detail the accomplishments on the client and server side.

7.1 Server
The server was successfully constructed to accept connections from clients running in web browsers on foreign hosts and provide them with access to the Pilpul world. The system was verified to work with up to four clients simultaneously.

- **A memory resident world representation.** Java RMI object implementations of Actor, Room, User, Obj, and Rolemodel effectively represented the Pilpul world. Instances of these objects were placed on the server. Implementing the objects in Java RMI allowed for clients to access and manipulate the objects from a remote host. The RMI framework worked flawlessly, if not a bit slowly.

- **A persistent, database resident world representation and reconstruction into memory.** The database tables were successfully parsed into memory resident objects at server startup. Inputting room and user information into the database by hand initialized the Pilpul world. The success of the restoration procedure was evidenced by the client’s representation matching the underlying room structure and object locations entered into the database.

- **Multiple user logins.** The RMI registry of Java handled this aspect automatically and flawlessly. A user request for login was really a request for a handle on the server object from the RMI registry. Once the client has access to the server object, logging in was as trivial as requesting pointers to the world, after the username and password was verified against the database. This login was tested with multiple users on multiple machines. One oversight was there was no mechanism in place to prevent the same user from logging on to the system multiple times.
7.2 Client

The client achieved its goals of interfacing with the server, displaying a 2d graphical representation of the world, and allowing navigation from one room to another. In addition, the client successfully facilitated the creation of objects and role models and successful communication of this information to the server.

- **Connection to server.** This was achieved by connecting to the RMI registry and requesting a handle to the server object. The RMI registry had a bug of rejecting the first three requests for a connection, regardless of the source, but subsequently it would accept and service all connections without incidence of failure.

- **Netscape support.** The client applet was tested in both the Sun Java Appletviewer and in Netscape and Microsoft Internet Explorer environments, on both Windows and Macintosh platforms. The applet ran in the Windows Appletviewer and Netscape but not in IE or Netscape Macintosh. This was considered a victory because Windows Netscape is acceptably widespread to be an effective form of distribution. That Pilpul ran as an applet in a browser was a huge victory because it made distribution trivial and facilitates widespread access without downloading and installing components.

- **Simple navigation.** Users were able to move from room to room visibly to other users.

- **Object and role model creation.** Users were able to create objects and role models, assign them graphics, and write and edit stories and biographies/descriptions about them. (Figures 5 & 6)
- **Refresh daemon.** The refresh daemon updated the screen every five seconds, so changes such as other users entering and leaving the room, or creation of objects, would be reflected every five seconds.

Figure 5: An actual Pilpul screen shot. Here the user is editing a story he wrote about a role model.
8. Future Work:

Future work on Pilpul would extend the functionality of the client. I wish to implement the following features that were left out of the implementation because of time constraints.

- **Collaborative biographies.** The collaborative biography, which I have implemented in the previous project Kaleidostories. This feature would allow all users to contribute to the biography of a role model. Each user may have a uniqmodel’s life and it is desirable for Pilpul to be able to capture as many perspectives as possible. The implementation is straightforward. Create a database table with columns Rolemodel, Username, and BioEntry. When a user adds an entry, the user’s login name, the role model’s name, and the biography text are entered into the database. When a user wishes to view a role model’s biography, Pilpul grabs all records from the database.
with the role model’s name. It displays all the biography texts and notes who the author was.

- **Real time interaction.** In Pilpul, users will want to be able to interact with each other in real time.

- **Conversational programming.** Conversation between users and rolemodels, programmed by the users, will help users introspect into the dynamics of conversation.

- **Room wizard.** I would have liked to implement a room wizard for users to create their own homerooms. Currently the system administrator creates the rooms by hand by entering the room name, exits, and graphic file into the database manually. The wizard would allow the user to upload graphics.

- **Uploading graphics.** It would be nice for the user to be able to upload graphics onto the server from their client machines. The first problem is to transfer the graphics file to the server. There was no readily available quick and easy Java technique to serialize and send graphics files over the network. I would have needed to implement a socket server to transfer files byte by byte. Another alternative is to upload images from the client to the server using a file upload Java Servlet. However, this technique separates the file upload process from the client engine, which is undesirable. An interesting side note is that images are stored in a folder whereas all other data used by Pilpul is stored in database tables. Currently the database cannot store images, only text and numbers. With up and coming technology of Object Relational Databases, databases will be used to store images as well as entire data structures.

- **Copying objects.** Another feature that is part of the Pilpul vision is the copying and taking of objects and role models. If the user likes the role model of another user, it is desirable for the user to be able to copy the item and take it with him back to his room. The copy has no of the stories and conversations of the original, as the user will program his own stories and personality into the object or role model from his unique perspective.

- **Multiple Pilpul worlds.** It would not be hard to allow multiple worlds on the same Pilpul server. The database would need to have a separate set of tables for each world, and the client would need to be modified to allow the user to select the world to enter.
The server process would need to have multiple world representations in memory, one for each world. This is as straightforward as duplicating the server data structures that exist for the single world.

9. Bibliography


