Doppelgänger Goes To School: Machine Learning for User Modeling

by

Jon Orwant

S.B., Computer Science and Engineering
Massachusetts Institute of Technology
(1991)

S.B., Brain and Cognitive Science
Massachusetts Institute of Technology
(1991)

SUBMITTED TO THE PROGRAM IN
MEDIA ARTS AND SCIENCES
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE IN MEDIA ARTS AND SCIENCES
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 1993

Author: _____
Program in Media Arts and Sciences
August 6, 1993

Certified by: _____
Walter Bender
Principal Research Scientist, MIT Media Lab
Thesis Supervisor

Accepted by: _____
Stephen A. Benton
Chairperson, Departmental Committee on Graduate Students

Doppelgänger Goes To School: Machine Learning for User Modeling

by

Jon Orwant

Submitted to the Program in Media Arts and Sciences
on August 6, 1993 in partial fulfillment of the
requirements for the Degree of
Master of Science in Media Arts and Sciences

ABSTRACT

One characteristic of intelligence is *adaptation.* Computers should adapt to who is using them, how, why, when and where. The computer's representation of the user is called a *user model*; *user modeling* is concerned with developing techniques for representing the user and acting upon this information. The DOPPELGÄNGER system consists of a set of techniques for gathering, maintaining, and acting upon information about individuals, and illustrates my approach to user modeling.

Work on DOPPELGÄNGER has been heavily influenced by the field of *machine learning.* This thesis has a twofold purpose: first, to set forth guidelines for the integration of machine learning techniques into user modeling, and second, to identify particular user modeling tasks for which machine learning is useful.

Thesis Supervisor: Walter Bender
Principal Research Scientist, MIT Media Laboratory
This work was supported in part by IBM and the NIF Consortium.

Doppelgänger Goes To School: Machine Learning for User Modeling

by

Jon Orwant

Thesis readers

Reader: _____

Pattie Maes
Assistant Professor of Media Technology
MIT Program in Media Arts and Sciences

Reader: _____

Judy Kay
Senior Lecturer
Sydney University Department of Computer Science

Preface

Writing a thesis is hard. On one hand, you're trying to convince the reader that you're knowledgeable and methodical, and that you've made a worthwhile contribution. These goals are best served by a formal, didactic style. On the other hand, you're trying to spread your ideas, and that requires a good read—prose that people will enjoy and recommend to others.

I've chosen an informal writing style because I'd like a lot of people to read this. There is a potential for a wide audience, because instead of targeting a particular problem, this is a "cross-pollination" thesis: I'm advocating a set of approaches for a set of tasks, rather than suggesting the best way to solve one task.

This thesis is available for anonymous FTP from media.mit.edu (IP address 18.85.0.2), in the directory pub/orwant/doppelganger.

Thanks to the people who have helped me develop DOPPELGÄNGER: Michelle McDonald, Gillian Lee, and Klee Dienes. Michelle in particular has written a significant amount of code described in this thesis, including the code relating to advertisements and the dopmail parser. Klee is facilitating conversion of DOPPELGÄNGER to AFS and Kerberos 5.0. Gillian is working on the storage of portable user models on PCMCIA cards. Mark Kortekaas wrote C and Postscript code that facilitates the generation of Markov Model diagrams.

Thanks to Walter Bender for providing me with the opportunity and the environment in which to perform my research, and IBM and the News in the Future Consortium for providing financial support. Further thanks to my readers, Judy Kay and Pattie Maes.

Jon Orwant
orwant@media.mit.edu
August 6, 1993

Contents

List of Figures

Chapter 1

Why Model Users?

This chapter provides a motivation for user modeling in general, and my system, DOPPELGÄNGER, *in particular.*

A friend of mine once recounted an episode from his childhood. He was showing a computer program to a neighbor, and the program asked the neighbor to type his name. He typed the characters "BOB", but didn't hit the return key. When the program did nothing, he complained, "Why isn't it working?" My friend retorted, "You can't expect the computer to know your name! You have to hit the Return key!"

The point of user modeling is this: Your computer should not only know you, it should be an expert about you. I'm not suggesting that they hit your return keys— that would generate more problems than it solves. And I'm certainly not advocating computer illiteracy to the extent that computerphobes should be free of the return-key burden. Man-machine interfaces can only be simplified so much. Every task has an "interface limit"— the more complicated the task, the more complex is the simplest interface. There's simply no way to banish all the frustration inherent in

any complex task.[1] The more ways there are to use something, the more ways there are to make mistakes.

But your computer could allow a typo in a long name. Or it could sense you coming before you get to the computer, and log you in automatically. Or it could recognize your voice, or your face [TP91]. It could sense the pressure of your footstep [Spe88], [Spe90], [Spe93], or the pressure with which you hit the keys on the keyboard [GB86]. Once you logged in, it could customize your environment, infer your goals and help you reach them, teach you, remember what you had trouble with before, and perform tasks for you without explicit instruction. Computers should act like intelligent conversational partners. Instead of telling the computer what to do, people should be able to show the computer what they want, or better yet, have the computer *deduce* what they want.

When a computer customizes its behavior for individuals, it is said to be *modeling* its users. The study of how best to accomplish this feat is called **user modeling.**

1.1 People are Complex

The more computers know about their users, the better they can serve them. In the ideal computing environment, your computer will know all about you—where you are, the topics that interest you, your favorite foods, the differences between you and other people, and (perhaps most important) how much you value the privacy of this information.

[1] My utopian interface is a computer with only a large red button marked NO; users swat the button when the computer isn't doing what they want. The computer figures out what the user wants instead, and does it.

1.1.1 Natural Language

Our greatest source of information about people we see on a daily basis is *natural language*—what they speak, or what they write. And for the majority of user modeling researchers, who want to infer the user's cognitive state, their sole means of obtaining information about the user is through text typed by him. Complete computer understanding of natural language is still many years away, but there are plenty of tools that provide clues to understanding.

A model of what the user is thinking can help computers understand their natural language, and in turn, understanding the natural language can help computers understand what people are thinking. Thus, natural language research and user modeling research have a lot to offer one another, and there are many projects working at this junction. These projects can be further divided into two classes, *generation* systems and *acquisition* systems. A good review of the relationships between user modeling and discourse modeling can be found in [KF91]. *Natural language generation* systems tailor natural language output of a system to users. By inferring what a user knows, the computer can present information that neither repeats concepts the user already knows, nor assumes knowledge of concepts unfamiliar to the user. *Natural language acquisition* systems modify the interpretation of hard-to-parse natural language based on knowledge about the user.

1.1.2 Planning

Another common direction for user modeling is *planning.* When using a computer, a user often has a particular goal or plan in mind. If the computer is aware of it, that information can be used to help the user reach his goal or fulfill his plan. *Plan recognition* systems attempt to infer and represent these plans.

Detecting user misconceptions is an important part of an effective interface. Consider the interchange in Figure 1.1, from [Kap82] (printed in [Kob91a]):

Without presupposition analysis:

```
User: Which students got a grade of F in CS200 in spring, '79?
System: NIL
User: How many people passed CS200 in spring, '79?
System: Zero.
User: Was CS200 given in spring, '79?
System: No.
```

With presupposition analysis in COOP:

```
User:  Which students got a grade of F in CS200 in spring, '79?
System:  CS200 was not given in spring, '79.
```

Figure 1.1: An example of plan recognition.

Plan recognition is a necessary prerequisite for correcting faulty plans, this being the concern of *Intelligent Tutoring Systems* (ITS). These attempt to teach users through personalized, automated instruction by modeling the user's knowledge and identifying mistakes or inconsistencies. There is an ongoing argument about whether this is feasible, or even desirable [KL91]. Webb uses feature-based modeling [Web91] to describe student cognition at a very high level. Several systems model students' knowledge growth from novice to expert: Chin's system [Chi] does this for the UNIX operating system domain, McCalla *et al.*'s [MGGC91] system addresses recursive programming in LISP, and the system described in [MST91] models this performance independent of the domain.

1.2 People are Different

The Apple Macintosh bills itself as "the computer for the rest of us." And it *is* preferred by millions of people, because it has a simpler interface than its competitors.

But it's not for everyone. Some people will always prefer an interface that trades off ease of use for power.[2]

People want different things when they interact with a computer. And when they interact, they bring with them different skills, experiences, and misconceptions. Human beings are complex systems, and because they're so complex, there are many ways in which they can differ. The executive, the game player, and the information hound all have different tasks, goals, and even styles of interaction. And sometimes the same person is all of these, just at different times.

1.3 People Change

People exhibit change between sessions with a computer, but they also change over longer periods. Yet virtually no user modeling systems, save DOPPELGÄNGER, attempt to model change over minutes, days, months, and years; they usually merely model users for the duration of their sessions with the computer.

As we change, we want our computers' user models to change with us. When I play chess against a computer, I want the computer to know how good I am. And as I get better, I want it to notice that as well. A good chess tutor, one who knows his student and wants him to become more proficient, will play in a way that furthers those goals. He will mentally model his student: predicting what traps will work best, what mistakes the student will make, and even what the student will think when making his next move, and change this model as his student learns.

A large proportion of user modeling research is divorced entirely from applications.[3] It's purely theoretical, and considered part of the AI knowledge representation literature [Kob91a]. This research focuses on representations for user models: how a

[2]NeXTSTEP users get the best of both worlds.

[3]I use the word *applications* frequently in this thesis. It's hard to define exactly, but what it really means is "some *other* computer program." If it helps, think of a personalized newspaper, or an electronic mail program, or a calendar program, or a video game.

system can organize knowledge so as to facilitate reasoning that resembles human cognition.

1.4 Big Stuff vs. Little Stuff

A computer chess program should think while its opponent is thinking, and most importantly, think *about* what its opponent is thinking. It should watch its opponent's eyes to identify what problems he considers worthy of attention. And it's easy for a computer to put on a poker face. It's a bit more difficult, but possible, for the computer to notice when humans lack poker faces.

The Basser Data Project [TBKC91] monitored students' use of a text editor, and suggests that useful information can be deduced from the low-level editor events of keystrokes and mouse clicks. From these events, repertoires and patterns of commands can be deduced, from which statements about each student's learning style can be made. Any computational environment needs to pay attention to this level of data if it is to best adapt to its users.

To me, user modeling is about providing a useful, intelligent repository of *all* possible information about a user. Of course, this is an unreachable goal. Our efforts should then be directed toward accumulating information that will be useful for adapting to user behavior. Some types of information are often deemed uninteresting by user modeling researchers because they aren't "deep", *i.e.* related to some aspect of the user's cognitive state, such as his plans, beliefs, or knowledge. But shallow information—the "little stuff", or "pragmatic" data, as it is known in the user modeling community— is useful for two reasons: first, it is needed by applications trying to make your life easier, as in the calendar program that needs to know you want your newspaper in the early morning on weekdays, and in late morning on weekends. Second, these minutiæ can provide profound insights into deeper states, just as the direction of a chess player's gaze reveals what he is thinking about.

And little stuff is much easier to obtain than big stuff. In the future, as people's lives become increasingly accessible to a computer, the opportunities for user modeling will multiply. This is one of the tenets underlying the DOPPELGÄNGER user modeling system, described in the next chapter.

Chapter 2

Doppelgängers!

This chapter provides an overview of my work, and sets the tone for the next chapter, which specifically addresses the aspects of machine learning that can benefit user modeling.

2.1 The History and Philosophy of Doppelgänger

DOPPELGÄNGER is a user modeling system I've developed over the past three years in the MIT Media Lab Entertainment and Information Technology Group [Orw91a], [Orw91b].[1] DOPPELGÄNGER was originally developed as a component of a personalized electronic newspaper; further discussion of the role of user modeling in this context can be found in Appendix A.

DOPPELGÄNGER can be thought of as a server, with two types of clients: sensors and applications. This is illustrated in Figure 2.1. *Sensors* provide data about people to DOPPELGÄNGER, which applies reasoning to the data. *Applications* make requests to DOPPELGÄNGER for information about people. The intelligence behind

[1] A Doppelgänger is a mythical creature from German folklore that observes someone and slowly changes its appearance to look like him. It is evil.

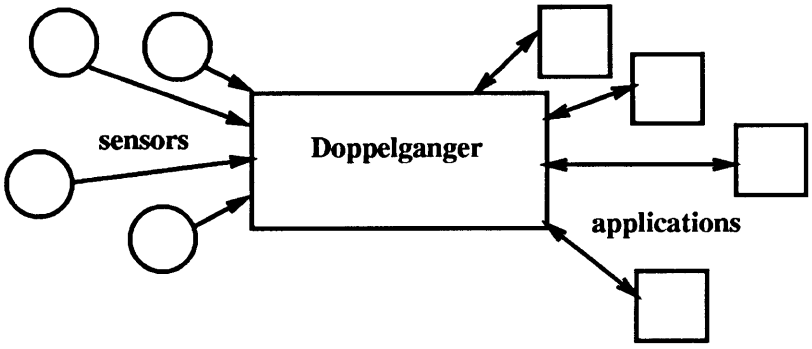

Figure 2.1: The DOPPELGÄNGER architecture.

DOPPELGÄNGER, and what makes it more than a mere database of personal information, comes from the reasoning that is applied to the personal information. This reasoning occurs when information arrives at DOPPELGÄNGER from sensors, *and* when applications make requests to DOPPELGÄNGER.

Formally, DOPPELGÄNGER is an application-independent user modeling system consisting of executable programs, C libraries, a user modeling database, and protocols for communicating about the user. DOPPELGÄNGER accumulates information about people through a large number of sensors, and has an extensible architecture that permits addition of new sensors. *Application-independent* means that it's not built for a specific application such as a personalized newspaper, datebook program, television, or mail program, but instead fields inquiries about users from all of them. Its sensors are passive (non-intrusive), in comparison to active sensors that require intervention by the user. Sensor passivity is gained at the expense of accuracy; since the sensors don't require feedback from users, their information is less reliable and less complete, necessitating better inference engines and a capability for fault tolerance. Although I refer to DOPPELGÄNGER as simply a user modeling system, it can more properly be called a **User Modeling Shell** to distinguish it from user modeling components that are constructed for a particular domain [Kay91].

DOPPELGÄNGER's models contain both short-term and long-term information, and both pragmatic ("the little stuff") and cognitive data. New information is gathered continuously, and models change as the new information becomes available. In addition to inferences on data within individual models, DOPPELGÄNGER obtains *default* information from *communities* which it constructs. Each user is a member of multiple communities of people who share some trait or set of traits. When an application asks DOPPELGÄNGER for information not explicitly present in a user model, DOPPELGÄNGER makes an educated guess by evaluating the contents of similar models and communities.

DOPPELGÄNGER was originally targeted toward one set of applications: personalized electronic newspapers. Personalized newspapers are demanding; to create a presentation maximally tailored to the user, the system must model not only the user's cognitive state, beliefs, interests, and preferences, but also the capabilities of the display in front of him, the amount of ambient noise and light, and whether he's in a hurry, so that the selection of multimedia articles is appropriate for the user's computational environment. Constructing the perfect personalized newspaper is what I call a **UM-complete** task, meaning that any system that can model users sufficiently well for a personalized newspaper will be sophisticated enough to tailor the behavior of most applications. In other words, the amount of knowledge about a person needed to create the ideal personalized newspaper is so great that the same compendium of knowledge will be sufficient to tailor the behavior of countless other applications.

This chapter presents some features of DOPPELGÄNGER that bring it closer to the goal of supporting UM-complete tasks.

2.2 Tolerance

If you forgot everything about other people as soon as they left our presence, you'd have considerable trouble getting along in life. But many user modeling systems do

just this when they remove all traces of the model as soon as the user's session with the computer has finished.

And if you had only one way of gathering information about people, say your ears, you'd have a great deal of difficulty communicating. But many user modeling systems do this by maintaining only one sensor with which to gather data about their users.

The above two criticisms suggest something lacking in many user modeling systems: they should be *robust*. The system should not stop modeling people because its lone sensor fails. Instead, it should maintain many sensors, so that it can degrade gracefully, and compensate for less complete information by making educated guesses based on communities of users within the modeled population. And users should be modeled continuously.

Systems that claim to adapt their behavior to the user should have many ways of gathering information about the user. One of the most important areas of user modeling research should be building and integrating new sensors of information, such as a chair sensor that identifies whether someone is sitting in it, into user modeling systems. There is a reluctance to explore new avenues of information about the user, because of the distance from the lofty goal of understanding cognition. This is unfortunate, because such exploration can be quite creative, fun, and illuminating.

Sensors should be aggressive in attempting to gather as much information as possible. This means they will often fail, or conflict with one another. The "sensor fusion" problem in robotics is well known; briefly, the problem can be stated as "How should a system arbitrate between conflicting sources of information about the environment?" DOPPELGÄNGER makes use of many sensors; its methods of arbitration are discussed in section 3.4.

DOPPELGÄNGER was developed in a UNIX[2] environment; consequently, many of its sensors make use of UNIX to gather information about the user. As operating systems go, UNIX allows extensive access to information about user actions.

[2]UNIX is a trademark of AT&T.

A user modeling system should make use of *all* available information about a user. And the field of user modeling should concern itself with trying to make more information available.

The sensors used by DOPPELGÄNGER (in various stages of completion, anticipation, and disrepair) are:

- A "login sensor" that tracks when people log in to computers.
- Command sensors that keep track of which UNIX commands users are executing.
- Various personalized electronic newspapers.
- "Active badges" that transmit physical locations of users.
- Model-editing programs that let DOPPELGÄNGER know what the user wishes to change. This is feedback about how well DOPPELGÄNGER is doing.
- A calendar sensor that understands scheduling information.
- A face recognition system.
- A program that identifies subscribers to electronic mailing lists.
- Floors that can identify who is walking on them from weight and the duration of the step.
- The "DOPPELGÄNGER seeder", which is a questionnaire that people can use to bootstrap or augment their user models.[3]
- A mail interface to DOPPELGÄNGER to which people can direct natural language input.
- Remote Doppelgängers share information about people, and are treated as sensors, complete with estimates of their accuracy.

[3]If you want to try it out, type `telnet monk.media.mit.edu 10891` from a machine that has a TCP/IP connection to the Internet.

2.3 Passivity

Ideally, sensors should be passive, requiring no user intervention. Many user modeling systems depend upon an interaction session with the user, and if that session doesn't occur, no user model will exist. Forcing users to describe themselves is a good way to ensure that user models are complete and accurate. Unfortunately, it is also a good way to ensure that the system remains unused.

"When people go home at night, they don't want to watch television with a joystick," says Walter Bender. They also don't want to navigate their newspaper through pull-down menus, or even log in. Keyboards and mice make entertainment into chores. People are lazy; they want to talk to their computers. Or even better, they want the computers to read their minds. Some people will want to exert control some of the time, but the perennial mistake made by designers of interactive media is this: More control isn't better if the user must always exert that control. Interactive televisions will fail if viewers can't just sit back sometimes and let the network programmers do the choosing for them.

When DOPPELGÄNGER creates a model for someone, it estimates an **annoyance parameter** for the person. The higher the value, the less the person will be expected to help DOPPELGÄNGER out in its objectives to better understand users. This annoyance parameter belies a conservative attitude on the part of DOPPELGÄNGER—if in doubt, don't bother the user. It's difficult to quantitatively measure the negative impact of intrusive sensors because it's difficult to measure intrusion, but DOPPELGÄNGER assumes that junk electronic mail is as annoying as its paper counterpart.

DOPPELGÄNGER maintains a queue of messages for each user. The lower the annoyance parameter, the more often this queue is flushed, which means that DOPPELGÄNGER will send an electronic mail message to the user telling him what DOPPELGÄNGER has done to his model recently. In the DOPPELGÄNGER server, there are three messages: `(get last message)`, which returns the latest message from the

message queue, **(get messages)**, which returns all the current messages, and **(flush messages)**, which removes the message queue. When messages are removed from the queue, they aren't lost forever. They're stored in the user model, providing a history of DOPPELGÄNGER's actions for each user.

2.4 Communities

The sensor passivity just described places DOPPELGÄNGER at a disadvantage, because passive sensors are less informative than active sensors. To compensate, DOPPELGÄNGER makes educated guesses about users based on aggregate information from user populations. If you heard about someone at MIT, you might assume certain things about them: that "he" is male, a student, knowledgeable about science, good at math, and so on. When you do this, you are using a **stereotype**. Their use is common in user modeling systems.

DOPPELGÄNGER maintains **communities**, which are like stereotypes, except that they change continuously as their constituents change, they can be generated automatically, and membership is a matter of degree rather than "yes or no." This allows DOPPELGÄNGER to make use of relationships between different user models, such as having applications behave the same for person X as it does for person Y. But more important, communities make possible *multiple inheritance* in user models: the answer to a query about a user, if not present in the user's model, can be estimated from the combination of communities which claim the user as a member. If an application asks if Orwant is interested in tennis (perhaps to ascertain whether he's interested in a local racquet sale), the knowledge needed to answer might not be contained in Orwant's user model. DOPPELGÄNGER would then analyze the communities in which Orwant is a member. The more members who like tennis in the community, the greater the confidence of DOPPELGÄNGER's answer that "Orwant does like tennis."

DOPPELGÄNGER's communities allow customization of behavior to groups of individuals. A personalized newspaper application can generate a newspaper for the student population; a scheduling application can tell when the members of some small community (say, of a lab group) will be free for a lunch meeting.

It's important for DOPPELGÄNGER to allow impatience in its users. Since there's potentially a huge amount of information about which applications might ask, a user can prime the pump by saying, "I don't want to have to train you from scratch, and I'm too impatient to wait for you to learn about me. Start me off with Walter's user model." It's important that the right parts of Walter's user model be used: his newspaper interests, but not his name, which is a reason for embedding knowledge representation into the user modeling system.

2.5 Cooperation Between Applications

Everything in your environment should be integrated: Your calendar program should tell your mail program that you're on vacation. Your car should tell your purchasing program that your tires are bald. Figure 2.2 shows communication between Doppelgängers[4], sensors[5] (represented as ovals), and applications. Thick arrows denote large information transfers.

The reason applications always fall short of good adaptive behavior is simple: they don't talk to one another. The mail program knows who you talk to and what you say, and the calendar program knows where you'll be on Tuesday. But the mail program doesn't know about the calendar program, and the calendar program doesn't know about the mail program. The friendlier personal computers integrate the

[4]The word "Doppelgänger", when in a normal font, refers to a particular Doppelgänger at some location. DOPPELGÄNGER refers to the entire research project.

[5]The word *sensor* has a broad meaning in DOPPELGÄNGER: anything that provides information about a person. A program that gathers natural language input from people is as much a sensor as an EEG electrode.

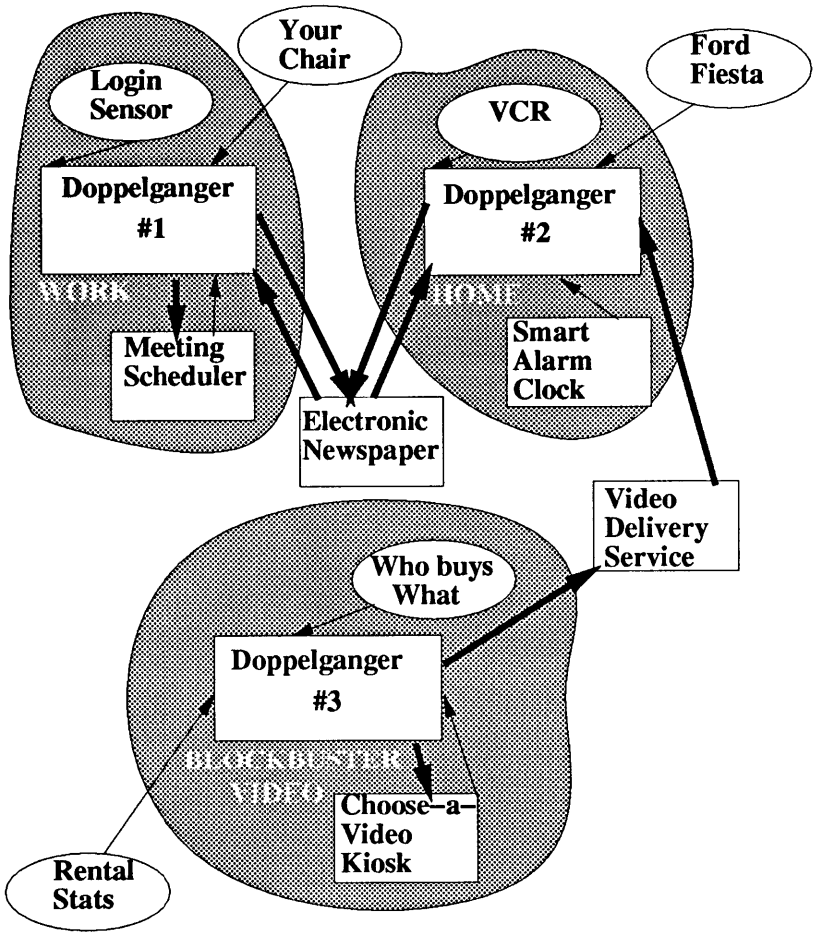

Figure 2.2: The flow of information in DOPPELGÄNGER.

applications into the operating system, so the communication between applications is richer. However, it's not rich enough, because they don't communicate in an expressive enough language. Integrating your application with the operating system isn't enough; it should know how to talk to other applications about you, and that requires a language general enough to express anything that applications might want to say. That lies dangerously close to knowledge representation, which is a bit far-out for operating systems, which need to be fast and reliable.

This places an additional burden upon the application developer: treating applications as not solely consumers of personal information, but also as providers. For instance, a personalized newspaper that makes use of the user modeling system should provide feedback so that the system can learn for subsequent newspapers, or a television that observes your reaction to its selections and modifies its behavior accordingly.

2.6 User Modeling Awareness

User modeling has a low profile. The forthcoming Apple Newton is the only marketed computer system that touts a user modeling component. In reality, this means that after you've associated "Jake" with "Jake Barnaby" once, the Newton will remember his last name for different applications. This is definitely a step in the right direction. But in spite of the Newton, people don't think about user modeling very often. People are bound by the unfortunate idea that an application is isolated: it works well, or it doesn't.

When people build new applications, they should ask themselves, "How can I make this information available and useful?" A wider variety of applications will prompt more flexible views of the field, and argue for a toolkit of general techniques for manipulating user modeling data. This will spur other people to develop and use user modeling systems.

2.7 Knowledge Representation

2.7.1 Sponge

Any system that attempts to encode aspects of user mental states requires a knowledge representation language. I wrote a knowledge representation language called SPONGE. I wrote my own, rather than using another, for three reasons. First, I wanted something fast, POSIX-compliant,[6] and able to make use of multiple computer architectures and remote procedure calls. Second, I wanted something flexible enough to handle other representations. Third, I wanted to learn more about knowledge representation, and the best way to reach this goal was to do it myself.

DOPPELGÄNGER is written in C, but the models are encoded in SPONGE. SPONGE is neither complicated nor ambitious—it's more a collection of conventions for storing data. Each data object is stored within parentheses, like a LISP list. The first element of the list is a number, called a **Sponge Tag**, that tells programs what to do with the data that follows.[7] These elements and lists are not quite LISP; they are transportable C data structures called **Dtypes**, developed by Nathan Abramson [Abr92a], [Abr92b]. Dtypes have LISP-like structure, but with the efficiency of C, portability across different architectures, and support for client-server operations.

Sponge Tags illustrate my *laissez-faire* approach to knowledge representation. The sole purpose of each tag is to tell programs what data will follow the tag. OBJECT is the most generic tag, with no constraints on what may follow. The MARKOV MODEL tag is more specific: following the tag are the numbers of states and output symbols, then a MATRIX (itself a Sponge Tag) of the Markov Model's transition probabilities, another MATRIX of the output probabilities, and optionally the names of the states and output symbols. New Sponge Tags are added all the time, whenever

[6] POSIX is a set of standards for portability across UNIX operating systems; a POSIX-compliant application is guaranteed to be portable across POSIX-compliant platforms.

[7] A listing of Sponge Tags is in Appendix C.

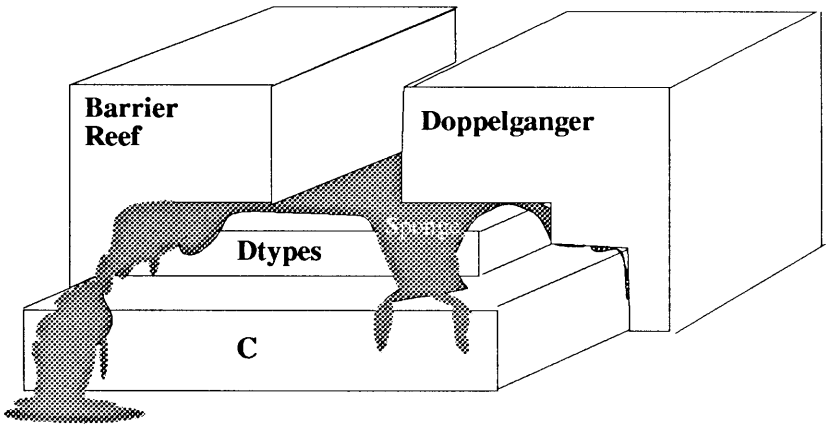

Figure 2.3: DOPPELGÄNGER building blocks.

DOPPELGÄNGER developers find it convenient. There is no rigid, fixed knowledge representation language design that limits the expressivity of code—people extend the "language" as they wish.

In this thesis, I replace Sponge Tags with their actual names to make the text more readable. The program **sponge_show** makes these substitutions.

Here's part of my model:

```
(object orwant
  (object biographical_data
    (string_binding ''true name'' ''Jon Orwant'')
    (string_binding ''e-mail address'' ''orwant@media.mit.edu'')
  ...)
  (object control
    (int_binding ''doppelganger ID'' 4))
...)
```

This is excerpted from my primary model, which is very large and doesn't change often. The primary model is one of several files that comprise the entire user model. This is elaborated in section 3.6.

The "doppelganger ID"[8] parameter in the control object indicates that this model is maintained by Doppelgänger number four, which is at a user's home. Geographically remote Doppelgängers communicate about people and communities.

2.7.2 Barrier Reef

DOPPELGÄNGER sometimes needs to know things that aren't particular to a person or community. Knowing how to manipulate information about a user's schedule requires knowledge about time zones. Knowing how to choose a topic for a personalized newspaper might require knowing what keywords comprise a boolean query representing that topic, if that were how topics were being stored. In DOPPELGÄNGER, this information is stored in a centralized *knowledge base* called BARRIER REEF. The information stored in BARRIER REEF is, like the user models themselves, encoded in SPONGE.

One BARRIER REEF convention is the use of a *units* slot to indicate how a measurement is being measured. Sometimes there is no units slot: `(integer-binding ''age'' 35)`. Age is so often expressed in years that the units slot is left out. More precisely, I expect that applications will assume that the default slot for age units is years. People do the same thing when they say that someone "is 35" rather than saying "35 years old." This is part of a loosely defined default unit set for BARRIER REEF. Ages are measured in years, time is measured in seconds since January 1, 1970,[9] and probabilities are taken to range between 0 and 1. It's BARRIER REEF's place to contain the knowledge that will allow programs to convert between units. At the moment, this knowledge is not in the knowledge base.

[8]Sadly, there can be no umlaut above the 'a' in doppelgänger. It will be a long while before my dream of umlauts in UNIX/C names will arrive. POSIX (the portable operating system specification) dictates that only 7 bit printable characters need be standard across platforms, which makes it difficult to have umlauts everywhere I'd like them in my project. However, progress is being made, thanks to GNU Emacs' support of the ISO-Latin1 standard.

[9]Because that's how UNIX does it, and you can find the difference between two time periods easily.

Information stored in BARRIER REEF need not even be written in SPONGE. There are some instances where SPONGE is too expressive (and thus too slow), or where there is another, more agreed upon format for representing data. For instance, Michelle McDonald's *Adpaper* stores *lex*[10] scripts in BARRIER REEF for parsing classified advertisements—given an advertisement, these scripts determine *e.g.* what is being sold, as well as characteristics of the item that might influence a potential buyer's decision. The better this understanding, the better the selection of advertisements in your personalized newspaper. Klee Dienes' NEWSKIT[Die93] stores *gdbm* database files in BARRIER REEF to record which articles people have read. It's futile for a knowledge base to only have one representation. It's not the way the brain works [Min86], nor is it practically efficient. Different applications will have their own ways of storing information that is best for its own use; it's arrogant of knowledge representation to purport to have a better answer. There are many good programs out there. Rather than try to develop systems that supersede them, why not make use of them instead?

Knowledge representation is an integral part of many user modeling research projects. For a better understanding of knowledge representation in user modeling, see [Kob91b].

2.8 Privacy

User modeling goes on all the time, even though it's rarely labelled as such. Credit agencies, banks, magazines, and the government all engage in user modeling to some extent. The difference in DOPPELGÄNGER is that you can see what the system knows about you, and change it.[11]

[10] *Lexical analyzers* read and convert input into a stream of tokens suitable for parsing.

[11] Sometimes people won't be able to change their models; for instance, people shouldn't be able to give themselves good credit ratings.

There are two types of privacy maintained in DOPPELGÄNGER: **model privacy** and **transaction privacy**. Model privacy means that other people should not be able to see your model if you don't want them to. More granularity here is better: you should also be able to selectively control who can see which parts of your model. Transaction privacy assumes agents that make use of your user model to act on your behalf. These agents might make purchases for you, or sell your products or information, or filter a stream of data for you. When you make these transactions, you may not want others to know their content—other people shouldn't be able to see what movies you've requested from Blockbuster.

Model privacy (data security) is part of DOPPELGÄNGER. Application privacy is supported by DOPPELGÄNGER but needs cooperation from other agents to assure people of confidentiality when agents correspond with one another.

2.8.1 Model Privacy

This is an issue of data security: how can DOPPELGÄNGER ensure that only appropriate people and applications can access information in a user's model? The first task is to figure out what constitutes "appropriate use." Users can explicitly designate who is to have access to their model, but if they do not, DOPPELGÄNGER has some default choices to fall back upon. Currently, these are liberal: let other people see the information unless the owner requests otherwise. However, in a real-world setting, the opposite would be preferable. For any part of the user model, users can choose one of four types of privacy: 1) let everyone access the data, 2) let no one access the data, 3) let only selected people access the data, and 4) let all but selected people access the data.

Imagine if Blockbuster Video stores made use of a user modeling system. This might not seem so dire; perhaps they just want to compare the number of romance rentals between June and December. But they could keep track of which movies

you're seeing, and sell that information to direct marketing firms. Or to reporters, if you're a public figure.[12]

DOPPELGÄNGER is making two changes to facilitate model privacy: it is changing filesystems, and it is adopting an authentication scheme. These are discussed in the next two sections.

Filesystems

Most UNIX systems use NFS,[13] to maintain access control over files. Unfortunately, the NFS protections are rather simple: each file or directory has an *owner* and a *group*. The owner of the file can set read, write, and execute privileges separately for all people, members of the group, and himself.

AFS[14] provides a better granularity of protections than UNIX: Access Control Lists. ACLs let individual users construct their own groups, and maintain greater control over what file operations each group may perform. This is precisely what DOPPELGÄNGER's model privacy needs.

AFS has other advantages: it allows transparent file access across other AFS cells, so I can change directories to, for instance, `/afs/andrew.cmu.edu/usr` without having to explicitly mount that filesystem. AFS also caches frequently accessed files to minimize network traffic, and supports backup servers, so that if one AFS server fails, another can transparently take its place.

Authentication

In an open network computing environment where many workstations are making use of network services, there is a danger that users and applications on a local machine will be able to falsify their identity to a remote service. There are three approaches one

[12]Whenever you give your name to some service that you suspect will place it on a mailing list, use a different middle initial. Then you can watch your name propagate from list to list...

[13]the Sun Microsystems Network File System.

[14]the Transarc Andrew File System.

can take to access control in such an environment: one can assume the local machine will prevent unauthorized access; one can require the host to prove its identity and then trust the host to authenticate the user; and one can require the user to prove his identity for each required service.

The Kerberos system [SNS88] takes the third approach. Kerberos is a *trusted third-party authentication service* that keeps a database of its clients (these can be either users or network services) and their private keys (*e.g.* encrypted passwords). Kerberos' function is to create messages which convince one client that another is really who it claims to be. Kerberos also generates temporary session keys that are given to two clients and no one else, for encrypting messages between the two clients.

When a user logs in, the authentication server uses his password, which is converted to a DES[15] key and is used to decrypt the *Kerberos ticket*, which contains identifiers for the user, server, the user's network address, a timestamp, a ticket lifetime, and a session key for the server-user pair. When the user logs out, or when the lifetime of a ticket has expired, the ticket is destroyed. As long as the software on the workstation has not been tampered with, no information exists that will allow someone else to impersonate the user beyond the lifetime of the ticket.

In the eyes of Kerberos, DOPPELGÄNGER is a network service. A server (described below) will handle all connections to DOPPELGÄNGER, whether from sensors or applications. However, in the current Kerberos system, tickets have a maximum lifetime, and the DOPPELGÄNGER server requires tickets that do not expire. Kerberos 5.0, which will be released shortly, supports infinite-duration tickets, which is what the server will use.

Doppelserve

All communication with DOPPELGÄNGER takes place through a server called **Doppelserve**. When Kerberos 5.0 is installed, the server will only provide information to

[15]Data Encryption Standard, a private-key cryptographic standard developed by the U.S. Government.

authenticated clients (which might be sensors, applications, or users running a program): those with the appropriate Kerberos tickets. The Media Lab Doppelgänger server is on monk.media.mit.edu, port 10890.

DOPPELGÄNGER is research in progress, and there is an unfortunate tradeoff between privacy enforcement and allowing DOPPELGÄNGER researchers to get their work done. Most of the code written for DOPPELGÄNGER affects not just one model, but many models, and may be dependent on the contents of the model, and even how the model changes from minute to minute. If the researcher wants to see the effects of his code, he has to be able to see the user models, which can be viewed as an invasion of privacy.

2.8.2 Transaction Privacy

It's fairly easy to ensure that reasonable safeguards protect the contents of your model. What's a bit more difficult is ensuring that actions based on your model remain private as well. One could even say that "Your phone bill will be inversely proportional to your privacy." Picture your typical user modeling application: an electronic newspaper that retrieves text, audio, and video from a central distribution site. If your computer/television/newspaper requests exactly the information it thinks you want to see, the amount of data transferred between the central distribution site and your home will be minimized. This will keep costs to a minimum as well, presuming a telecommunications infrastructure (whether cable, twisted-pair, or broadcast) that charges based on amount of usage. The other end of the spectrum maximizes privacy: your computer/television/newspaper requests a huge collection of information, possibly all that's available, and makes the selection decisions itself.

Ideally, all sales should use anonymous transactions.[16] In an anonymous transaction, the seller cannot identify the buyer other than to verify his ability to pay the selling price.

[16]The word *all* is arguable. What about guns? Drugs? Literature?

Digital signatures provide a well-established method of secure (but not anonymous) transactions [Cha92]. If I want to withdraw some money from the bank to pay for a shop purchase, I generate a large number at random and encrypt it with my private key. The bank can authenticate that I am who I say by verifying the message with my public key. Then the bank removes my signature, signs the note number with a "voucher" encrypted by its private key, and returns the note to me. Then, when I want to pay for a purchase, I transfer the note to the shop, which signs the note and forwards the note to its bank. I can't deny withdrawing the note from my account (nor spend it twice), the shop cannot deny that it received payment, and the banks cannot deny that they issued the notes or that they accepted them from the shop for deposit.

This digital signature system is secure, but it has no privacy. If the bank keeps track of note numbers, it can link each shop's deposit to the corresponding withdrawal and so determine where and when you spend your money. A good system for anonymous transactions using **blind signatures** is described in [Cha92], and these will ensure transaction privacy. The system is simple: before you send your note number to the bank for signing, you multiply it by a random factor, so that the bank doesn't know what it's signing. After receiving the blinded note signed by the bank, you divide out the blinding factor and use the note as before.

Blinded electronic bank notes protect privacy, but because each note is simply a number, it must be checked against a central list when it is spent. This is too much overhead when considered for a large number of small transactions, such as buying a newspaper. Chaum proposes a method for generating blinded notes that requires the payer to answer a random numeric query about each note when making a payment.

Chaum's system will be implemented in DOPPELGÄNGER on *portable user models*, discussed further in section 3.4.2.

There's an interesting quandary here, because DOPPELGÄNGER bases so much of its decisions upon communities. Since the characteristics of the communities are

determined by their members, it's possible to deduce characteristics of a community's members given some knowledge of the community. To prevent this "indirect" privacy intrusion, community memberships are kept private, unless the user explicitly requests otherwise.

2.9 Talking to Doppelgänger

A subsystem for parsing electronic mail called **dopmail** allows DOPPELGÄNGER to act upon messages from either users or applications. Sleator and Temperley's Link Grammar Parser[ST91] is used to parse the input syntax. The result is that you can send mail to DOPPELGÄNGER containing sentences such as "I like technology" or "I am working at home," and your user model will be affected accordingly. The "Subject:" line of the mail message helps dopmail disambiguate the appropriate context for interpreting the message. For instance, a subject line of "News" tells dopmail that the message is feedback about the user's news environment, a subject line of "Location" tells dopmail that the message is information about the user's whereabouts, and a subject line of "Interdoppel communication" tells dopmail that the message is communication from one Doppelgänger to another.

Simple English has its advantages as a protocol for communicating with a Doppelgänger. People don't have to remember some awkward syntax or program name, and the language is guaranteed to be expressive enough for anything a sensor might want to say to a Doppelgänger. The drawback is that you have to parse it. Communication between DOPPELGÄNGER modules is required to be a "subset" of English: only certain sentence forms are recognized. Using English means that there's a guarantee of extensibility: as the messages between sensors and DOPPELGÄNGER, and DOPPELGÄNGER and applications, grow in complexity, more complex sentences will be required. The structure for those sentences already exists; we use it every day when we talk. Furthermore, different Doppelgängers might maintain different levels

of parsing ability. A future enhancement might allow Doppelgängers to trade information about how to parse English. When full natural language parsers become available, it will be a small step for DOPPELGÄNGER to make use of them.

User modeling researchers are well aware of the strong connections between the field of natural language processing and user modeling. In the next chapter, I explore some other connections inspired by the field of machine learning.

Chapter 3

New Ideas for User Modeling

The ideas I've mentioned already are by no means widespread among user modeling systems. In this chapter, I go further and make some broad observations about how machine learning can help user modeling.

Much of user modeling is concerned with ensuring that the requisite structures for modeling cognition are in place, revealing the field's roots in AI. This has happened to such a degree that new creative directions have been stifled. User modeling is a young field, and can benefit from some rethinking. In addition to the recommendations I made in the previous chapter, there are a body of techniques (methods and algorithms) and ideas (approaches to thinking about problems) from machine learning that benefit user modeling.

The field of *machine learning* concerns itself with how computers can be made to exhibit adaptive behvaior in an unknown environment. Machine learning can certainly be used to improve the performance of sensors. This is more common sense than anything else: sensors, like most anything else, benefit when they adapt to their environment. Speeter's smart floor system [Spe90] is an example: floor tiles containing piezoelectric sheets rapidly sample the location and amount of pressure upon them. Then time series analysis is used to identify people by their weight/time

profiles. On my home workstation, I have a neural net that decides when to call in and retrieve my electronic mail: it adapts to my behavior (working hours, amount of mail, phone line usage) over time. Besides making sensors more accurate, learning techniques can also be used to help sensors become more passive. A sensor might estimate how much it distracts the user, or its noise level, or how much of the user's time it's taking, and adapt to minimize the burden on the user.

3.1 How Machines Learn

3.1.1 Formal Models of Machine Learning

Why should machine learning models be useful to user modeling?[1] Because user modeling tries to make computers understand and react toward a very complex phenomenon—the user.

An overly rigorous tack comes from behaviorism: treat the user as a complex system, for which the computer must learn to predict the output given the input. Such an approach would employ control theory and signal processing exclusively [Lju87]. A better approach is to treat the user as a set of hundreds of complex systems, each of which can be modeled to varying degrees.

Machine learning consists of a body of techniques for making adaptive changes to a computer's representation of some domain. There are four broad types of machine learning: *rote learning*, in which presented information needs to be memorized; *learning by being told*, in which information is given to the learner in the form of abstract

[1]The dual use of the word "model" to mean both a template for theoretical work and to mean a collection of data about a person is unfortunate; if the context in my text is ambiguous, I probably intend the first meaning, as I use "user model" for the second. Cheeseman [Che90] defines a model as "any formal description (mathematical or logical) that assigns probabilities (including 1 or 0) to experimental observable outcomes. The term...can be interchanged with *assumptions*, *hypotheses*, *concept*, or *theory*; the important property these terms share is the ability to make (probabilistic) predictions about possible observations."

advice, which needs to be integrated with the learner's current knowledge and specialized for particular situations; *learning from examples*, in which an environment contains specific examples which must be generalized to form more abstract concepts or rules; and *learning by analogy*, in which the learner must discover the similarity between trial situations and future situations [MCM86], [Car89].

There is a need not only for having machine learning strategies available but for guidelines deciding which ones to use and when to use them. Machine learning techniques are developed without regard to their eventual use, because the techniques are general enough to be applicable to innumerable tasks. The job of the user modeling researcher should be to identify new tasks make possible by machine learning that will aid in modeling the user.

User modeling, as a discipline, has not matured enough for a more rigorous treatment. In part, I hold this belief because my philosophy of user modeling is somewhat at odds with most researchers in the field. User modeling research rarely users machine learning. A counterexample is Webb's intelligent tutoring system, described in [Web91], which uses feature extraction for student modeling.

Machine learning develops results that tell us what can and can't be learned, how well, and what methods are appropriate. Research targeted toward specific applications is not generally considered part of the primary body of literature.

3.1.2 Components of Machine Learning Paradigms

A *machine learning paradigm*[2] has five components: a learner, a domain, a source of information about the domain, some prior knowledge, and a performance criterion.

[2]The phrase is mine, but machine learning researchers would figure it out by context. As far as I can tell, that's all the word *paradigm* really means—"there's no good word for this thing, so you figure it out by context."

Learner

Our learner is the user modeling system. It is usually an actual software system; in Doppelgänger's case, it consists of approximately 20,000 lines of C code. It runs continuously, adding data to the population of users that it models, and occasionally "forgetting" data that hasn't proven useful enough.

Domain

The domain, framed in the broadest possible terms, is the cognitive and physical state of people. This includes both the states of individuals and of larger groupings of these individuals.

To make this more tractable, we can take the following steps:

- Break the problem into smaller domains, and deal with the specific rather than the theoretical. If this step isn't taken we run the risk of being overly abstract. It's much easier to address the problem of "Here is a source of information, and here is our interface to it. What can we learn from it?" than to address properties common to all interfaces and all information sources. Otherwise, user modeling becomes system theory[3][Wei72] with a touch of behaviorism thrown in.

  Once this is done, the usefulness of machine learning becomes more apparent. There are many sources of information that could constitute input to a user modeling system. Most of these involve trying to induce rules given some sample data collected from a user, and hence are good candidates for machine learning. (A notable exception is natural language processing, which might have been considered part of machine learning had it not been so complex in its own right.)

[3]Weinberg, in Trends in General Systems Theory, wrote: "Writing about general systems theory is like writing about unicorns. Even after you eliminate all the people who have never heard of them and all the people who have heard of them but are not interested in the subject, you have to contend with the people who do not believe that they exist."

- Model a small subset of all people.
- Allow for some "slop" by ignoring data that will not foreseeably be used by applications making use of the user modeling system.

DOPPELGÄNGER's user modeling data can be divided into three granularities of data. In order of refinement, they are: the raw data, the model, and the entire population modeled by the system. Raw data is an anonymous series of observations, which are then incorporated into the second domain, user models, which are then used to establish the third domain, communities of models within a population. The data in each of these domains is refined over time as more data is gathered. Thus, these tasks are good candidates for machine learning.

Source of Information

Like the robot-in-an-environment paradigm, the user modeling system has various sensors that gather information about the world. There is no one source of information; instead, there are a multitude of sensors, which might be hardware or software, or gather time-dependent information, or perform arbitrarily sophisticated processing on its data before relaying it to the user modeling system, or gather ordinal, cardinal, or qualitative information.

In *A Wattled Theory of System* [Wym72], Wymore introduces the concept of a *wattled* system. A wattled system is one that consists of qualitatively different subsystems; most notably, a system that contains both data that is best modeled by a discrete finite automaton and other data that is best modeled by a differential equation. People are clearly wattled systems: a person's address, or how he voted in the last election, are undeniably discrete quantities; his trajectory as he runs his errands or makes a gesture, are continuous functions. This poses problems for algorithms that expect data of the same type. DOPPELGÄNGER's resolution to this dilemma is in section 4.4.1.

Prior Knowledge

Making generalizations about subsets of people, communities, is an invaluable computational aid for filling in holes in user models. Indeed, many user modeling systems have hardcoded stereotypes that consist of rules of the form, "Professors do not make much money." The authors of these systems make no claims that these stereotypes are always correct; however, criticism is inevitable. Prior knowledge, when applied to people, is usually termed "prejudice."

The Guides project [OSKD90] at Apple Computer attempted some simple user modeling. Its goal was laudable: to enable a user to explore the topic of 19th century American westward expansion from multiple perspectives, such as a pioneer's wife or an Indian. However, the Guides project has been criticized by some because it claims (though only in the loosest possible sense) to capture a broader perspective that it cannot hope to do justice to.

Doppelgänger's "start state" is *tabula rasa*—it begins operation with no previous knowledge of anyone. However, it is always running, and thus experiences the ignorance of its start state only once. Over time, as it gathers more and more knowledge, those rules it infers that have been successful predictors become prior knowledge of sorts.

Performance Criteria

In the domain of user modeling, it will often be difficult to assign performance criteria. This might be because it will be hard to tell what the target concept is, but it is also because it is impossible to say when the system has stopped learning, since it never stops learning. In any event, most user modeling has no performance criteria because it is concerned with such tasks as determining when some set of beliefs logically yields another belief. It's hard to assign performance criteria to such an abstract task.

Typically, work in machine learning assumes a domain much more carefully defined than the domain for user modeling described above. Two common formal models of

machine learning with different performance criteria are *mistake bound* and *PAC learning* [Blu90]. Both of these models address learning some concept from a set of concepts.

The mistake bound model is the simpler of the two: examples are presented to the learner, which classifies each example as it is presented, and then is told whether its classification was correct. The success of the algorithm is measured by the number of classification mistakes.

The PAC[4] model of machine learning assumes the existence of an *oracle* which will produce a classified example on demand. PAC learning algorithms use the oracle to hypothesize a target concept. The concept so chosen is guaranteed with some probability $1 - \delta$ to be accurate to some degree ϵ, where δ and ϵ are parameters of the algorithm. PAC learning is *distribution-free*—the results proved are independent of the input probability distributions [KV93].

What criteria are appropriate for user modeling? Suppose the user modeling system, in conjunction with a personalized newspaper application, decides that a person will like a certain article, and presents it to him. However, the user reads only the headline and moves on—he has decided the article is uninteresting. Clearly, the system has failed, but why? Perhaps the article was chosen because it mentions the user's hometown, but was skipped because the user was more interested in something else, or in a hurry, or was simply experimenting with this new-fangled personalized newspaper. In these cases, the system's reasoning about the user's hometown was not incorrect; it merely failed to model the user thoroughly enough to account for the vagaries in his reading style. *Predictive success* is the only available means by which the system can evaluate itself, which is unfortunate because this analysis is ill-formed.

In the following sections, I describe ideas prompted by my thinking about machine learning paradigms.

[4]Probably Approximately Correct

3.2 If You Don't Know, Ask

Query-based learning is a machine learning paradigm that assumes an oracle to which the learner can direct occasional questions. The application to user modeling seems obvious: if DOPPELGÄNGER needs to know something about a user, it asks him, by sending him mail. Strangely, this has not been used in user modeling systems to date.

When applications request information from DOPPELGÄNGER, they should do so with an importance value. The higher the importance value, the more likely DOPPELGÄNGER will directly go and ask the user. This is another advantage to having a natural language protocol—not only is it good for programs talking to DOPPELGÄNGER, it also works well when DOPPELGÄNGER needs to communicate with its users.

The most common means of communication between DOPPELGÄNGER and users is simple English through electronic mail. How often it does this depends on the *annoyance parameter* in the user model.

3.3 Sensor Arbitration

Sometimes DOPPELGÄNGER's sensors will disagree with one another. The badge sensor might disagree with the login sensor. Chances are that the badge sensor is right; it's more accurate than the login sensor. DOPPELGÄNGER could encode this into a heuristic: "if the badge sensor and the login sensor conflict, favor the badge sensor." But this is only a temporary solution. DOPPELGÄNGER potentially has a huge number of sensors. More are added all the time, and the sensors will get better over time as their developers add features and fix bugs. Applications can provide feedback not just about the user, but about the quality of the information provided by DOPPELGÄNGER, which DOPPELGÄNGER can then use to maintain its own estimates

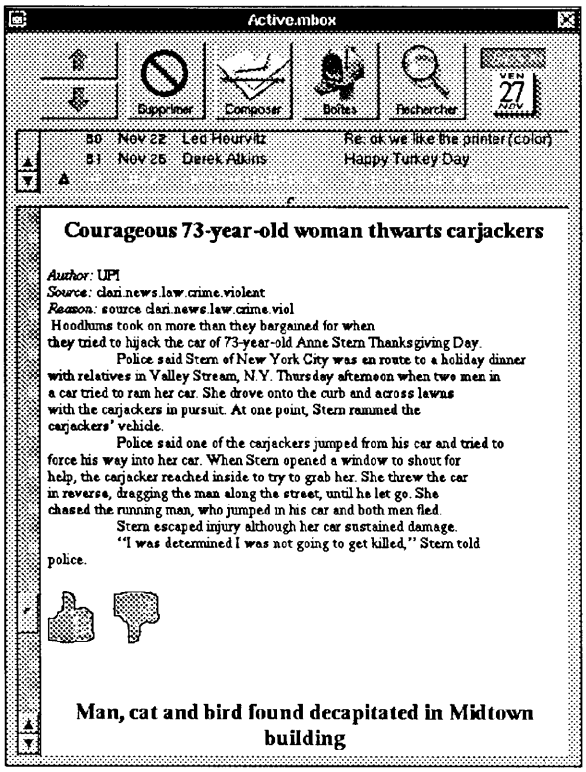

Figure 3.1: FeedbackPaper: A newspaper application using DOPPELGÄNGER.

of sensor accuracy. These change over time as feedback from applications agrees or disagrees with the sensor.

Sometimes a sensor will disagree with itself. A personalized newspaper application written by Doug Koen called FeedbackPaper[CK93] (see Figure 3.1) presents two icons with every article. If the reader clicks on the thumbs-up icon, it extracts keywords and sends a message to DOPPELGÄNGER, saying "I like <keyword>" with a subject line of "News." DOPPELGÄNGER then modifies the reader's model accordingly. If the person clicks on the thumbs-down icon, the same actions are taken, but with "like" replaced by "hate." DOPPELGÄNGER now has the problem of trying to estimate how much a user likes the "technology" topic based on conflicting input: the user will like some articles and dislike others.

Consider a coin that has some bias p (If $p = 1$, the coin always comes up heads, $p = 0$, tails, and $p = .6$ means that the coin comes up heads six times out of ten). You

wish to determine the bias of the coin. How do you do it? I have DOPPELGÄNGER use the **Beta distribution** for problems like these. The Beta distribution [Dra67] allows one to model the mean and variance of a random variable from a sequence of one-bit inputs, *e.g.* a series of heads and tails. If the number of heads is h, the number of tails is t, then our estimate of the bias of the coin is simply $\frac{h}{h+t}$, and the variance of the distribution is: $\sigma^2 = \frac{ht}{(h+t)^2(h+t+1)}$. The higher the variance, the lower the confidence of DOPPELGÄNGER in the assertion. An estimate of the amount of information is $\frac{\sigma^2(p)}{maxvar(p)}$, where I define $maxvar(p)$ as $\frac{1}{4(h+t+1)}$. $maxvar(p)$ is the maximum variance that p could have given the number of observations. This will occur when $h = t = \frac{h+t}{2}$, so the maximum variance is: $\frac{(\frac{h+t}{2})^2}{(\frac{h+t}{2}+\frac{h+t}{2})^2(h+t+1)} = \frac{1}{4(h+t+1)}$.

Input to dopmail can conceivably get much more complicated than simple barrages of "I like ..." or "I hate ..." dopmail can also parse sentences of the form "I like sports more than finance in my newspaper." This is tricky, because the sentence is a statement about the relation of one mean-variance estimate to another. This is more difficult than it might seem: there might be multiple comparators (*e.g.* "I like sports more than finance" and "I like sports more than environmental issues"). There might be a sequence of comparators that violate transitivity: "I like sports more than finance," "I like finance more than environmental issues," "I like environmental issues more than sports." And what should DOPPELGÄNGER do if it has a weak estimate for the finance topic, but a strong estimate for the belief that the user likes sports more than finance?

This is where the information estimate of $\frac{var(p)}{maxvar(p)}$ comes in handy. "Like more" and "like less" are preset to mean ten percent more and ten percent less. This is an arbitrary value that might change over time, although DOPPELGÄNGER does not currently support this adaptation. The variance of the left side of the comparator is then set to be $1.1\frac{var(p)}{maxvar(p)}$.

These intermediate calculations are not stored, because each new bit of information can affect all the others in the group. Instead, they are computed on the fly. This

highlights an important point in client-server architectures like DOPPELGÄNGER: the speed of the reply. On occasion, answers from DOPPELGÄNGER that can't be calculated beforehand will need to returned quickly. Ideally, requests by applications will include an **urgency** value that fixes a compromise between the accuracy and speed of the answer.

3.4 Distributed User Modeling

It would be nice if there were no centralized user model—when an application wanted information about the user, it would broadcast its request to the "ether", and answers would come pouring in from the multitude of sensors in the environment. The broadcasts described would happen thousands of times per second. Furthermore, sophisticated reasoning upon sensor data, as well as computation based on a history of behavior, both require mass storage, and therefore centralization as well. DOPPELGÄNGER is *centralized.* To be precise, it consists of independent Doppelgängers, each centralized. It has to be, because it collects a lot of information about a lot of people from a lot of sources, supports multiple inheritance based on communities, and makes its conclusions available for a lot of applications.

Centralization has to end somewhere, though—you can't have one Doppelgänger spanning the globe, modeling all people. That would be risky (the Doppelgänger might break), awkward (there would be a message bottleneck), and totalitarian (if you control that Doppelgänger, you control everything). These deficits manifest themselves less and less as the scope of each Doppelgänger shrinks, culminating in the other extreme: a maximally wired world where everything has its own Doppelgänger, and they all communicate with each other all the time.

Many Doppelgängers will remain private, because their users will choose to compromise the power of the system so that their privacy will remain uncompromised.

```
From: doppel@home.media.mit.edu <Doppelganger at Orwant's home>
To: doppel@media.mit.edu <Doppelganger at the MIT Media Lab>
Subject: Interdoppel communication

Partial Schedule Model for Jon Orwant:

(object orwant
  (object schedule
    (object alarm_clock
      (time ''6 August 1993 830'')
      (action ''turned off''))
    (object state
      (descriptor ''late'')
      (reasoning alarm_clock))))
```

Figure 3.2: Interdoppel communication: a partial schedule model.

A recent topic of interest among machine learning researchers is *distributed machine learning* [The89], in which sources of information about a phenomenon are distributed, meaning that there is some significant cost involved in sharing information about the phenomenon. Weather prediction involves making decisions (*e.g.* cold air mass from the east) based on information from distributed locations. Surveillance schemes might use spatially separated sensors to observe a common object.

The medium for communication between Doppelgängers is electronic mail, and the language is English and SPONGE. A sample message from one Doppelgänger to another is shown in Figure 3.2. My Doppelgänger at home initiates this mail and sends it to the *doppel* account at the Media Lab. The Media Lab Doppelgänger then assimilates this information into its user model for Orwant. It might choose to disregard parts of the message, or to ignore the message entirely, because of what it already knows about Orwant, or because my Doppelgänger at home has provided consistently incorrect information.

3.4.1 Synchronizing Home and Work Environments

One of the best reasons to employ communication between distributed Doppelgängers is to ensure that when you move from place to place, your computational environment moves with you.

I've written some Elisp code[5] that sends whatever files I'm editing at home to work, and vice versa, because often I'll wish to view or edit the same files at home as at work, but don't want to take the trouble to log in, download the files, and so on.[6] I'm using the application, which I call **dopsync**, to write this thesis. It ensures that when I edit a file at home, the corresponding file at work is updated, and vice versa. dopsync maintains security by verifying the mail author,[7] only updating portions of the filesystem that will not damage system operations, and saving backups just to be sure. DOPPELGÄNGER stores information about the address of my home and work machines. BARRIER REEF will store information helping applications guess what I want transferred; right now this is stored in the application.

In essence, this is a crude virtual filesystem, suitable for very low bandwidth communication, and illustrates yet another example of the benefits of integrating the user modeling system with the operating system.

3.4.2 The Portable User Model

There will be times when a user will want to have manual control over his user model. Ideally, he should be able to carry his user model with him. He might want to bring his user model to a newspaper machine so that it can print out his newspaper, or bring it along when he travels so his computational environment can be customized, or as a carrier for his medical records should he become injured.

---

[5] Elisp is a dialect of Lisp bundled with the text editor Emacs.

[6] I have a UUCP (*U*nix to *U*nix *CoPy*) link between my home computer and the Media Lab, which transfers mail every twenty minutes.

[7] This is futile, since it's easy to flawlessly forge electronic mail.

Two classes of portable user models will be considered here. The first is just a copy of the user model: these could in theory be stored on magnetic strips, like credit cards. The second contains an onboard processor in addition to the storage.

Dumb Smart Cards

Portable user modeling devices could be stored on magnetic strips, like credit cards. These would allow you to swipe your card through a card reader to log in and have the computer set up your environment.

The PCMCIA[8] standard is an emerging design standard for credit card-sized adapters [Int93]. PCMCIA *smart cards* are small, thin, flash memory cards that can store up to forty megabytes. They're about the size of a thick credit card, and can be used to transfer data between any two computers that have PCMCIA interfaces.

Gillian Lee is working on storing DOPPELGÄNGER user models on PCMCIA cards. Carrying your user model in your pocket affords both psychological comfort and ease of use. The MIT Media Lab Electronic Publishing Group has a newspaper "point-of-sale" box (one of the metal boxes on street corners that dispenses newspapers). There are some crucial differences between this box and its urban cousins, however: it has a printer inside, and a PCMCIA interface on top attached to an IBM RISC System/6000 workstation. When the user slips his smart card into the PCMCIA interface, his personal newspaper is printed for him on the spot.

Smart Smart Cards

As memory, storage, and CPUs get smaller and faster, the feasibility of a computation on smart cards in general will increase. A smart card with an onboard processor would be even better. Portable computers like the Apple Newton is almost small enough, and certainly possesses the necessary computational power for limited user modeling

[8]Personal Computer Memory Card International Association, an association of over 300 companies created in 1989 to set standards for mobile computers.

as well as a PCMCIA interface. The Digicash card [Cha92] fulfills the transaction aspects, but that's all: there's no way to extend its capabilities to deal with a user model.

In the Digicash system, and in the plans for DOPPELGÄNGER, each card contains an embedded observer in addition to its own microprocessor. The representative and the observer generate numbers that the observer uses to produce a set of blinded digital pseudonyms. The observer signs the pseudonyms with a special built-in key. The representative checks the pseudonyms to make sure they do not disclose any illicit information and passes them to a validating authority. The validating authority checks the observer's special key, removes it and attaches its own unforgeable signature. The representative removes the blinding from the validated pseudonyms and stores them for future use by the observer. By requiring a PIN, the smart card could safeguard itself from abuse by thieves.

Chaum observes that this system can also be used for not just money, but credentials as well: a job applicant could authorize a university to verify his degree (but not his field); potential employers could send requests to validate his résumé, and the university would verify only what the applicant allowed.

3.5 Computational Bigotry, or Prior Knowledge is Prejudice

If a flying saucer landed in Times Square, and green-skinned octopeds emerged, you'd probably assume that this alien race consists of green-skinned octopeds, and not blue-skinned octopeds that spent too much time beneath their planet's thinning methane layer. When the blue-skinned octopeds arrived, one by one, you might assume that they were the anomalies, until you saw enough of them. If you were stupid, you might cling to your outdated belief for longer than logic would dictate, because beliefs have

persistency. And with DOPPELGÄNGER, the first people to influence a community can have a marked effect on what the community becomes.

This is a good example of why people should be able to interact directly with their user models. When they do so, all of a sudden DOPPELGÄNGER has a sensor that's guaranteed to be correct (presuming people know themselves well), which can help to accelerate recovery from mistaken assumptions that have outlived their usefulness.

3.6 Conditional Models

A user model in DOPPELGÄNGER is an NFS (soon to be AFS) directory, consisting of a primary model, *submodels*, and backup models. All are encoded in SPONGE.

I recognized that keeping all of the information in one file is awkward, because many processes will want to access any given user model at the same time. The large **primary model** contains biographical information; it is often read by applications, but is rarely modified. The directory also contains **submodels**, which can be either **domain models** or **conditional models**. Each domain model contains information about a certain aspect of a person's life, such as his location. Conditional models are "special case" models that are only true when some condition is satisfied in the outside world. People who want a different newspaper in the morning than in the evening might have a conditional model for each: in the morning I'm interested in hard news, while in the evening I like to read more about recreation.

The domain submodels are listed in Figure 3.6. I've chosen them based on how often the data will be modified and how much data will need to be accessed by sensors and applications. But DOPPELGÄNGER could just as well figure this information out for itself, if I were to encode some rules for determining when a submodel should be split off. Doppelgänger automatically generates conditional models for users based on templates contained in BARRIER REEF.

- primary: biographical data, and other data that does not change often. This file is large; file writes take a long time, which is why data that does not change very often is stored here.
- location: where the user is.
- schedule: events in the user's life.
- newspaper: what the user wants to see in his newspaper.
- login: past and predicted future login times.
- commands: UNIX commands that have been executed.
- ads: purchases that user might make.
- questionnaire: information that the user has explicitly provided for DOPPELGÄNGER.
- message: new messages from DOPPELGÄNGER to the user.
- message-archive: old messages from DOPPELGÄNGER to the user.

Figure 3.3: Domain submodels.

Communities can also possess domain and conditional models. DOPPELGÄNGER maintains news submodels, so one can request the Media Lab Community newspaper. Some domain submodels have a drastically altered meaning when applied to a community rather than an individual. For instance, an individual's location model is straightforward: where the person is, and predictions about where the person is going. But a community might be *defined* by a particular location (*e.g.* residents of Louisville, KY). Or the community might be a business group that has meetings in certain rooms—at those times, the community has a location, but at other times the community will have no location at all.

In the next chapter, I'll move from the broad to the specific, and describe three specific learning methods that I've used for user modeling tasks.

Chapter 4

Three Techniques and Four Tasks

While the previous chapter made general arguments for new approaches to user modeling, this section explores three specific applications of machine learning techniques for particular user modeling tasks.

Evaluation for user modeling tasks is often difficult, because of the nature of the phenomena: users change. And they change over many time scales: minutes, days, months, and years.

I've chosen my tasks to span the three different granularities of data in DOPPELGÄNGER—the raw data, the model, and the community. The tasks and their domains are depicted in Figure 4.1.

4.1 Location Prediction

DOPPELGÄNGER has three ways of sensing user locations: a sensor that keeps track of which computer consoles someone is using (this provides information about his physical location), a schedule sensor that given a calendar file deduces the locales to which a user has flown, and Active Badges. These are badges that emit infrared bit patterns, which are used to track people as they walk around the Media Lab. In the

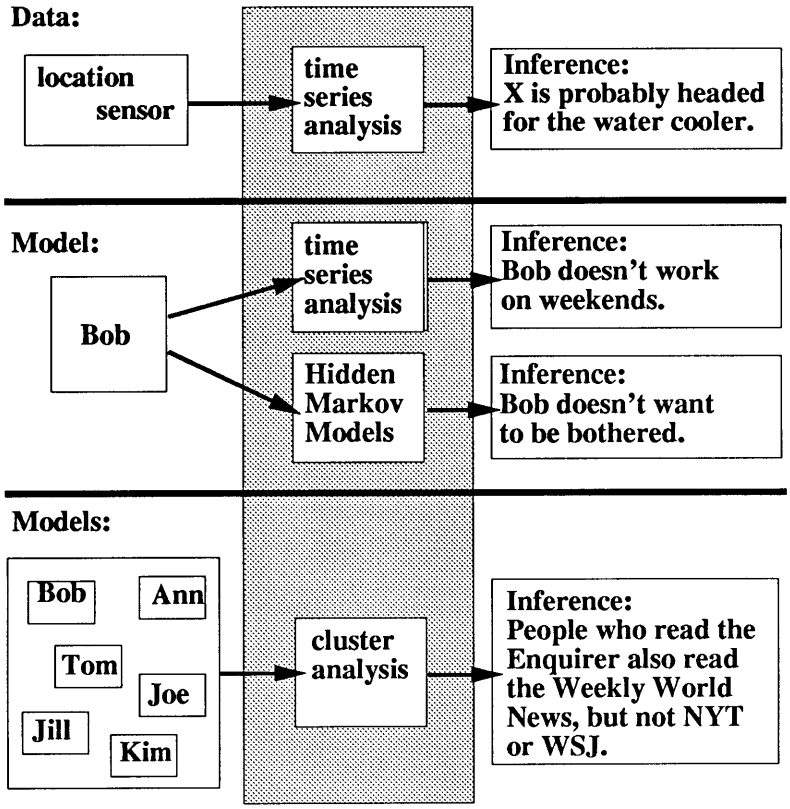

Figure 4.1: Four applications of machine learning to user modeling

future DOPPELGÄNGER may be using the smart floors described in [Spe90] to track people. I used Hidden Markov Models to predict where people would walk, given their location data.

Part of Walter Bender's location submodel, using data gathered from the badge sensor, is shown in Figure 4.2. The rooms Walter was sighted in during the sample period of an hour are listed on the next page.

"E15 3" is the Media Lab, third floor. Rooms with a 'c' as the middle character are actually corridors. Here is the sequence of observations:

1 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 4 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1

1 1 1 5 5 5 5 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 2 2 2 2 2 6 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 7 7 7 7 7 7 7 7 7 7 7 1 1 1 1 1 1 1 1 1 4 4 4 4 1 1

The table transition probabilities generated by DOPPELGÄNGERare shown on the next page.

```
(object walter
  (object location
    (time-binding ''E15 3 344''
      (time 746346881))
    (time-binding ''E15 3 3c2''
      (time 746346888))
    ...))
```

Figure 4.2: Part of a location submodel.

```
Room 1: E15 3 344
Room 2: E15 3 3c2
Room 3: E15 3 356
Room 4: E15 3 3c1
Room 5: E15 3 343
Room 6: E15 3 3c3
Room 7: E15 3 350
```

```
0.909091 0.022727 0.011364 0.034091 0.011364 0.000000 0.011364
0.100000 0.700000 0.100000 0.000000 0.000000 0.100000 0.000000
0.033898 0.000000 0.966102 0.000000 0.000000 0.000000 0.000000
0.333333 0.166667 0.000000 0.500000 0.000000 0.000000 0.000000
0.200000 0.000000 0.000000 0.000000 0.800000 0.000000 0.000000
1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.100000 0.000000 0.000000 0.000000 0.000000 0.000000 0.900000
```

Given what room a person is in, the Markov Model generated by DOPPELGÄNGER makes it easy to predict what room he'll enter next, as well as the accuracy of that estimate.

The algorithm used to generate these transition probabilites is very simple: there is a one-to-one correspondence between rooms and states, which means the output probability matrix will be the identity matrix—state i has probability 1.0 of producing output symbol (*i.e.* room observation) i and probability 0.0 of producing any other output symbol. In a sequence of n observations, there are $n - 1$ transitions between states. The algorithm sets the transition probability between state i and state j to $\frac{\textit{number of transitions between } i \textit{ and } j}{\textit{number of transitions between } i \textit{ and any other state}}$.

How accurate was location_observer? This depends on the amount of data gathered. Clearly, with only one data point, location_observer has too little information with which to accurately predict the next value. If the data points span a week, they'll pick up daily cycles but not weekly patterns. And if the data points span only a year, they'll fail to pick up yearly cycles.

This could be used for more than just user modeling. Imagine a Global Positioning System.[1] The information relayed by a GPS (presumably by broadcast) could be analyzed by DOPPELGÄNGER; the results would indicate which areas connect, as well as which are corridors and which are rooms. DOPPELGÄNGER could perform simplistic mapping of a remote site in this way.

Unfortunately, DOPPELGÄNGER's badge sensor needs work. It spawns a process every minute to check the Badgeserver, which is too CPU-intensive, slowing down its workstation too much. It needs to be made event-driven: integrated into the Badgeserver code so that it doesn't take so many CPU cycles. Also beneficial would be a visual interface to DOPPELGÄNGER's results: I'm writing a system for generating pictures of the Markov Model describing user location behavior.

[1]These are small devices that can determine their location (represented as latitude and longitude) through a system of twenty-four triangulating geosynchronous satellites.

Spectral analysis on the data showed up some simple daily and weekly patterns: people don't work on weekends. As the data-gathering continues, I suspect further patterns will reveal themselves.

Because the Markov Model states are fixed, DOPPELGÄNGER's task is simplified considerably—it doesn't have to figure out what the states are. If it did, it would use a more sophisticated approach: the Baum-Welch reestimation procedure [The89]. By removing the restriction that each state is a room, DOPPELGÄNGER could postulate its own states. One state might be "working in the Media Lab Garden," which would consist of probabilities of being sighted at the different sensors around the Garden. Sensors closer to the Garden center would have higher output probabilities in that state, and the probabilities would fall off (perhaps according to an inverse square law) as distance from the Garden center increased. Another inferred state might be "going to the bathroom," which would be a pattern of states leading down the appropriate corridors. Luckily, there are no Badger sensors in the bathrooms themselves.

4.2 Environment Analysis

The location prediction task *generates* Markov Models. It does not attempt to pick one of many "Hidden" Markov Models that best describes some observed behavior. This is how many important problems in speech recognition (and many other fields) are framed: choosing a model (*e.g.* the speaker said "cat") that maximizes the probability that the observed sequence of output symbols would be generated. A well known algorithm for efficiently solving this problem is the Viterbe algorithm [Rab89]. I implemented the Viterbe algorithm for the task of *environment analysis*: identifying what state a person is on based on his interactions with a computer.

Figure 4.3 shows the *hacking* Hidden Markov Model. Each model consists of states (the *compile* state is listed in figure 4.4) and probabilistic transitions between them. The eight models are listed in Figure 4.5.

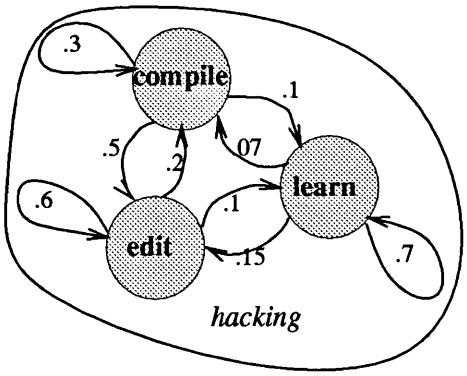

Figure 4.3: Hidden Markov Models for environment analysis: the hacking model.

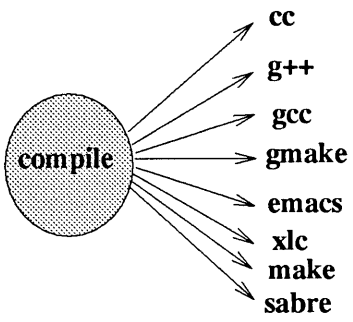

Figure 4.4: Hidden Markov Models for environment analysis: the compile state.

- hacking
- idle
- frustrated
- writing
- information and entertainment *i.e. news and games*
- concentrating
- image processing *(a common task among the people modeled by Doppelgänger #1.)*
- connection *(i.e. using the computer as a gateway to another.)*

Figure 4.5: Hidden Markov Models for environment analysis: models.

- compile
- edit
- mail
- shell
- window manager
- text format
- games
- image processing
- user modeling
- connection
- news
- view
- learn

Figure 4.6: Hidden Markov Models for environment analysis: states.

The fourteen states out of which the models are constructed are listed in Figure 4.6.

4.3 Login Prediction

One of DOPPELGÄNGER's sensors relays information about logins to DOPPELGÄNGER—when people log in to a computer, and the duration of the session. These are time-based, sequential observations of real values—a natural candidate for time series analysis [And71], which means using a sequence of observations to predict future observations. I used linear predictive coding [PTVF92] to predict the time and duration of the next login. The reason for using linear predictive coding rather than some other

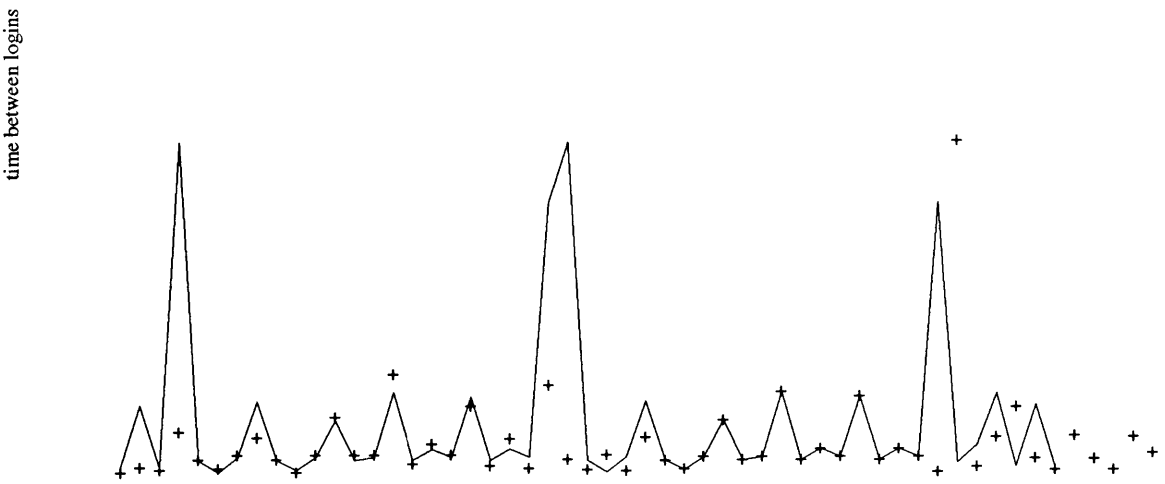

Figure 4.7: DOPPELGÄNGER's interarrival predictions

method of regression analysis is because of its utility for predicting cyclical behavior, as opposed to smooth movement increasing or decreasing with time. Linear predictive coding uses spectral analysis, which identifies periodicities. One would expect periodicities to occur on daily and weekly scales; they do.

I analyzed the times between logins, or *interarrival* times, rather than the actual times of occurrence, because interarrival times remove drifts in the data. Using interarrival times, If the person starts his day half an hour later, then only one data point is affected, rather than all the points for the day.

In addition to predicting the interarrival times for logins, the same method can be used to predict login durations: how long a user stays logged in. Only login durations greater than four minutes were included, on the basis that this was a good threshold for "meaningful" logins: logins for a shorter duration might be FTP transfers, or accidental logins. Each login duration is rounded to the next lower minute.

4.4 Cluster Analysis for Community Inference

User modeling stereotypes have come under fire for reflecting the bias of the user modeling system designer. Ideally, the user modeling system should generate its own communities. Then, any claims of bias are weakened, because the designer's influence

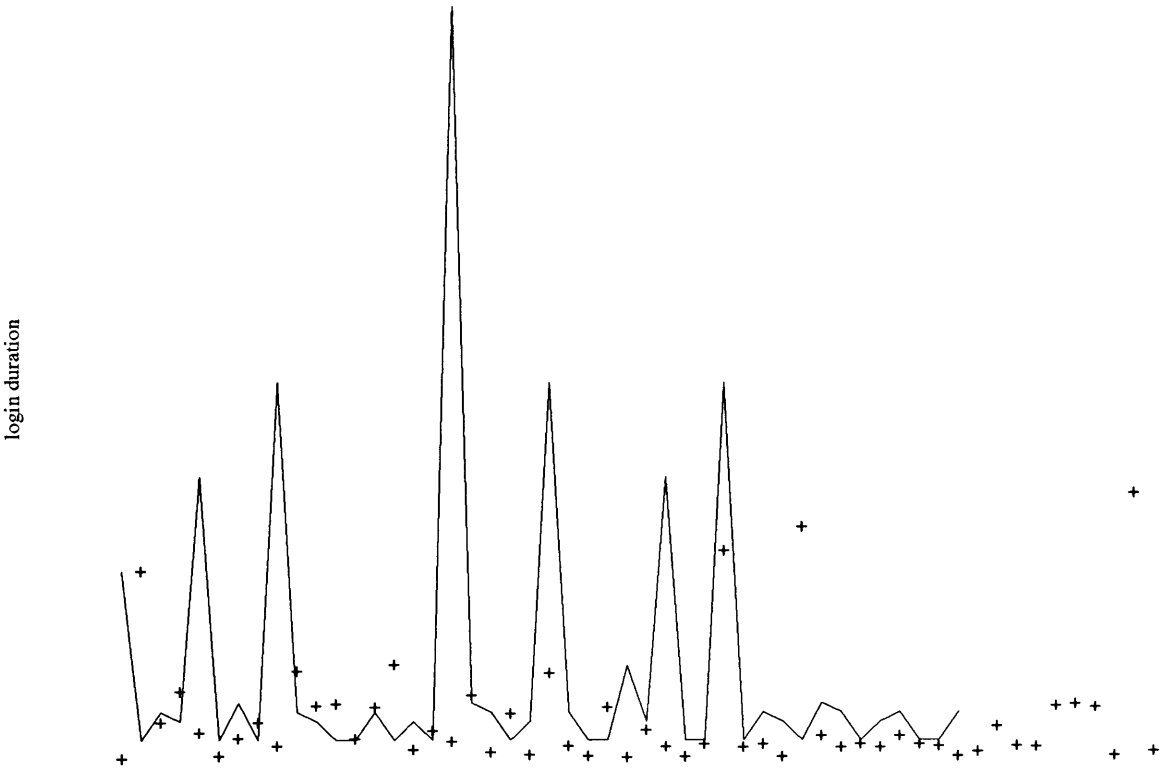

Figure 4.8: DOPPELGÄNGER's duration predictions

has been pushed back from choosing the communities to merely choosing the rules that choose the communities. Eventually, systems will proceed to the next level, choosing the rules that choose the rules that choose the communities.

The ISODATA clustering algorithm is a method of unsupervised pattern recognition, helpful for identifying classes in multidimensional data [The89], [TG74]. I implemented the ISODATA algorithm, to help DOPPELGÄNGER automatically identify interesting communities in the user modeling population.

However, what is to be clustered? User models consist of disparate types of data: proper names, preferences, ages, heights, colors. How can an algorithm compare these different types of values? My solution to this is discussed in the next section.

4.4.1 Knowledge Handlers and the Credit Assignment Problem

Fisher's COBWEB system [Fis90] proposes a good system for identifying salient features in a collection of traits. This seems like it would be good for identifying communities. COBWEB uses a heuristic evaluation measure of category utility to classify data points. It's an advanced, well-known system, but it is of limited usefulness to DOPPELGÄNGER because it only deals with nominal attribute-value pairs. CLASSIT, a sequel to COBWEB, partitions numeric attributes. But neither handles both symbolic and numeric attributes simultaneously. Similar systems are described in [SD90].

It's easy to see why this is. Imagine trying to develop some algorithm that can handle with equanimity data triplets of 1) miles per hour, 2) nationality, and 3) color. There are two choices: you can first classify the numeric attributes to make them nominal, or you can discretize the nominal attributes to make them numeric. The system described in [SM86] converts attribute domains into ranges based on how well they contribute to higher-order conceptual descriptions. A range of values can then be treated like a nominal value.

I developed a different approach to this problem for the community clustering task posed by DOPPELGÄNGER. I wrote several **handlers** for different types of data:

Each of these handlers transforms data in a user model into bitstrings that can all be treated alike: the leftmost bit is the most significant, and every bit is more significant than its right-hand neighbor. The *linearly scaled integers* handler would be used for age; *logarithmically scaled integers* would be used for income; *linearly scaled real numbers* would be used for height; *logarithmically scaled real numbers* would be used for amount of time spent talking on the phone; *proper names* would be used for names of people that the user associates with; *boolean queries* would be used for boolean expressions about news topics, and the *arbitrary string* handler would be used

- linearly scaled integers
- logarithmically scaled integers
- linearly scaled real numbers
- logarithmically scaled real numbers
- proper names
- boolean queries
- arbitrary strings *(these include nominal and unordered attributes)*

for data that can't be ordered in any other way. The effect of each of these handlers is to take a class of inputs and reduce each input to a bitstring such that if you interpret each bitstring as a binary integer, their numeric difference is proportional to their *perceptual* difference, or at least as good an approximation as possible. This list of handlers is all DOPPELGÄNGER currently supports; there should in principle be many more, but these suffice for all the data DOPPELGÄNGER currently gathers.

Within each class, all bitstrings will be the same length, but the length will differ from class to class. The ISODATA algorithm uses an absolute distance metric, so if the bitstrings are kept the same length, the classes with longer bitstrings will have disproportionate influence over the results. To avoid this, each class is scaled up so that its bitstring length is equal to the maximum bitstring length. Then the resulting bitstrings, each comprising one component of a vector are clustered by ISODATA; the top five communities thus generated are shown in Figure 4.9.

These same handlers can be used for more than just community inference through cluster analysis. They can also be used to perform **credit assignment.** Let's say you have a mail application that prioritizes its mail based on a collection of rules personalized to each individual. These rules are imperfect, and the mail application uses feedback from the user to determine the efficacy of each. The rules operate upon disparate data types: some operate on names (who it's from), some operate

```
1. threshold on number of children: 0
2. threshold on annoyance parameter: 3.7
3. knowledge of the command bash
4. has bike
5. privacy of biographical information
```

Figure 4.9: DOPPELGÄNGER's suggested communities.

on boolean queries (used to determine the content of the message), some operate on numeric quantities (the message length). The bitstrings could be computed as for cluster analysis, but then the results could be then input into a neural net. Standard neural net adaptive weighting techniques such as backpropagation would determine the appropriate weights for each rule. The questions of how many hidden layers, input nodes, and what threshold functions to use are interesting. A simple design would have a hidden layer with one node for each rule; more complex designs would match the complexity of each class handler to the complexity of its architecture in its part of the neural net. Perhaps the system should grow over time, adding nodes as more categories of information are acquired. The Cascade Correlation algorithm [FL90] would be good for this. An entropy measure can be used to identify the best predictors.

The disparate data problem is a good argument for needing knowledge representation in a sophisticated user modeling system.

4.5 Computer Aided Introspection

What has DOPPELGÄNGER taught me about myself? Not very much, yet. But that's okay—I already know a lot about myself, and DOPPELGÄNGER's most important task is helping other applications to learn about me. But it has shown me patterns in my behavior. I used its results to program the neural net that decided when to call up

and retrieve my electronic mail from the Media Lab. It wasn't sophisticated enough to tell me what the patterns were, but it showed me the data, which helped me deduce that there were patterns. In time, DOPPELGÄNGER will become sophisticated enough to tell me a lot more about myself, and others.

Chapter 5

What's Next?

5.1 The Twelve Ideas

I've made many prescriptions for user modeling systems in this thesis. Here they are in an easy-to-digest package:

1. Tolerance: Prefer many imperfect sensors to a single sensor. Arbitrate between conflicting sensors.
2. Cooperation: Applications employing user information should speak to each other.
3. Autonomy: Distributed user modeling systems should talk to one another. They should be on your desktop, in your home, and in your shirt pocket.
4. Privacy: User modeling systems should contain safeguards for model privacy and transaction privacy.
5. Curiousity: User modeling systems should ask the user questions every now and then.
6. Hints: Clues to "deep" mental states can be obtained from easily-sensed, "shallow" data.
7. Patterns: As user modeling systems track behavior, they should periodically apply regression analysis to see what patterns emerge.

8. Community: user modeling systems should attempt to identify interesting communities within a population, and use them as a source of educated guesses for questions about users.

9. Passivity: Don't bother the user. Otherwise he'll find another way to occupy his time.

10. Peaks: When there are many data points for some phenomenon, the user modeling system should use spectral analysis and note peaks or troughs in the spectrum, either using the information for itself, or telling people and communities what it has discovered.

11. Slack: Use a *laissez-faire* approach to knowledge representation. Otherwise your system will be too abstract or too general to be useful.

12. Pragmatism: The system should work, and be portable. The best way to propagate ideas is to demonstrate them firsthand.

5.2 The Four Tasks

The tasks were all successful: they are all being currently used by DOPPELGÄNGER, and they will be improved upon. The actual utility of each has been limited more by the sensors than by the techniques themselves. The biggest problem for all of them was sensor overhead. More rigorous evaluation should wait for two things: better sensors and better techniques. My point in this thesis was merely to show that they should be done, not how to do them best.

The most important use of machine learning for user modeling isn't these four tasks. Those would have been stumbled upon eventually. More important is the contribution of ideas to user modeling that changed the architecture of DOPPELGÄNGER.

5.3 What's Next

5.3.1 New Sensors

In general, user modeling needs better techniques of gathering knowledge about people in their information environment. In addition to the broad problems of voice recognition, machine vision, tactile sensing, there are once again "little things" that can be used to infer "big things."

We need lots of sensors everywhere. A Global Positioning System would be a good sensor if combined with maps in BARRIER REEF, but the system provides no interface to the output other than its LCD screen. It's also too bulky, and the cost, at approximately $1200, is prohibitive. Those can be forgiven, but how hard would it have been to provide a digital output jack so that its information could eventually have found its way into a computer? Sensor designers too often see no use for tying their devices into the world of other devices. There's no monetary incentive for them to do so, because the applications made possible by the user modeling that requires multiple sensors doesn't yet exist—a chicken-and-egg phenomenon.

Pascal Chesnais' CANARD is a palm-size news presentation device. It can receive news via packet radio, but the real reason CANARD is of interest to user modeling is because of its mercury tilt sensor. When CANARD is pointed down, as it would be in a belt holster, a blob of mercury slides down, letting the computer would know that the user is not reading news. It can then concentrate on retrieving news rather than displaying news. When CANARD is held up, the blob of mercury slides the other way, letting the computer know it should change its behavior. And when the user walks, the blob of mercury slides back and forth, so the computer would know to make the text larger.

Perfect speech recognition eludes us, but there is some useful information we can deduce from audio. A system for detecting emphasis in speech is described in [Hu88],

and a system for continuously recording a user's speech all day (for easy selection, marking, and retrieval later) is in [Hin91]. Sensors that record ambient noise would also be useful—they might identify characteristic subway noises, or computer hums, and in so doing be able to tell if someone is outside, at work, on a plane, and so on.

The same is true for video: object recognition is a long way away, face recognition is so-so, and merely telling when someone is present is easy. A simple photosensitive sensor could tell the portable user model whether the person is inside or outside during the day.

Sometime I'll build a chair sensor that can tell when someone is sitting in it. This might be a simple piezoelectric device that registers pressure and (hopefully through wireless communication) updates location submodels.

5.3.2 The Architecture

User modeling needs better interfaces. It's ironic that while much of user modeling is designed to aid the man-machine interface, I've never seen a good interface *to* a user modeling system. This is understandable; designing an interface to such a large and complex system is a monumental task.

There's more work to be done on enhancing both model privacy and transaction privacy. The underlying architecture of DOPPELGÄNGER has been solidified; the next level includes better privacy safeguards. Privacy makes DOPPELGÄNGER research more difficult.

The stability of DOPPELGÄNGER has increased almost to the point where automatic installation can occur: installation scripts could facilitate compilation of the DOPPELGÄNGER binaries, libraries, sensors, and databases, and copy BARRIER REEF (or perhaps even assist in managing a distributed knowledge base).

5.3.3 New Applications

DOPPELGÄNGER opens up new possibilities for thousands of applications. Many of these are current applications that can be made better through user modeling. Others are applications that are made possible *by* user modeling.

One application made possible through DOPPELGÄNGER is a simple modeler of "who knows what" that expedites communication by connecting people that have something to say to one another. Such a system might be used to send questions to people best qualified and most likely to answer. Another class of applications in which user modeling often occurs is dating services. Video games could benefit from user modeling components, and I'm still waiting for the chess program that watches my eyes move.

In addition to this augmentation of today's applications, entirely new applications will be made possible by user modeling. These will be systems that depend on the reasoning ability of the user modeling system, that can identify patterns or anomalies, and *deal* with them: perhaps by bringing them to your attention, perhaps by invoking special behaviors tailored to deal with abnormalities in your schedule or the actions you take on a computer. For instance, imagine a system that makes predictions about you. Given some knowledge of what books or songs you like, it could search for other people who like the same selections you do, and make new recommendations for you.

User modeling systems can help make computers into intelligent conversational partners. Along the way, they'll figure out the right questions to ask.

Appendix A

What User Modeling Promises for the Newspaper of the Future

The motivation for the DOPPELGÄNGER *user modeling system arose from my work in the MIT Media Laboratory Electronic Publishing Group.* DOPPELGÄNGER *has used news as a complex domain in which to address topics common to all of user modeling.*

Over the past year, the architecture of the DOPPELGÄNGER user modeling system has been recast toward a prototype *information marketplace.* The four major developments during this time have been a knowledge representation language called SPONGE, a knowledge base called BARRIER REEF, protocols for communication between distributed Doppelgängers, and a framework for privacy safeguards and anonymous transactions. These are all critical components of a network in which your location, your (inferred) reading preferences, your grocery bills, your datebook, and many other sources of personal information all contribute to applications that unify this data into the most effective presentation possible, making use of news, entertainment, and advertisements.

DOPPELGÄNGER is part of an effort to develop an architecture [Die93], [Blo91] that facilitates the creation of electronic newspapers: their distribution, personalization, assembly, and presentation.

A.1 Choosing News

There's more to a personalized newspaper than filtering. Information retrieval systems take an overly narrow view of news selection: given a topic, find the articles inside a huge database that best fit the topic. It's old hat. But personalizing an electronic newspaper can be broken into three more adventurous problems: understanding the news, understanding the user, and presenting the news effectively. DOPPELGÄNGER helps all of these problems, and attempts to provide a "best available" solution for the second problem.

The role of DOPPELGÄNGER in each person's newspaper *depends on that person.* For some, it will mean that they won't have to fill out a questionnaire to tell the newspapers what their interests are, because DOPPELGÄNGER will learn about them over time. For others, it will mean that they will be notified just as soon as possible (through either paper or portable news devices) about certain topics—stock quotes, or sports scores. For still others, it will mean that new services will become available to them through their newspaper. And for everyone, it will mean that they'll be able to treat their newspaper as having the abilities of a computer without any of the frustrations.

One advantage of a generalized user modeling system, as opposed to a user modeling component that resides inside some application, is that it forms the backbone for an integrated computational environment, one in which your newspaper, your datebook program, your electronic mail program, and your car can all coordinate their

actions by sharing information about you. For those leaving their computational environment (which includes both home and work), their user model can be carried on a portable user modeling card, for services that require "user modeling on demand."

Information retrieval systems don't learn over time; news systems that focus on incrementally *adapting* to the user are rare. [JHL91] and [Bac91] use neural nets to choose appropriate news, and [SM93] uses genetic algorithms for information retrieval. But in addition to learning how to better search for news, DOPPELGÄNGER learns more about *you*. This modeling has to occur if the system is to grow with the needs of the user.

Credit card companies, magazines, and governments all model their users. What they don't do is let the user see how he is being modeled, and allow the user to make his own adjustments. In the future, as DOPPELGÄNGER's architecture is fleshed out, there will be an emphasis on the *user modeling interface*: viewing sensor input, DOPPELGÄNGER's inferences upon that data, and the reasoning behind DOPPELGÄNGER's newspaper decisions.

A.2 Advertisements

Electronic newspapers will be paid for by two sources: the reader and advertisers. Advertisers will have a special interest in user modeling; not only do individual user models contain a consumer profile, but the communities maintained by DOPPELGÄNGER perform demographic modeling for free.

Reebok will want to know who bought Nikes six months ago, newspapers will want to know what parts of the paper people really read, and McDonald's will want to know what you had for dinner last night and how much you paid. Your computer at home can make all this information available to them—for a price.

Americans' fear of unwanted junk mail underscores the need for privacy in DOPPELGÄNGER. There are two types of privacy involved in user modeling: privacy of

the user model itself, and privacy of the transactions made on behalf of your user model. The former is a data security issue; the latter is an underlying issue of the nature of transactions in an electronic marketplace.

Of course, there will be less unwanted junk mail with more accurate user modeling. The direct marketing industry will be better able to target their advertisements when they have an idea of the consumer habits of each household. The result will be that you'll see fewer ads, and the ones you do see will be of greater interest to you.

A.3 Communities

When you say, "I'd like to see Walter's newspaper," DOPPELGÄNGER remembers that. If you're happy with what you read, DOPPELGÄNGER might nudge your newspaper in his direction. Or it might make construct a section for future newspapers with a sampling from Walter's newspaper, or refrain altogether, depending on what it knew about you. I can choose to view my newspaper, Walter's newspaper, or a newspaper anywhere in between.

Everyone is a member of many different, loosely defined, *communities.* I'm a member of the student community, the MIT community, the Cambridge community, the US community, and the coin-collecting community. DOPPELGÄNGER assumes that news that interests one of my communities will be likely to interest me as well, and that if Walter and I are in many of the same communities, then we will share many interests.

DOPPELGÄNGER allows for treating communities as individual user models, so that the system can build a news environment for a community: not only can I see Walter's paper (if he lets me), but I can see a newspaper constructed for the entire Media Lab, or one that is 20% me, and 80% Media Lab. I can even "caricature" my user model, and get a newspaper that's 120% me and -20% the Media Lab: in essence, removing the Media Lab influences from my user model.

Electronic communication will also create (and recently has created) new communication flows. Instead of the classic mass media, in which a few produce articles for many, we'll see newspapers in which readers can jot off an electronic letter to an author, and get a response back within minutes or hours, in time for the reader to write again. And as it happens, other readers could tune in, or even join in.

A.4 Predictions

Ideally, each newspaper reader would have his own editor. And that's where DOPPELGÄNGER comes in handy. Applications making use of DOPPELGÄNGER can embody some of an editor's knowledge to help a local computer make decisions about each issue of an electronic newspaper. DOPPELGÄNGER isn't going to replace an editor. This would be bad for two reasons: first, DOPPELGÄNGER just plain isn't *smart* enough to always tell what's important and what's not. Second, there needs to be some standardization, so people have some common topics of conversation at cocktail parties.

The traditional newspaper isn't going to disappear. It has too much in its favor: familiarity, intuitiveness, and most of all, it's made of paper. Electronic newspapers will *supplement*, not *supplant*, the newspaper. There are currently hundreds, and probably thousands, of channels by which people can receive news electronically. The opportunities for enhancing the selection, manipulation, and presentation of the news environment with user modeling are immense.

Appendix B

Barrier Reef

This is a brief synopsis of the information stored in the BARRIER REEF *knowledge base.*

Each entry described below (save the first) is a directory. Most of the knowledge in BARRIER REEF has either been entered by hand or obtained from other structured knowledge sources. It is hoped that BARRIER REEF may one day be able to glean its own information from news articles automatically, like the CYC knowledge base[GL93].

Index.dtbin

A single file (the ".dtbin" extension indicates Dtype binary format) providing information about other entries at this level (the top level) of BARRIER REEF. Most directiories inside BARRIER REEF contain an Index.dtbin file.

Relations

A collection of SPONGE assertions about facts that are used for personalized newspaper selection, *e.g.* "baseball is_a sport."

Names

A collection of names that DOPPELGÄNGER uses to distinguish names of people from other proper names.

Places

A collection of names that DOPPELGÄNGER uses to distinguish names of places from other proper names.

Lexicon

BARRIER REEF's most ambitious attempt at a general knowledge base: its directory structure reflects an attempt to classify knowledge into fifteen subdivisions:

- Abstract
- Body
- Commerce
- Domestic
- Entertainment
- Food
- Industry
- Life
- Measurement
- People
- Sensations
- SpaceTime
- Substances
- Thought

- Transport

I came upon these categories while reading a lexicon [McA81], which is like a dictionary sorted by topics (*e.g.* there is a section with all of the verbs and nouns related to "hitting things"). This inspired the observation that I should duplicate its structure in BARRIER REEF—with the justification that "if there are enough words in English to justify a subdivision in the lexicon, it merits a category of its own in BARRIER REEF." That's why these fifteen top-level subdivisions look unusual for a knowledge base, with "Body" meriting a spot equal to "Life" and "Substances", both of which are more general categories.

In some sense, this division of BARRIER REEF (in contrast to other knowledge bases) strives not for an organization of knowledge that is logically or philosophically "correct", but rather attempts to divide knowledge based on utility—measuring how we have chosen to assign importance to concepts through the evolution of our language.

Commands

Knowledge about UNIX: what the different commands mean, and the Hidden Markov Models and states used for state analysis (described in section 4.2).

Applications

Knowledge particular to an application: either information describing how to access the application (*e.g.* this application can be reached at this machine with this port number) or other information (not necessarily SPONGE code) for use by the application.

Appendix C

Sponge Tags

This is the listing of the English names for the Sponge Tags, which are simply integers used in the C programs that comprise DOPPELGÄNGER and BARRIER REEF. Each datum in SPONGE is a Dtype list, and the first element of the list is the Sponge Tag, which says what information will follow in the rest of the list. An explanation of the tag is given in italics after the name. In the descriptions below, reference is often made to the "surrounding datum." This means that the Sponge Tag is part of a datum that is within some other datum. For instance, some assertions contain three parts: the title of the assertion, the assertor, and an opinion, which is itself a sponge datum.

The Sponge Tags illustrate my *laissez-faire* approach to knowledge representation.

1. object *The most "generic" sponge type—anything may follow the tag.*
2. int_binding *A string is bound to the following integer.*
3. string_binding *A string is bound to the following string.*
4. real_binding *A string is bound to the following real number.*
5. time_binding *A time value follows.*

6. assertion *An assertor-assertion pair: the person or sensor who is making a claim, followed by the claim.*

7. distribution *The name of a probability distribution, followed by its parameters in this instance, e.g. mean and variance.*

8. confidence *The confidence in the preceding statement. Assertions often contain a confidence. They range between 0 and 1.*

9. accuracy *An estimate of the accuracy of a statement. Also ranges between 0 and 1. How does this differ from confidence? Here's an example: "I am confident (confidence .8) that answer is very accurate (accuracy .99)." as opposed to "I am confident (confidence .8) that answer is very inaccurate (accuracy .2)." Assertors will have different confidences about different accuracies of a statement.*

10. condition *A prepositional phrase or a confidence judgement.*

11. VO *A verb-object pair.*

12. SVO *A subject-verb-object triple.*

13. domain *A domain submodel.*

14. address *Information for finding something in either the* DOPPELGÄNGER *database or* BARRIER REEF.

15. ptr *A pointer to a* SPONGE *address.*

16. backptr *The inverse of a pointer: an indication that something is pointed to.*

17. and *Boolean* AND.

18. or *Boolean* OR.

19. not *Boolean* NOT.

20. implies *A condition followed by another condition; the first implies the second.*

21. equation *The name of an equation, followed by the equation itself.*

22. variable *A free variable, used in equations and distributions.*

23. application *The name of an application, followed by information describing it.*

24. pathname *A string containing a UNIX pathname.*

25. map *Where to find information in* BARRIER REEF.

26. file *Indicates a file. The value will usually be of sponge_type* PATHNAME.

27. time *UNIX time: number of seconds since January 1, 1970.*

28. future *These three tags are used to indicate the tense of a* SPONGE *datum.*

29. present

30. past

31. attributes *What follows is a list of attributes that are taken to modify the surrounding condition.*

32. word *A word, plain and simple. This is usually used for talking about the text of a news article.*

33. exec *A tag used by* **doppelserve** *indicating that the following C function should be executed.*

34. rule *A sequence of actions to be followed when the preceding conditions are fulfilled. One or more of these actions will often be an exec.*

35. degree *Anything that varies between 0 and 1.*

36. units *What the surrounding datum is measured in, e.g. inches.*

37. privacy *A privacy category (see Section 2.8.1) followed by an optional list of people who are permitted/denied access to the data.*

38. opinion *A belief about something. If someone says he doesn't like firearms, he is making an assertion about an opinion.*

39. relation *A verb and one or more nouns follows.*

40. is *These are particular relations.*

41. contains

42. is_described_by

43. is_defined_by

44. describes

45. like *These six tags are particular relations used by dopmail: they contain the number of times an application has said that the user likes/hates/... something. The beta distribution is then used to estimate the user's true preference for something given this possibly conflicting information. See section 3.3.*

46. hate

47. like_more

48. hate_more

49. like_less

50. hate_less

51. markov_model *The name of a Markov Model, followed by a matrix of transition probabilities, a matrix of output probabilities, and lists of the states and output symbols.*

52. matrix *The name of the matrix, followed by its dimensions, followed by its values.*

Bibliography

[Abr92a] Nathan Abramson. Context-sensitive multimedia. Master's thesis, Massachusetts Institute of Technology, 1992.

[Abr92b] Nathan Abramson. dtype++ — unification of commonly used data types. Technical report, Electronic Publishing Group, MIT Media Laboratory, October 1992.

[And71] T. W. Anderson. *Statistical Analysis of Time Series.* John Wiley and Sons, Inc., 1971.

[Bac91] Paul E. Baclaski. Personalized information filtering. Technical report, Autodesk, Inc., 1991.

[Blo91] Alan Blount. Bettyserver: More news than you can beat with a stick. Technical report, Electronic Publishing Group, MIT Media Laboratory, December 1991.

[Blu90] Avrim Blum. Separating distribution-free and mistake-bound learning models over the boolean domain. In *Proceedings of the Thirty-First Annual Symposium on Foundations of Computer Science.* IEEE, 1990.

[Car89] Jaime Carbonell. *Machine Learning: Paradigms and Methods.* MIT Press, 1989.

[Cha92] David Chaum. Achieving electronic privacy. *Scientific American*, pages 96–101, August 1992.

[Che90] Peter Cheeseman. *On Finding the Most Probable Model*, pages 73–93. Morgan Kaufmann, Inc., San Mateo, California, 1990.

[Chi] David N. Chin. *KNOME: Modeling what the User Knows in UC.*

[CK93] Pascal Chesnais and Douglas Koen. Strategies for personal dynamic systems: News in the future. In *NextWORLD Expo*, 1993.

[Die93] Klee Dienes. Newskit: An electronic newspaper toolkit. Bachelor's Thesis, MIT Department of Electrical Engineering and Computer Science, 1993.

[Dra67] Alvin W. Drake. *Fundamentals of Applied Probability Theory.* McGraw-Hill, Inc., New York, 1967.

[Fis90] Douglas Fisher. *Knowledge Acquisition via Incremental Conceptual Clustering.* 1990.

[FL90] Scott E. Fahlman and Christian Lebiere. *The Cascade-Correlation Learning Architecture.* Morgan Kaufmann, 1990.

[GB86] Simson Garfinkel and Walter Bender. A pressure-sensitive keyboard, 1986. This was built in the Media Laboratory, but never written up.

[GL93] R.V. Guha and D.B. Lenat. *CYC: A midterm report*, chapter 8, pages 839–866. 1993.

[Hin91] Debby Hindus. Infinite audio. Master's thesis, MIT Media Lab Speech Research Group, 1991.

[Hu88] Antonio Hu. Detecting emphasis in speech. Bachelor's Thesis, MIT Department of Electrical Engineering and Computer Science, 1988.

[Int93] Emerging standards for mobile computing enhancements: Pcmcia and the exca standard, January 1993.

[JHL91] Andrew Jennings, Hideyuki Higuchi, and Huan Liu. A personal news service based on a user model neural network. In Judy Kay and Alex Quilici, editors, *Proceedings of the IJCAI Workshop W.4 Agent Modelling for Intelligent Interaction*, August 1991. *Augmented keyword search on news articles.*

[Kap82] S. J. Kaplan. Cooperative responses from a portable natural language query system. *Artificial Intelligence*, 19, 1982.

[Kay91] Judy Kay. Generalised user modelling shells - a taxonomy. In Judy Kay and Alex Quilici, editors, *Proceedings of the IJCAI Workshop on Agent Modelling for Intelligent Interaction*, August 1991. *Augmented keyword search on news articles.*

[KF91] Robert Kass and Tim Finin. *General User Modeling: A Facility To Support Intelligent Interaction*, pages 111–128. ACM Press, New York, 1991.

[KL91] Sandra Katz and Alan Lesgold. Modelling the student in sherlock ii. In *Agent Modelling for Intelligent Interaction Workshop, International Joint Conference on Artificial Intelligence*, 1991.

[Kob91a] Alfred Kobsa. User modeling. In *Tutorial presented at the 1991 IEEE Conference on AI Applications*, 1991.

[Kob91b] Alfred Kobsa. User modeling: Tutorial presented at the 1991 ieee conference on ai applications, 1991.

[KV93] Michael Kearns and Umesh Vazirani. *Topics in Learning Theory.* 1993. To be published. Has an excellent coverage of learning by example.

[Lju87] Lennart Ljung. *System Identification: Theory for the User.* Prentice Hall, Inc., 1987. *A theoretical (and mathematical) approach to inferring models (patterns) of unknown systems. For user modeling, the unknown system is the user.*

[McA81] Tom McArthur. *Longman Lexicon of Contemporary English.* Longman House, Essex, England, 1981.

[MCM86] Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell. *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufman, 1986. A three volume set from 1986 to 1990, and the standard reference for machine learning.

[MGGC91] Gordon McCalla, Jim Greer, Dinesh Gadwal, and Randy Coulman. Granularity-based student modelling and plan recognition. In Judy Kay and Alex Quilici, editors, *Proceedings of the IJCAI Workshop W.4 Agent Modelling for Intelligent Interaction*, August 1991.

[Min86] Marvin L. Minsky. *The Society of Mind.* Simon and Schuster Inc., New York, 1986. Helpful for its ideas on representations and its insights into aspects of personality.

[MST91] Claus Mobus, Olaf Schroder, and Heinz-Jurgen Thole. Runtime modelling the novice-expert shift in programming skills on a rule-schema-case continuum. In Judy Kay and Alex Quilici, editors, *Proceedings of the IJCAI Workshop W.4 Agent Modelling for Intelligent Interaction*, August 1991.

[Orw91a] Jon Orwant. Doppelganger: A user modeling system. Bachelor's Thesis, MIT Department of Electrical Engineering and Computer Science, 1991.

[Orw91b] Jon Orwant. The doppelganger user modeling system. In *Agent Modelling for Intelligent Interaction Workshop, International Joint Conference on Artificial Intelligence*, 1991.

[OSKD90] Tim Oren, Gitta Salomon, Kristee Kreitma, and Abbe Don. Guides: Characterizing the interface. In *The Art of Human Computer Interface Design.* Addison Wesley, 1990.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, New York, 1992.

[Rab89] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, 1989.

[SD90] Jude W. Shavlik and Thomas G. Dietterich, editors. *Readings in Machine Learning.* Morgan Kaufmann, 1990. Comprehensive coverage of symbolic machine learning.

[SM86] R. E. Stepp and R. S. Michalski. *Conceptual clustering: Inventing goal-directed classifications of structured objects*, volume 2. Morgan Kaufmann, Los Altos, CA, 1986.

[SM93] Beerud Sheth and Pattie Maes. Evolving agents for information filtering. In *Proceedings of the Ninth Conference on Artificial Intelligence for Applications*, March 1993.

[SNS88] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. Technical report, MIT Project Athena, Cambridge, MA, 1988.

[Spe88] Thomas H. Speeter. Flexible, piezoresistive touch sensing array. *SPIE Optics, Illumination, and Image Sensing for Machine Vision III*, 1988.

[Spe90] Thomas H. Speeter. Smart floor and smart desktop: Concepts and architecture. Technical report, AT& T Bell Laboratories, Holmdel, NJ, 1990.

[Spe93] Thomas H. Speeter. Identification using ground reaction force patterns, 1993. Submitted to SIGCHI.

[ST91] Daniel Sleator and Davy Temperley. Parsing english with a link grammar. Technical Report CS-91-196, Carnegie Mellon University Computer Science Department, October 1991.

[TBKC91] Richard Thomas, David Benyon, Judy Kay, and Kathryn Crawford. Monitoring editor usage: The basser data project (part ii). Technical report, University of Sydney Basser Department of Computer Science, July 1991.

[TG74] J.T. Tou and R.C. Gonzalez. *Pattern Recognition Principles.* Addison-Wesley, 1974. *A good standard reference for pattern recognition: the sections on classifiers and feature selection are most relevant.*

[The89] Charles W. Therrien. *Decision, Estimation, and Classification.* John Wiley & Sons, New York, 1989.

[TP91] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1), 1991.

[Web91] Geoffrey Webb. An attribute-value machine learning approach to student modelling. In *Agent Modelling for Intelligent Interaction Workshop, International Joint Conference on Artificial Intelligence*, 1991.

[Wei72] Gerald M. Weinberg. *A Computer Approach to General Systems Theory*, pages 98–141. Wiley-Interscience, New York, 1972.

[Wym72] A. Wayne Wymore. *A Wattled Theory of Systems*, pages 270–300. Wiley-Interscience, New York, 1972.