Morphing Content in Mobile Applications

by

Kevin Y. Wang

S.B. Electrical Engineering and Computer Science, M.I.T., 2008

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

[June]
May 20, 2009

Author_____
Department of Electrical Engineering and Computer Science
May 20, 2009

Certified by_____
Professor Glen Urban
David Austin Professor of Marketing
Chairman, Center for Digital Business at MIT
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Morphing Content in Mobile Applications

by
Kevin Y. Wang
S.B. Electrical Engineering and Computer Science, M.I.T., 2008

Submitted to the
Department of Electrical Engineering and Computer Science

May 20, 2009

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Smart phones are quickly becoming an integral part of our everyday lives. However, the mobile industry is still young, and the full potential of mobile phones has yet to be tapped. In this thesis, I present the design of a new mobile "super-application" called MobileHelp that aims to push the boundaries of how smart phones can make people's lives easier. MobileHelp uses Bayesian inference to determine a user's current purpose. Then, it suggests applications the user may want to use, offers deals and discounts for relevant nearby businesses, shows information about nearby friends and their statuses, and present search results for relevant queries. The power of MobileHelp is that it does all this without actively querying the user for information. It can use past information to make an accurate guess at the user's current purpose, which, if wrong, can be corrected by the user and learned from. I discuss how such a system is conceptually designed and then go into the details of how it could be implemented on the Android platform.

The purpose of this thesis is to lay out the framework for a context-aware mobile application that can be implemented as a first-stage demonstration for France Telecom/Orange.

Thesis Supervisor: Glen L. Urban
Title: David Austin Professor of Marketing
Chairman, MIT Center for Digital Business

Acknowledgements

I would like to begin by thanking my advisor Professor Glen Urban for his help, guidance, and friendship. Glen helped introduce me to a world of digital marketing that I never knew existed before. I've learned many things over the past year working with him, from how to build and a lead a team of diverse skills and backgrounds, to how to think outside the box to tackle open-ended questions like those that appear in the field of marketing. I would like to thank Erin MacDonald for the great work she did managing the General Motors and Suruga Morphing projects. I should also mention Professor Gui Liberali, who has been a kind friend and generously offered up his time to answer my questions about the mathematics of morphing.

I want to thank my fellow researchers Ele Ocholi and Jong-Moon Kim. They have been great partners to work with, and made the lab a very fun place. I also owe thanks to Clarence Lee, whom I consider to be one of my great mentors at MIT. He was responsible for my development as a leader in the MIT Technology Fair, and was the person who first brought me into Glen's lab. I would also like to thank Jimmy Li and Shirley Fung for their help and friendship when I first joined the lab.

I would be remiss to not thank Dorothée Bergin and Diego Panama for their invaluable contributions to the France Telecom/Orange project, which is the subject of this thesis. Besides being a real pleasure to work with, they are tremendously creative people whose innovative thinking helped drive the project to where it is today.

Finally, I would like to thank my parents, who have inspired me in vastly different ways. My mom, by her strength and humility in the face of terrible adversity, has set an example of how to stay positive no matter what the circumstances. My dad, through his persistent wise words, has made me realize that I only am solely responsible for my success and fate in life. Most importantly, I thank them for the sacrifices they made in their life, so that I could have the freedom and opportunity to pursue whatever I wanted in mine. The gift of opportunity is the greatest gift of all, and I wouldn't be where I am today without them.

Table of Contents

Chapter 1: Introduction

Imagine it's the year 2010, and you, like a majority of people in the country, have an internet-enabled smart phone. Everywhere you bring this phone, it seems to know exactly what you want and when you want it. When you're commuting to work in the morning, it brings up the train schedule and morning news for you to read. When you're going out to lunch with your coworkers, it suggests a 10% discount for a place nearby that your friends have been to. When you're home at the end of the day, it realizes you're trying to find the show times of a new show, so it automatically brings up the TVGuide application, even though you don't yet own it. On the weekends when you're out shopping, it shows you nearby friends who are doing the same so you can meet up. Later in the day, your phone suggests you and your friends go to a movie theater. In the process, it automatically brings up the Fandango application so you can find show times and purchase tickets ahead of time. Best of all, it does all of this without it ever asking you for anything.

1.1 Vision

This thesis discusses the design of a mobile phone application that uses Bayesian inference to infer user purpose and offer helpful recommendations. The goal is to work towards the user experience just described by building intelligence into mobile phones in way that hasn't been done before. Ideally, the smart phone of the future will

- Know what you're doing based on passively observing your actions and comparing it to your previous history.

- Offer extremely relevant suggestions and deals for you that will make your life easier and save you money.
- Present information to you in a way that resonates with how you think.

This is the result of collaboration with Orange Labs, the research division of France Telecom, and builds on the work of previous projects that used Bayesian inference to morph content.

Hitherto, morphing technologies have been applied to websites. The idea is to learn about enduring latent variables, specifically, cognitive style. Once we know a person's cognitive style (for example, whether they're analytic or holistic), we can display web pages in a way that better suits them. Previous work with Suruga Bank in Japan resulted in the design, implementation, and test of such a morphing site that showed improvement in user satisfaction.

In this project, we would like to apply the same Bayesian inference and morphing technologies to mobile phones. In addition to using Bayesian inference to infer cognitive style, we are also applying it to infer user purpose. This is a new area of research and we hope that our inference engine can perform in this capacity. We will also be morphing the recommended content based on cognitive style. This is especially important in a mobile environment where screen real estate is at a premium.

1.2 Overview

In Chapter 2, I illustrate the concept of content morphing by discussing cognitive styles and previous work with Suruga Bank.

In Chapter 3, I give an overview of the mathematics of morphing. Specifically, I discuss the Bayesian inference engine and Gittins index.

In Chapter 4, I introduce MobileHelp our mobile assistant application. I discuss our choice of the Android platform for this project and system design decisions for the demo and prototype.

In Chapter 5, I give specifications for data models and a few important classes of our demonstration application. I also provide some implementation hints based on past mobile development experience.

Finally, in Chapter 6, I summarize the contributions of this thesis and lay out plans for future work.

Chapter 2: Content Morphing

The idea of morphing is based on the fact that humans have difference preferences for how information is presented to them. This is grounded in the different ways that people process information and captured in the theory of cognitive styles.

2.1 Cognitive Styles

Cognitive styles influence how people think process information, and learn. Examples of contrasting cognitive styles are Analytic/Holistic and Visual/Verbal. A person is analytic if they want to see all the nitty-gritty details, while a person is holistic if they're more interested in understanding the whole picture. Similarly, a person is visual if they prefer graphics and diagrams and verbal if they prefer textual explanations.

One definition of cognitive style is given by Riding and Rayner, who say "Cognitive style is seen to be an individual's preferred and habitual approach to

organizing and representing information." (Riding & Rayner, 1998) Cognitive style is an enduring latent variable. That is, it is one which is not directly observable but must be inferred (latent), and also enduring in that it for humans, it generally does not change over time.

This implies two things. The first is that we need to be clever in evaluating a user's cognitive style. Second, once we have estimated a user's cognitive style, we do not need to estimate it again later. We can then use our knowledge of a user's cognitive style to our advantage by morphing content. The details of how to make these inferences and choose which morphs to display is discussed in the next section. In the meantime, we provide an overview of a previous morphing project to give a better sense of how morphing works in practice.

2.2 Previous Work with Suruga Bank

Since 2006, we have been working with Suruga Bank in Japan to design a morphing card loan website. The site uses a combination of Analytic/Holistic, Deliberative/Impulsive, Hierarchical/Egalitarian, and Individualistic/Collectivistic cognitive styles. Deliberative people tend to plan out their decisions more than impulsive people. Hierarchical people have more respect for expert opinions while egalitarian people believe in the equality of all opinions. The last pair contrasts users who like to form their own opinions about issues against those who like to seek group opinions.

There are two components to this morphing site. The Bayesian engine evaluates a user's clickstream on the website, and from that, infers the user's cognitive style. This was done by associating characteristics to each link, after which a user panel was done to evaluate how those link characteristics map to cognitive styles. After this mapping was

done, we could build a probabilistic model of a user's cognitive style based on the links they clicked and the order they clicked them. This was built by Clarence Lee, a former research assistant (Lee, 2008).

The second component is the Gittins index. Given a probability distribution of cognitive styles for a user, this engine decides which morph to show a user next. A previous research assistant, Shirley Fung, designed and implemented the morphing layout functionalities of the website (that is, how to turn the output of the Gittins index to an actual webpage). Section 5.1.1 of her thesis show great examples of how the same data can be presented in two entirely different ways based on cognitive style (Fung, 2008). For purposes of exposition, I will include a couple of samples to let you see the contrast between the different morphs.

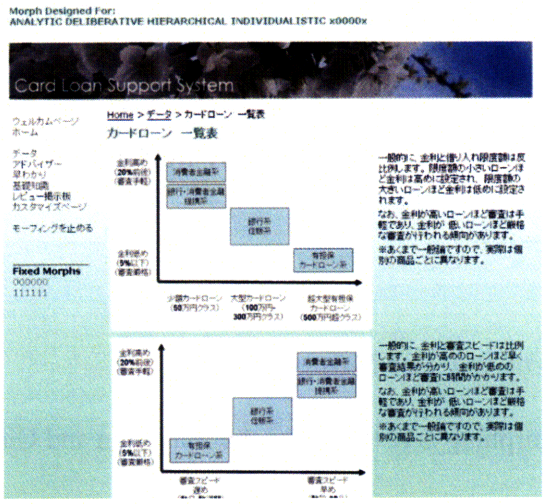


Figure 1: Analytic/Deliberative/Hierarchical/Individualistic morph of a data specs page. Notice the abundant text, attention to detail, and how the data is projected two ways to allow different analysis.

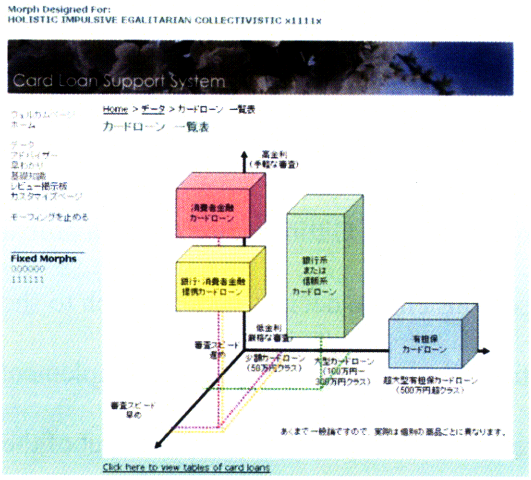


Figure 2: Holistic/Impulsive/Egalitarian/Collectivistic morph of the same data specs page. The data is much more condensed and summarized with almost no descriptive text. Also, the data is not projected and shown in one condensed 3D view.

Previous inference work has been done by the Reality Mining group at the MIT Media Lab. Some of their work involved trying to predict what users were going to do at any point in time (Farrahi & Gatica-Perez, 2008), and infer relationship status in a social network (Eagle, Pentland, & Lazer, 2009). Their tools were focused more on machine learning methods such as Hidden Markov Models. Ours is the first example known to us that uses Bayesian inference. More information is available on the group website: http://reality.media.mit.edu/.

Chapter 3: Mathematics of Morphing

The mathematics of morphing is divided into the Bayesian and Gittins engines. Bayesian inference uses Bayes rules to calculate the probability of being in a certain cognitive style given a user's clicks. The output is a vector of probabilities of being in each cognitive style, which is fed into the Gittins engine. The Gittins engine then decides the optimal morph to show the user. As we will see, this is not simply the morph with the highest probability. The paper "Website Morphing" by Hauser, Urban, Liberali, and Braun

describes these methods in great detail (Hauser, Urban, Liberali, & Braun, 2009). We provide a brief summary here.

3.1 Bayesian Inference

Our Bayesian inference engine uses a multinomial logit function to model the probability of the user clicking a given link. After this is calculated, Bayes rule is used to calculate the probability of being a certain cognitive style based on the link the user chose. The probability of clicking on a specific link is defined as:

$$f(\vec{y}_{kn}|\vec{c}_{kjn}, r_n, \Omega) = \prod_{j=1}^{J_k}\left(\frac{e^{\vec{c}_{kjn}\Omega r_n}}{\sum_{l=1}^{J_k} e^{\vec{c}_{kln}\Omega r_n}}\right)^{y_{kjn}}$$

where for a particular user n on web page k, J_k is the number of links on the page, y_{kjn} is 1 if the user clicked link j and 0 otherwise, $\vec{c}_{kjn}$ is a vector of link characteristics, r_n is a vector describing cognitive style, and Ω is a matrix (estimated from a pre-study) that maps link characteristics to cognitive style.

This vector of probabilities of clicking on a specific link is calculated for each user after every click. Then, the posterior probability of being in a certain cognitive style is calculated by Bayes rule as follows:

$$q_{rn} = f(r_n|\vec{y}_{kn}, \vec{c}_{kjn}, \Omega) = \frac{\prod_{k=1}^{K_n}\prod_{j=1}^{J_k} q_0(r_n) f(\vec{y}_{kn}|\vec{c}_{kjn}, r_n, \Omega)}{\prod_{k=1}^{K_n}\prod_{j=1}^{J_k}\sum_{r_0}^{15} q_0(r_n) f(\vec{y}_{kn}|\vec{c}_{kjn}, r_n, \Omega)}$$

where $q_0(r_n)$ is the prior distribution (taken to be uniform before any user data, and afterwards is the posterior from the click before), k indexes the click made by the user (the first click, second click, etc.), and K_n is the number of clicks made by user n. Notice that in the denominator, we sum over 16 cognitive styles (r_o from 0 to 15). This is

because in the Suruga study, we have four binary dimensions, giving us $2^4 = 16$ unique combinations of cognitive styles. In the study with France Telecom, we have two dimensions (Analytic/Holistic and Visual/Verbal) so we would sum over $2^2 = 4$ cognitive styles. The result, q_{rn} is a vector of probabilities of being in one of the 16 combinations of cognitive styles. This is fed into the Gittins Engine (below), which decides which morph to show.

3.2 Gittins Index

The purpose of the Gittins engine is to determine the best morph to shown to a user at a specific time, given the probabilities of the user have a combination of cognitive styles. The greedy approach would be to show the morph corresponding to the cognitive style with the highest posterior probability (in other words, maximum a posteriori). However, this is not ideal since it's easy to get stuck in local minima: if we keep always show the user the morph with the highest probability, then the user may never get a chance to explore other morphs which may better fit him.

The Gittins engine solves this problem by calculating what are called Gittins indices. Without going into much detail, the idea is determine which morph has the highest Gittins index, and display that morph. The index value takes into account both the present and future discounted value of showing this morph. Please refer to Section 4 of "Website Morphing" for a more detailed explanation.

Chapter 4: France Telecom/Orange Mobile Morphing

4. 1 Goal

The goal of the France Telecom/Orange Mobile Morphing project is to develop a new mobile phone application that combines Bayesian inference on cognitive style and user purpose with proprietary network data about user statistics and usage. Specifically, France Telecom is interested in improving the mobile experience for its users by making life more convenient for them.

This project will culminate in a demonstration application, not a full-fledged prototype. If this demo proves interesting to FT (or perhaps other network providers), a reasonable follow-up project is to create a live working prototype that communicates with network providers to get access to user search histories and other proprietary information.

This demo will consist of a few storylines that should show evidence of potential in mobile morphing to better user experiences, should more funds be invested in this work. It will be limited in scope in that the application will not consist of any "live" features. Instead, the application will consist solely of pre-determined scenarios that show off the breadth and depth of what morphing could do. We will have a simple inference engine running, but probabilities will not be updated. They will be hardcoded in the database.

We also wanted to limit the demo entirely to the phone itself was for portability. If someone wanted to show the application to an executive in France, they would not have to worry about internet connectivity and whether there was a server with a database running somewhere that could handle the network API calls. However, in this document, we consider the design of a prototype system that much more mimics real-life scenarios with dynamic data and inference.

4.2 The MobileHelp Application

We achieve the goal by designing a mobile application called MobileHelp. MobileHelp is a mobile concierge service that tries to offer the use relevant information, applications, business deals, and information on friends based on the user's inferred purpose. That is, the application uses Bayesian inference to determine (on a coarse level) what the user is currently in the process of doing. Based on that inference, it tries to offer the user helpful suggestions or information. Specifically, MobileHelp offers

- Information scraped from the web – Examples include search results, news stories, weather info, etc.
- Applications the user may be interested in – There are 25,000 applications in the iPhone marketplace. Finding the one you need when you need it is extremely difficult. We lessen that burden on you.
- Deals about businesses and restaurants nearby – Deals that are relevant to the user. Includes restaurant, bar, movie, and shopping savings.
- Information about nearby friends – Discover who among your friends are nearby, and what they're doing.

4.2.1 Rationale

When we were designing this application, we wanted to create something that could leverage our proprietary Bayesian inference engine. Inferring user purpose seemed a natural application of that engine in a new context. Then, assuming we had user purpose, we brainstormed problems that this information could be applied to solve.

One problem that users face today is the overwhelming number of applications available for download. The iPhone app store has twenty different categories with over

25,000 applications. This number will only increase in the coming years. We believe this problem can be attacked by offering users applications, whether owned or not, that are relevant their current purposes. These applications would be tagged by and presented to the user based on demographic information, location, time, and purpose. For example, we may serve a news application to a user commuting to work, and a TV Guide application to a user coming home who recently searched for "American Idol" on their phone.

Another situation where we can use purpose to improve user's lives is by offering them relevant discounts and promotions for nearby businesses. The idea is that if the user is interested in buying food or going shopping, we can serve them deals from restaurants, bars, or stores that they may be interested in. This is substantially different from traditional advertising in that these ads are elicited by the user. If the user never uses this feature of the application, then they will never see any ads. Furthermore, we expect the action rate on these ads to be much higher than traditional ads because the relevancy is orders of magnitude greater. We will only serve you food ads when it's time to eat, and discounts at Gap or Best Buy when you're going shopping.

Lastly, people are interested in where their friends are and what they're doing. We decided to put in a friends feature in our application. The idea is that MobileHelp can determine the location and purpose of nearby friends who also have the MobileHelp application installed. If your friend is having lunch nearby, we can alert you and present an opportunity to meet up, as well as deals you may both be interested in.

4.2.2 Making Inferences

The following figure gives a high level view at the data flow in MobileHelp:

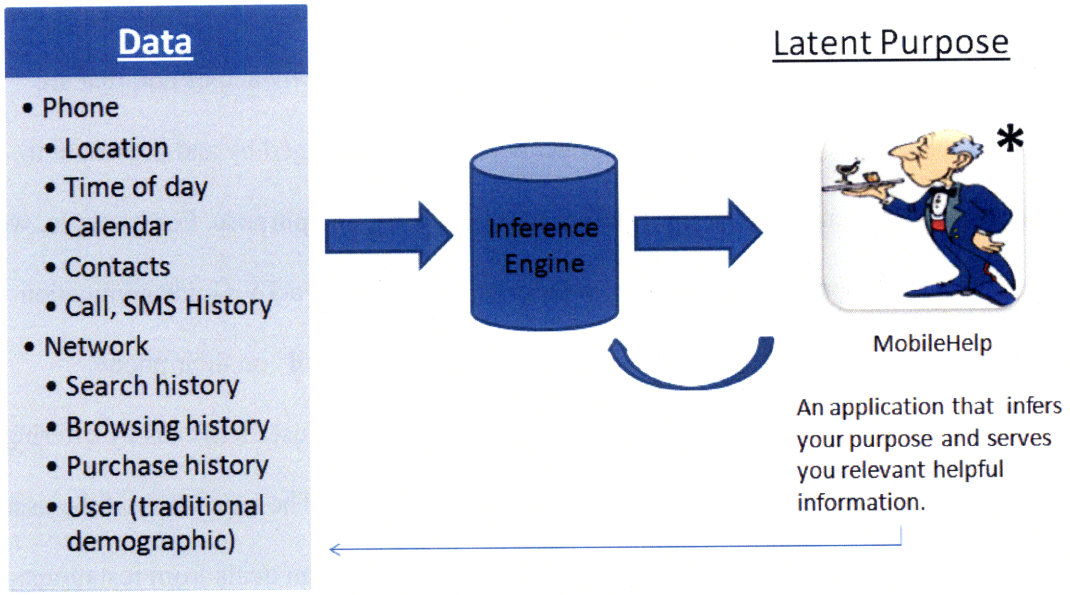


Figure 3: This figure shows the data available on internet-enabled smart phones (specifically, Android phones).

We make extensive use of the phone's data to make inferences on user purpose and cognitive style. We now discuss the details of that process.

4.2.2.1 User Purpose

A large part of this project hinges on our ability to infer user purpose. Although it's been previously stated that this is a static demo, we discuss how a prototype system could infer user purpose in a Bayesian framework. Our simple model consists of purpose (P), time (T), and location (L). From data and surveys, we will estimate $P(L|P,T)$. The whole data gathering process is opt-in. Users who are interested would be incentivized fill out a questionnaire on the FT/Orange site that would tell us about their cognitive style. We would also ask questions such as "How likely will you be in this location if you're having lunch at work?" We can also gather data points for that conditional probability when users change their purpose inside MobileHelp. Users who choose to opt-in would also agree to be in a panel that records their location, purpose, and time, and sends it back to a

central server. An example where this has been done is with the Statistics Bureau of Japan, where people were asked to log their activities (games, gardening, walking the dog, etc.) every 15 minutes (What People in Tokyo are Doing on a Tuesday, 2008).

We have to be careful when treating these probabilities, because in general, location and time are continuous variables. To deal with that, we can quantize the possible values they can take on. For example, we can view time as buckets of hour-long intervals. Two times T_1 and T_2 will be equivalent if $|T_1 - T_2| < \delta$. Unfortunately, this is not an equivalence relation because transitivity does not hold. We can define a similar equality function on locations using Euclidean distance.

Let $E_T(T) = \{T_i | |T - T_i| < \delta\}$, or the set of all times that are within δ of T. The value of δ will have to be adjusted, but a reasonable starting point may be 1 hour. Define a similar equality set for locations $E_L(L)$. Then our new probabilities become $P(E_L(L)|P, E_T(T))$, or the probability of being within a certain physical area given the current purpose and chunk of time we're in. Using Bayes rule, we can solve for $P(P|E_L(L), E_T(T))$ as follows:

$$P(P|E_L(L), E_T(T)) = \frac{P_0(P)P(E_L(L)|P, E_T(T))}{\sum_i P(P_i)P(E_L(L)|P_i, E_T(T))}$$

In our demo, we will implement this form of Bayesian inference with fixed probabilities. That is, at each stage of our scenario, we will have probabilities for each possible combination of times, locations, and purposes stored. In that sense, the inference engine will be used, but not updated. To update the probabilities in the prototype, we can respond to user input. If the user corrects the displayed purpose, we will take that into account as another data point in our history of observations. In the future, that point affects what our computed probabilities are.

4.2.2.2Cognitive Style

We will have an independent component of the Bayesian engine making inferences about cognitive style. This component would be very similar to the previous work we've done with Suruga Bank and follows the scheme from the previous section. As mentioned, initial cognitive style information can come from surveys done on the FT/Orange site, which then get sent to the phone. Clearly, this would have to be done on external servers, hosted either by FT/Orange or MIT.

4.2.2.3Prototype Gittins Engine

The Gittins engine in the prototype will be used to determine which morph and relevant information to show users, given an inferred purpose. There will be two independent components. The first component determines which information the user should see given his purpose. This allows us to learn what type of information the user wants to see for a given purpose. The second component determines which morph of the relevant information to show users. This is the standard Gittins engine we've used in previous web morphing studies.

The Gittins engine needs success metrics in order to learn. In websites, we typically consider the information the Gittins present as "correct" if the user made some sort of "purchase" action (a conversion) such as requesting more information. The Gittins engine here will use that same idea to determine whether the morphs and information it showed the user was good and relevant. However, things are more complicated because we have to be careful not to conflate cognitive style with purpose. Initial thoughts include using the guessed purpose as a conversion. If the user does not change the purpose, then we can guess it's correct. Conversions for morphs may consist of clicking links, using

apps and apps, and interacting with nearby friends. The full details of this have yet to be determined and will require future research.

4.3 Demo Storylines

We now present the storylines for our demonstration. They tell the day-to-day activities of two different people, Jim and Pam. We follow Jim, a married male in his early forties, through a typical weekday. We follow Pam, a single female in her mid twenties, through a fun day Saturday. In both cases, we offer two storylines for two cognitive style combinations: Analytic/Verbal and Holistic/Visual. This gives us a total of four walkthroughs. In each case, we try to emphasize how MobileHelp can make their lives easier.

4.3.1 Jim

Jim is in his early forties, married, and has two children. He lives in Lexington, MA and works for a finance company in downtown Boston, where he commutes to work every day. Here is a typical work day in Jim's life, and how MobileHelp can make his hectic day easier:

Time of Day	Activity	MobileHelp
7:00 AM	Wakes up and prepare for the day.	Realizes he's at home early morning. Offers him Starbucks coupons.
8:10 AM	Begins commuting to work.	Based on time of day, day of week, and the fact he's moving, determines he's commuting and pulls up the day's news for him to read.
8:40 AM	Arrives at work.	Set his status to "Work".
11:30 AM	Heads off to lunch with his coworkers.	Sees it is lunchtime and Jim is out of the office. Gives him a few promotions for nearby cafes.
2:00 PM	Team meeting.	Reads his calendar and notices he's in a meeting, so sets his status to "Meeting" and silences his phone.

5:15 PM	Heads home. On the way back, he does a few searches for TV show names.	Gets his search history and categories them as entertainment. Brings up the TV Guide application.
6:00 PM	Arrives home, turns on the TV, and prepares dinner.	Sets status to "Home".

Since Jim is a working professional, the things that are most relevant to him are new applications and deals. Despite the fact that his day is fairly routine, MobileHelp still finds ways to provide value to Jim, without his actively seeking help.

4.3.2 Pam

Pam is a recent college graduate who lives in Boston. She's an active socialite and is usually busy on the weekends out with friends or meeting new people. Here's how a typical Saturday for her might go:

Time of Day	Activity	MobileHelp
9:00 AM	Wakes up and checks the weather on her phone.	As the phone turn son, sets her status to "Home".
10:15 AM	Calls some friends to arrange brunch.	
11:10 AM	Heads over to brunch in Copley.	Offers her discounts at a few restaurants her friends have frequented. Sets her status to "Out".
1:00 PM	Starts her day's shopping.	Alerts Pam that a few of her friends are nearby.
1:15 PM	Acting on the information, meets up with two friends.	
6:00 PM	Beings looking for a place to eat dinner.	Offers her specials at a bar where people of her demographic tend to go.
6:45 PM	Settles on a bar with her friends and eats dinner.	
8:00 PM	Finishing dinner.	Seeing that she used a coupon a couple hours back, infers that she's done with

		dinner. Pulls up some movie show times and recommends the Fandango application. Also displays information about a nearby jazz performance.
10:00 PM	Decides to see a movie and purchases tickets through her phone.	
12:45 AM	After the movie is out, needs a cab since the T is closed.	Realizes she's out past when public transportation is closed and pulls up a map with nearby taxis and gives her phone numbers
1:00 AM	Successfully hails a cab and heads home.	Sets her status to "Home".

Since Pam is young and single, her life is focused more around her social circle. Thus, she's more interested in friends and deals for places that let her meet new people. This is why MobileHelp recommended the bars and jazz performance to her. For someone of a different demographic (say Jim), it would not have recommended the bar scene, but instead an older or more family-oriented restaurant.

4.4 Features and User Interface

We present the MobileHelp features, which consists of the four main screens on our application: recommended applications, search results, offered deals, and nearby friends. We tried to design the user interfaces consistently so that all the pages of our applications share a common theme and navigation scheme. This isn't always possible because the pages present very different content. For example, the visual version of nearby friends is a map, which is nothing like a grid of application icons. However, we can make the verbal list view of applications look very similar to the list view for nearby friends.

4.4.1 Applications

The applications page lists the applications you may be most interested in based on your purpose. The list contains both applications you own and those you don't. It helps in the process of application discovery, which can be a challenge by itself given the plethora of applications available. The Analytic/Verbal morph shows a list of applications with some descriptive information. The Holistic/Visual morph shows a grid of large icons, much like the iPhone application selector. Applications you already own will be highlighted in such a way to make it obvious, but the grid/list you see will most likely be a mix of applications you do and don't own.

Applications you own will open when clicked. The state of MobileHelp will be saved and can be resumed later. Applications you don't own will bring up the app store when clicked and directly link to the chosen application where you can buy it.

4.4.2 Search Results

The information page shows web search results and other information we think is relevant to you. This is useful since there should be plenty of information the web that's of immediate interest to the user. We do not present the results in a typical way shown in web browsers, but instead in a sanitized list that looks similar to the other pages for consistency.

4.4.3 Deals

The deals page shows deals offered by nearby businesses that may interest you, given your current purpose. This is better than traditional advertising because the ads are much more targeted and entirely user solicited. The Analytic/Verbal morph will show a list of business names with short textual descriptions of the deal. The Holistic/Visual morph will be a grid of business logos (similar to the grid of applications). When a deal is

selected by the user, the screen goes to a new page which provides more information about the deal. This screen shows the deal name and a text description of the exact details of the offer, including location, expiration date, and business contact information to learn more.

4.4.4 Nearby Friends

The friends page tells the user the location and purpose of nearby friends. The Analytic/Verbal version is a scrollable list of nearby friend names, locations, and purposes. The Holistic/Visual version shows a map with pinpoints representing friends. When a pin is tapped, a popup comes up showing the friend's name, picture, purpose, and any text status he or she has set.

4.5 Cognitive Style and Morphs

As mentioned earlier, this demo focuses on Analytic/Verbal and Holistic/Visual cognitive styles. We consider the scenario where someone is eating lunch at Au Bon Pain. For each of the different pages (recommended apps, information, deals, and friends), we show what Analytic/Verbal (left) and Holistic/Visual (right) morphs would look like. Although these mockups were in an iPhone setting, the spirit is the same for Android. (Note that these images were created by Jong-Moon Kim, who graciously agreed to let me use them).

First, we present the apps page.

The Analytic/Verbal morph shows a list of applications, showing their icons, the name, a short description of the app, and whether the user currently owns it or not. The Holistic/Visual morph shows a very simple grid of application icons. When a user selects an icon, more information is shown below. Colored backgrounds around the large icons tell the user whether he currently owns the app or not.

We also have two mocks showing how the user can change his purpose:

There is a dropdown menu that shows other high probability purposes. If none are correct, the user can type the choice he wants into the textbox.

The next morphs are for the info/search results page.

We show search results in a much more organized way than a typical browser, in addition to tying it into your social network. Notice in the Analytic/Verbal morph that each search result shows not only how far away the relevant place is, but also how many of your friends have been to the place. This extends beyond restaurants to any geographic place, or even perhaps websites your friends may frequent. The Holistic/Visual morph shows relevant search results in a map with icons representing the relevant results. It shows the same information that the Analytic/Verbal morph does when an icon is selected.

Next, we compare the deals page.

The Analytic/Verbal offered deals display is similar to the info/search page, except instead of information about the business, we present information about the deal being offered. The Holistic/Visual morph is very different from any morph we've seen before. It is big and to the point. Users can browser different deals by swiping the screen left and right.

Last, we present the friends page:

The Analytic/Verbal version shows a list of friends with their names, pictures, and text statuses. Contrast this with the Holistic/Visual version which shows a map view of where users are, allowing the user to get a broad summarized view of the situation immediately.

4.6. Architecture

4.6.1 The Android Platform

4.6.1.1 Overview

We chose to develop our demo on the Google Android platform for several reasons. The overarching theme of Android is openness. The operating system provides user-level

applications with deep access to phone resources that other development platforms hide. The other main contender for the demo platform is the iPhone. Generally speaking, the iPhone platform is closed and doesn't give application much freedom. The following table contrasts what phone services are available to applications on both platforms.

Utilities	Android	iPhone
Calendar	*gcal*	N
Contacts	Y	Y
Call Log	Y	N
SMS In/Outbox	Y	N
Location	Y	*passive*
Notifications	Y	Y

Table 1: This table compares the features that are available on the Android and iPhone development platforms. It's evident that Android offers many more capabilities than iPhone does, and thus makes a better candidate for a protoype platform.

Even though our demo is strictly canned, we took future prototype prospects into consideration when choosing our development platform. It's clear to see that Android offer us access to all of the phone resources we need to implement a working prototype. Furthermore, Android allows services to run in the background. This is absolutely necessary for a prototype. The iPhone does not currently allow this, although there are talks of providing such a capability in the near future (Siegler, 2009).

4.6.1.2 Core Components

Android development is centered on four core components: activities, services, broadcast receivers, and content providers. We provide a brief overview of the first two components. Each component has a lifecycle it must follow. These stages in the lifecycle consist major phases in the lifetime of a component, from when it was created to when it is destroyed.

Components (except for content providers) are activated by *intents*, which are asynchronous messages that signal an "intent" to do something and carry information that helps that "intent" be acted upon. For example, an intent might notify an application that user wants to edit text, in which case the application can handle the intent by letting the user perform the desired action. As another example, a user might click on web URI, which launches an intent to browse a web page. This intent would be caught by the phone's default browser, causing that application to open and navigate to the requested URI.

4.6.1.2.1 Activities

Activities are visual interfaces for specific tasks the user can perform. For example, one activity might show a list of options for a user to select, or a map the user can scroll around and click on. Activities have three essential states: active, paused, and stopped. An activity is active if it is currently in focus and taking user inputs. It is paused when it is partially visible to the user but not the focus of the user's actions. This can happen when another activity takes up part of the screen (a popup, for example) and takes focus. Lastly, an activity is stopped when it is no longer visible to the user (perhaps because it was superseded by a new activity). Stopped activities retain all state information.

4.6.1.2.2 Services

Services are background processes that run for an indefinite period of time. They have no user interface. For example, a service might quietly poll GPS data and start an activity if anything interesting happens. An application can communicate with a service by binding to it. Services are necessary for a functioning prototype, but not for our demonstration

purposes. For the prototype, the service is a critical component. It is the part of the application always monitoring what the user is doing to detect a change in purpose and offer recommendations.

4.6.2 Prototype

4.6.2.1 Cross-Platform Compatibility and Client-Server Considerations

One of France Telecom's goals is to develop applications that are usable across a wide range of mobile platforms while rewriting as little code as possible to work with each individual platform. This is especially important to them because their network features many different phone types. Thin-client applications, where most of the brains are on a server and the client just renders what the server sends are good examples of this. The extreme example would be to consider the web browser as the ultimate thin client. It is mostly stateless (except for cookies) and features a universally adopted GUI layout language: XHTML/CSS and Javascript.

The cost of this compatibility is functionality. The advantage of customized native phone applications is that they can access the entire range of functionalities provided by each phone's API. This includes things like access to the calendar, SMS logs, call logs, contacts list, calendars, and GPS that traditional thin clients have difficulty getting access to. There has been some work in making GPS available to web browsers, but security issues make this a very tricky proposition.

To aid France Telecom in their goal of cross-platform compatibility, we consider ways of turning MobileHelp into a thin client. The extreme case is to move all the client processing to the server and have the client render (via WebKit, for example) a layout response from the server. Although we may strive for this ideal, we most likely will not

be able to achieve it. MobileHelp depends on several phone features that are only available through the phone's custom API and development environment. These include all of the features mentioned above, but specifically GPS, contact list, and calendar.

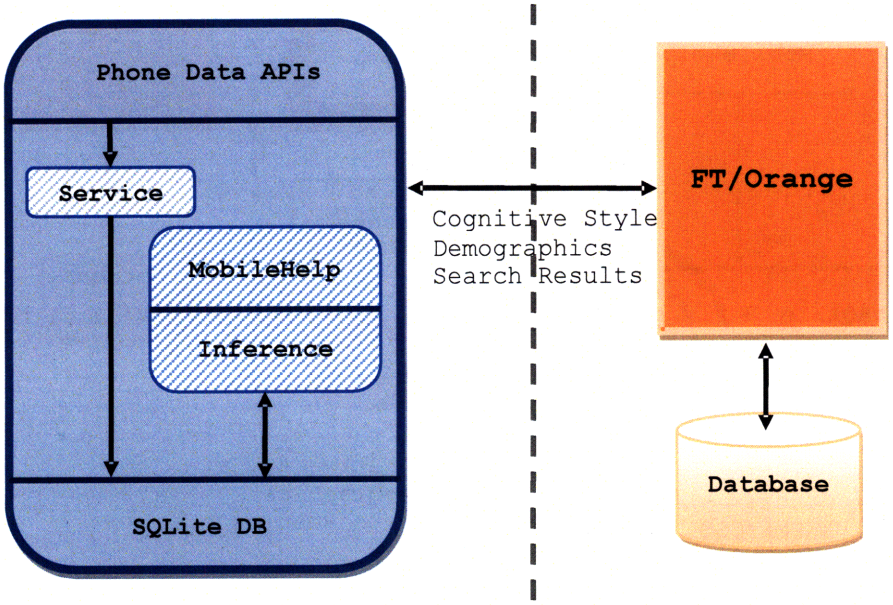
One way to get around this problem is to have the phone send this data over to a central server. Ignoring issues of server load, bandwidth costs, and battery consumption, we could conceivably update a central server with the phone's location and pushes any changes in its contact list or calendar as they happen. Then, the server would have all the necessary information to make a Bayesian inference and return the relevant information back to the phone to show to the user.

Another solution is to create a compromise between a thick and thin client. We leave the engine and custom capabilities on the phone, but design the application such that they are lightweight blackbox systems that can be replicated on different platforms with little work. The GUI would be written with WebKit so that it could remain untouched across platforms. This may make the most sense for a live prototype, since it will be extremely difficult to move everything over to the central server model. However, it is reasonable to change our views from Android Activities to WebViews.

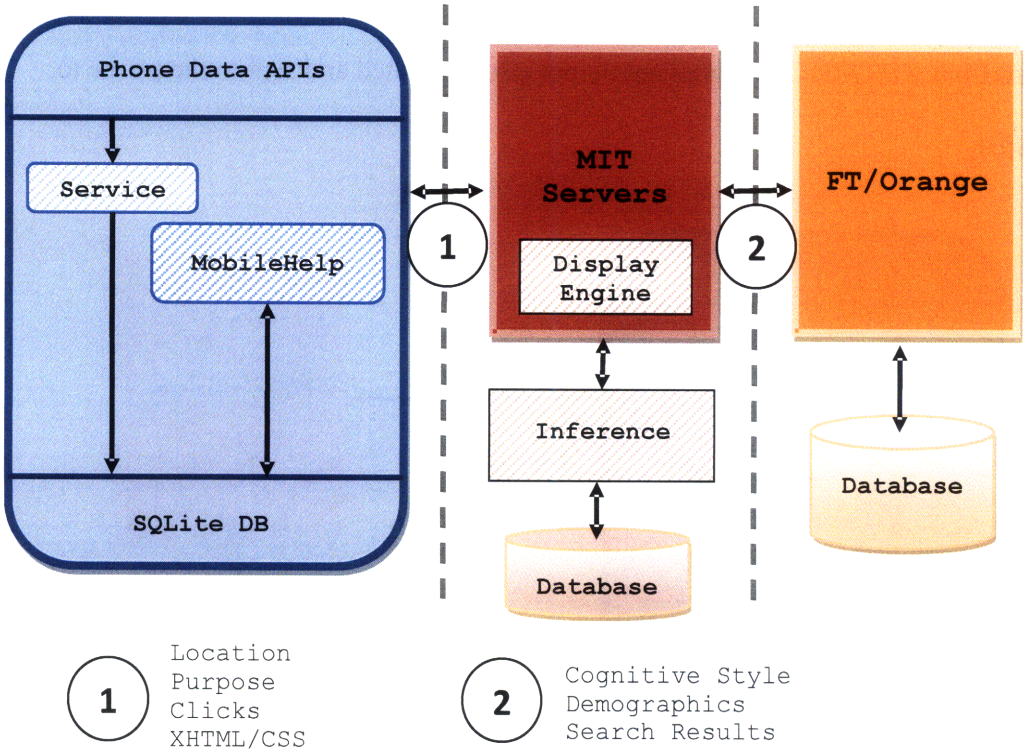
4.6.2.2Architecture

The prototype will be a full-fledged client-server model. There are a couple of possible designs. We present both options and defer a decision until later as our experience may make certain facts more apparent that will affect our choice. The two models differ in where the inference engine is. The thick client model puts the inference engine onto the phone itself, and only communicates with external servers to learn cognitive style and demographic information about users. The thin client model puts the brunt of the work on

external servers (hosted either by MIT or FT/Orange), where not only inferences are made, but XHTML/CSS compliant output also computed and sent to the phone to display.



This shows the thick client that gathers user data and performs inference on the phone. The only exchange with external servers (besides search results) is to ask the FT/Orange servers for information about the user's cognitive style and demographics. Contrast this with a thinclient architecture.



The main difference between these two systems is where the intelligence is located. In the first scheme, the phone is responsible for running the inference engine and creating the display. In the second scheme, external servers get a feed of all of the phone's information necessary to do inference, and sends back XHTML/CSS for the phone to display.

4.6.3 Demo

4.6.3.1Simulating Time

Since this is a static demonstration, we will need to "simulate time" to let our users progress in the storyline. The simplest way to do this is to make each scenario in the storyline a snapshot in time, such that the user can flip forward and backwards in time. A

snapshot consists of the user at a specific time and place with a given purpose. Then, each of the four tabs in MobileHelp will change to reflect this new scenario. As mentioned before, each tab is morphed to the user's cognitive style. Thus, time is not continuous, but treated instead as an ordered set of discrete events. Each event corresponds to one scenario in a user's storyline.

4.7 Demo Specifications

The following section presents a specification for the major components of the demo application. Standard software engineering principles dictate that writing a specification up front and strictly adhering to it is bad practice. Software engineering is an iterative process, and trying to exactly detail every aspect of a system before building it will surely lead to failure. In that sense, consider these specifications as design guidelines and not as dogma. As the application develops and engineering issues arise, they should be dealt with in the most practical manner at the time. In the subsequent section, I will also include performance considerations for developing on the Android platform, and mobile phones in general.

These specifications are by no means exhaustive. Since we are still in the process of finalizing the details of the application, there will be a need to change the design. With that in mind, we write these specifications only highlight certain parts of the application that are unlikely to change.

4.7.1 Constants

We begin by defining some integer constants that will be used throughout the application. We choose integers instead of enumerations because the latter is much costlier in terms of

performance. Note that although the design of these constant is reasonable, the evolving project may require the addition or deletion of constants (for example, as purposes are added and removed).

These constants should live inside the class `Constants`. Constants should be capitalized and declared `public static final`.

4.7.1.1 Cognitive Style

The most efficient way to store cognitive style is to use a bitmask on an `int` (or perhaps a `byte`). We define the least significant bit (1st bit) as the representative for Analytic/Holistic, and the second least significant bit (2nd bit) as the representative for Visual/Verbal. We define Analytic as 0 and Holistic as 1, and Visual as 0 and Verbal as 1. This is convenient because there's a simple integer representation of the current cognitive style, which is just the integer value. This can be trivially saved and read to and from the database. Manipulating the cognitive styles is simple using bitmasks.

Thus, we have the following constants:

```
public static final int CS_ANALYTIC_HOLISTIC_MASK = 1;
public static final int CS_VISUAL_VERBAL_MASK = 1 << 1;
```

In general, cognitive style constants will have the form `CS_*`. The following is an example of how to use the bitmaps to set the cognitive style to Analytic/Visual:

```
int cogStyle = 0;
cogStyle &= ~CS_ANALYTIC_HOLISTIC_MASK;
cogStyle |= CS_VISUAL_VERBAL_MASK;
// cogStyle = 0b00000000 00000000 00000000 00000010 = 0x02
```

4.7.1.2 Purpose

The full list of purposes is undetermined as of the writing of this thesis. The framework for specifying them is simple however. We have monotonically increasing integer constants of the form `PURPOSE_*`, where `*` is from the set `{ "HOME", "WORK", "OUT", "FOOD:, etc … }`. All references in the database specifications to the "home" purpose would refer to `PURPOSE_HOME`, for example.

4.7.2 Data Models

We now discuss the data models in our database. We break down the models into three groups. The first group consists of those necessary for the application, regardless of whether it's the demo or a real prototype. The second group consists of models used solely for the demonstration. The third group consists of models needed for the prototype.

In general, if the value of a field is unknown, it should be set to NULL. The application should check for this.

4.7.2.1 SQLite Considerations

We begin by discussing several subtleties of SQLite, the database on the Android phone. SQLite is a more compact form of a standard SQL database and is thus well suited for mobile platforms. However, this compactness comes at a cost in functionality. For most practical purposes, there aren't many differences between SQL and SQLite, but we list some issues that developers should be aware of.

The first thing to note is that SQLite stores DateTime fields as strings. Therefore, if you want to compare DateTimes in the database, you must store them in ISO 8601 format: `YYYY-MM-DD HH:MM:SS`. Since strings are compared in lexicographic order, following this standard ensures that a given DateTime is less than another DateTime if

and only if the former's string representation is lexicographically less than the latter's. A relevant discussion can be found online at http://groups.google.com/group/android-developers/browse_thread/thread/132ff8ac1671dadd.

The ISO 8601 standard is implemented in Java as follows:

```
private static final String SQL_DATETIME_FORMAT_STRING =
"yyyy-MM-dd HH:mm:ss";
```

This string can be parsed by the DateFormat class. A full SQL utility class which makes use of this can be found in the Appendix. Further information about how SQLite handles dates and times can be found here http://www.sqlite.org/cvstrac/wiki?p=DateAndTimeFunctions. From here on forward, unless explicitly state otherwise, a `DateTime` field in a database will be a `Text` field in the ISO 8601 format.

From the SQLite FAQ states that SQLite only supports INTEGER, REAL, TEXT, BLOB, and NULL. SQLite also does not do type checking on the data you insert into columns, so you are free (although certainly not encouraged) to insert floats in a boolean column, arbitrary length strings in integer columns, etc. Care should be taken to avoid this. In addition, SQL support foreign keys, but does not enforce them in any way.

One performance consideration is that SQLite uses atomic transactions. Each INSERT is considered a transaction, and these take a long time to complete. Therefore, if you are making many contiguous INSERT calls, it may be best to batch them together and surround the multiple INSERT statements with BEGIN ... COMMIT. This amortizes the transaction cost over all of the INSERT statements, which should increase speed dramatically.

4.7.2.2 Core Application Models

The core application models consist of database table specifications that would be needed by any version MobileHelp, regardless of whether it is a demo, prototype, or final product. This includes data about the phone user's cognitive style and demographic information, friends, deals, applications, and businesses offering deals.

4.7.2.2.1 UserInfo

The UserInfo table stores basic demographic information about the phone's owner, as well as information about cognitive style. This is traditionally proprietary information that mobile applications don't have access to. Since we are partnering with a network provider, we have special access to this information. The cognitive style information could be gathered by a survey, the user's action's on the network provider's website, or the user's actions on the phone.

Data	Format	Comments
FirstName	Text	
MiddleName	Text	
LastName	Text	
Age	Integer	
Sex	0/1	0 = Male, 1 = Female
Birthday	DateTime	The time portion of the ISO 8601 standard, if unknown, should be set to 00:00:00
HouseNumber	Integer	Can also be the apartment number
Street	Text	
City	Text	
State	Text	Two letter state code
Zipcode	Text	String is better than integer (consider "11790-2620")
Country	Char[3]	3 letter country code following the ISO 3166-1 alpha-3 standard (http://en.wikipedia.org/wiki/ISO_3166-1_alpha-3)
CognitiveStyle	Text	The **int** representing the user's cognitive style

We need one data model for each main page of our application: applications, deals, and friends. We also have a table for all the businesses that deals might be associated with.

4.7.2.2.2 Applications

We store all of the information we need about an application in order to present it to the user. Note that some information (like Purpose) would have to be manually examined by humans set in a central server for distribution.

Data	Format	Comments
Name	Text	
VersionCode	Integer	Integer code of version (for internal use).
VersionText	Text	A string like "3.1.4" to show to the user (http://developer.android.com/guide/publishing/versioning.html)
Description	Text	
Category	Text	The name of the category under the application store
Purpose	Integer	The purpose this application is associated with
Price	Text	A string like "$3.99"
AppStoreLink	Text	Link to the app store where the user can buy this app
Owned	Boolean	True if owned, else false
LastRecommendedTime	Datetime	Last time this app was recommended

4.7.2.2.3 Businesses

We store information about businesses so that users can find out more information about where their deals are being offered, such as the address to claim it and a phone number to call for more information.

Data	Format	Comments
Name	Text	The name of the business
Latitude	Float	The latitude from GPS

Longitude	Float	The longitude from GPS
PhoneNumber	Text	Pure numbers, no delimiters
StreetNumber	Integer	
Street	Text	
City	Text	
State	Text	
Zipcode	Text	3-letter ISO 3166-1 country code
Country	Text	
Category	Integer	Food, Drinks, Music, Movies, Transportation, Shopping, etc
Icon	Text	Local URI of icon image
Purpose	Integer	The purpose this business is (generally) associated with
LatestDeal	ForeignKey into Deals	Most recent deal offered by this business
LastRecommendedTime	Datetime	Last time this app was recommended

4.7.2.2.4 Deals

One point to keep in mind about our demonstration is that deals should always be valid. That means we need to dynamically generate a time window that would overlap the moment in time the deal was shown. Therefore, the fields highlighted below would not be used in the demo

Data	Format	Comments
Name	Text	The name of the deal
Description	Text	A short description
Image	Text	Local URI for image to display
Purpose	Integer	The purpose this deal is associated with
ValidBegin	Datetime	The valid start time
ValidEnd	Datetime	The valid end time
Business	ForeignKey into Businesses	The business this deal is associated with
UseCode	Text	Something the business supplies us that allows the user to use this deal

4.7.2.2.5 Friends

This tables stores information about friends and friend relationships between contacts and users. In a normal setting where a phone is used and owned by only one person, we don't need to store "friend relationships" since all the contacts are friends with the phone's user. However, this is a demo application with many simulated users. In this case, we have to store all the friends that each simulated user has. This is accomplished with the highlighted field below.

Data	Format	Comments
FriendOf	ForeignKey into UserInfo	Only for the demo
ContactKey	ForeignKey into Contacts	The friend's contact information
LastLatitude	Float	The latitude from GPS
LastLongitude	Float	The longitude from GPS
LastLocationTime	Datetime	The time the last location was recorded
LastPurpose	Integer	
LastPurposeTime	Datetime	Time the last known purpose was recorded
Picture	Text	Local URI for photo

4.7.2.3Demonstration Models

The models required for demonstration store all the scenario information required for each demo user's storyline. This includes a list of demonstrations, and tables that store nearby friends, recommended applications, recommended deals, and relevant information for each scenario in each user's storyline.

4.7.2.3.1 Storylines

We begin with a model that stores a list of all the storylines that a user can try.

Data	Format	Comments
DemoNumber	Integer	The unique ID for this demo/storyline
User	ForeignKey into UserInfo	The user (story) we're demoing
Name	Text	The name of this storyline

Description	Text	A short description of the storyline
DayOfWeek	Text	Monday, Tuesday, …
Photo	Text	Local URI of the person's photo
CognitiveStyle	Integer	The cognitive style of the user for this storyline
NumScenarios	Integer	The number of scenarios in this storyline

Although we store the cognitive style for this storyline, we can conceive of changing it throughout the demo as we please. Next we present a model that stores the scenario information for each storyline.

4.7.2.3.2 Scenarios

This model stores each "slide" of the "slideshow" that makes up one storyline.

Data	Format	Comments
DemoNumber	Integer	The unique ID for this demo
SequenceNumber	Integer	The order of this row in the given user's demo
Time	DateTime	The time of day this demo is happening (The yyyy-MM-dd should be ignored when read)
Purpose	Text	The user's purpose in this scenario
Description	Text	Text description of the scenario

We now present the models for recommended applications, deals, and nearby friends and relevant information.

4.7.2.3.3 RecommendedApps

This table simply joins a scenario with an application we recommend. We use join tables because they allow us to associate an arbitrary number of applications for any scenario. The software should determine the order in which to show the apps (alphabetically, for example).

Data	Format	Comments
RelatedScenario	ForeignKey into Scenario	The scenario where we are recommending this app

AppKey	ForeignKey into Apps	The recommended app

Likewise, the recommended deals table simply joins a scenario with a deal.

4.7.2.3.4 RecommendedDeals

Data	Format	Comments
RelatedScenario	ForeignKey into Scenarios	The scenario where we are offering this deal
DealKey	ForeignKey into Deals	The deal we are offering

4.7.2.3.5 FriendsNearby

This model is also a join table, except we include some extra information about the nearby friend.

Data	Format	Comments
RelatedScenario	ForeignKey into Scenarios	The scenario where we are recommending this app
FriendKey	ForeignKey into Friends	The friend that's nearby
Purpose	Integer	The friend's purpose
Status	Text	The friend's text status
Latitude	Float	The GPS latitude
Longitude	Float	The GPS longitude

4.7.2.3.6 RelevantInfo

This table holds lists of relevant information for each scenario. Once again, it is up to the software to determine what order to display this information.

Data	Format	Comments
RelatedScenario	ForeignKey into Scenarios	The scenario this information pertains to
Title	Text	The title of the info snippet
Text	Text	The info text body
URI	Text	The link for the web resource

4.7.2.4Prototype Models

This section includes database models that would be required by a working prototype to function. The main parts consist of required inference information about the user. Specifically, we store a history of past events that update our Bayesian engine, search history, and data aggregated and processed from the phone's data sources (calendar, contacts, SMS in/outbox, call logs, location, notifications, etc.).

4.7.2.4.1 Inference

The barebones essentials for an inference engine are history of user time and position coordinates. If we have the user's purpose at the given time, we store that also.

Data	Format	Comments
Latitude	Float	The latitude from GPS
Longitude	Float	The longitude from GPS
Timestamp	DateTime	The time this information was recorded
CognitiveStyle	Text	The cognitive style if the user set it.
TimeOfDay	0-1439	Minute of day
DayOfWeek	1-7 for Mon-Sun	Day of week
TypeOfDay	0/1	0 = Weekday, 1 = Weekend
Day of Month	1-31	Very high entropy (provides almost no information. We may not even want to use this).
Month	1-12	Month of year
Year	4 digit year #	

Note that in addition to the GPS latitude/longitude coordinates, the cognitive style (if the user set one), and the date and time the information was recorded, we also store summary information about the timestamp (highlighted yellow). This summary information would be invaluable for a Bayesian inference engine to use. Otherwise, the engine would have to recalculate this information every time it did a Bayesian update, which would be computationally infeasible.

4.7.2.5.1 Search

Our hypothetical prototype would also store search history, something we would get from the network provider. The Bayesian engine would also make use of this information in updating cognitive style and purpose. Search terms are stored token by token to ease some of the work of the Bayesian engine, which would most certainly need to tokenize the search results. This allows the engine to make direct search queries into the database, which can take care of things such as counts.

Data	Format	Comments
SearchID	UID	Not unique. We use this to join individual tokens of a search
KeywordOrder	Integer	The index of this keyword in the search phrase. First index is 0.
Keyword	Text	The search term
Latitude	Float	The latitude from GPS
Longitude	Float	The longitude from GPS
Timestamp	DateTime	The time the search was made

The above information would be enough for a very basic simple search inference engine. We also include other information worth considering putting into an inference model. They consist mostly of summaries of search and SMS history, and calendar information. In terms of summary information, the idea is to associate certain words with purposes. For example, we could associate restaurant names or food types with food, transportation terms like "MBTA" or "train schedule" with traveling, etc.

Data	Format	Comments
Calendar event @ this time	Integer	0 = None, 1 = meeting, 2 = food, 3 = class, etc.
# SMS in last hour with "work" words	Integer	
# SMS in last hour with "play" words	Integer	
…		
# searches in last hour	Integer	

with "food" words		
# searches in last hour with "shopping" words	Integer	
…		

4.7.4 Important Classes

4.7.4.1 Datastore

The Datastore is the data abstraction layer. All data access, whether local or across the network, should happen through the Datastore. This is for two reasons. The first is that it provides a good abstraction. Activities requesting data should not have to go to different sources for the data. The second reason is that centralizing data access allows the Datastore to cache both local and network data. The Datastore should either be a singleton or static class.

The Datastore class should have a private internal class named `DbAdapter`. The purpose of this class is to provide a database abstraction for Datastore to user internally when querying the database. We now discuss some details of the Datastore and DbAdapter.

4.7.4.1.5 Context

There is an important Android abstract class named `Context`. This class allows applications to access application-specific resources such as the database. The Datastore must be given a context before any data access is done. This is an invariant that must be maintained (and so should be checked before performing any operation that requires a context). The class Activity is an indirect subclass of Context (the inheritance hierarchy is `Context` → `ContextWrapper` → `ContextThemeWrapper` → `ActivityActivity`). Therefore, when an activity starts, one of the first thing it should do is pass *itself* (`this`)

as a context to the Datastore. This context should then be passed into DbAdapter. This allows DbAdapter to call `context.openDataBase(...)` to retrieve the proper database.

4.7.4.1.6 DbAdapter

The `DbAdapter` should take a `Context` parameter in its constructor. From that, it can get a `SQLiteDatabase` object from `context.openDataBase(...)`.DbAdapter should expose open/close calls and provide an API for accessing data from the database. The SQLiteDatabase class is mostly used via its query, execSQL, and update methods. The `SQLiteOpenHelper` class should be useful for interacting with the actual database. It has helpful functions that are called when the database is created for the first time (one should put table creation code in here). It will need to define a database name (string) and version (int) in order to access the database.

It would be too difficult to exhaustively list the methods the DbAdapter should provide, but the general principle is to have create/update/remove methods for each data model. DbAdapter also needs to have a list of table creation queries. A good format to adhere to is given by the following example:

```
/** Every table has a "last modified" field */
private static final String MODIFIED_FIELD = "modified";

/** The fields for the Events table */
private static final String USERINFO_TABLE = "userinfo";
private static final String USERINFO_ID = "uid";
private static final String USERINFO_FIRSTNAME = "firstname";
private static final String USERINFO_MIDDLENAME = "middlename";
private static final String USERINFO_LASTNAME = "lastname";
private static final String USERINFO_AGE = "age";
private static final String USERINFO_SEX = "sex";
private static final String USERINFO_BIRTHDAY = "birthday";
private static final String USERINFO_STREET_NUMBER =
      "street_number";
private static final String USERINFO_STREET = "street";
private static final String USERINFO_CITY = "city";
private static final String USERINFO_STATE = "state";
```

```
private static final String USERINFO_ZIPCODE = "zipcode";
private static final String USERINFO_COUNTRY = "country";
private static final String USERINFO_COGNITIVE_STYLE =
      "cognitive_style";

/** The SQL insert fields for the UserInfo table */
private static final String USERINFO_INSERT_FIELDS =
      "(" + StringUtils.join(",", Arrays.asList(new String[] {
          USERINFO_ID,
          USERINFO_FIRSTNAME,
          USERINFO_MIDDLENAME,
          USERINFO_LASTNAME,
          USERINFO_AGE,
          USERINFO_SEX,
          USERINFO_BIRTHDAY,
          USERINFO_STREET_NUMBER,
          USERINFO_STREET,
          USERINFO_CITY,
          USERINFO_STATE,
          USERINFO_ZIPCODE,
          USERINFO_COUNTRY,
          USERINFO_COGNITIVE_STYLE})) + ")";

   /** The prefix for insert queries into the UserInfo table */
   private static final String USERINFO_FULL_INSERT_PREFIX =
       "INSERT OR REPLACE INTO " + USERINFO_TABLE + " " +
           USERINFO_INSERT_FIELDS + " VALUES";

   /** Create UserInfo table */
   private static final String CREATE_USERINFO_TABLE =
       "CREATE TABLE `" + USERINFO_TABLE + "` (" +
       "`" + MODIFIED_FIELD + "` NOT NULL DEFAULT
                                 CURRENT_TIMESTAMP, " +
       "`" + USERINFO_ID + "` INT UNSIGNED NOT NULL ," +
       "`" + USERINFO_FIRSTNAME + "` VARCHAR(64) NOT NULL ," +
       "`" + USERINFO_MIDDLENAME + "` VARCHAR(64) NOT NULL ," +
       "`" + USERINFO_LASTNAME + "` VARCHAR(64) NOT NULL ," +
       "`" + USERINFO_AGE + "` INT UNSIGNED ," +
       "`" + USERINFO_SEX + "` INT UNSIGNED ," +
       "`" + USERINFO_BIRTHDAY + "` TIMESTAMP ," +
       "`" + USERINFO_STREET_NUMBER + "` INT UNSIGNED ," +
       "`" + USERINFO_STREET + "` VARCHAR(128) ," +
       "`" + USERINFO_CITY + "` VARCHAR(128) ," +
       "`" + USERINFO_STATE + "` VARCHAR(64)" +
       "`" + USERINFO_COUNTRY + "` CHAR(3) ," +
       "`" + USERINFO_ZIPCODE + "` VARCHAR(16) ," +
       "`" + USERINFO_COGNITIVE_STYLE + "` INT UNSIGNED ," +
       "PRIMARY KEY (`" + USERINFO_ID + "`))";
```

This creates the UserInfo table (specified earlier) and provides the prefix/skeleton for an `INSERT` or `UPDATE` query for later use. `StringUtils.join(...)` is a method from a utility class that concatenates the given array of elements by the given string. Note that we

define `VARCHAR` fields. These are treated as `TEXT` fields in SQLite. `TIMESTAMP` is treated as `DATETIME`, which is analogously treated as `TEXT` by SQLite. The `MODIFIED_FIELD` which is defined as `DEFAULT CURRENT_TIMESTAMP` creates a timestamp that is always set to the last date and time each row was created/modified. This could be useful (for caching, for example) and should be included as a field in every table.

4.8 User Interface Design

We now discuss the user interface design. Recall that we are presenting Analytic/Verbal and Holistic/Visual cognitive styles for this demonstration. We have designed two distinct morphs for each screen in our application that appeals to each pair of cognitive styles. Regardless of whether we're showing friends, recommended applications, or deals, Analytic/Verbal users will get vertical lists of options with text descriptions. Holistic/Visual users will get a map for friends and tables of icons for applications and deals with almost no text.

Since activities in Android typically correspond to one screen, we will need several activities. At a high level, we need one screen/activity (the demo selector) to let users choose which storyline they want to follow, one screen/activity to show lists of friends/apps/deals, one map to show friends locations for Holistic/Visual users, and one screen/activity to show grids of application and deals icons.

Android eschews traditional Java layout schemes such as Swing and AWT in favor of XML layout. We give a brief overview of this system and then talk about the different types of views (map, list, and table) (Murphy, 2009).

4.8.1 XML Layout

The purpose of laying out views in XML is to make clearer the physical structure of a view. It provides separation of concerns of layout and functionality. The layout should be in the XML, and the functionality should be in Java code.

When an Android project is loaded inside Eclipse with the Android plugin installed, the developer will see a `res` folder with two thee subfolders: `layout`, `values`, and `drawable`.

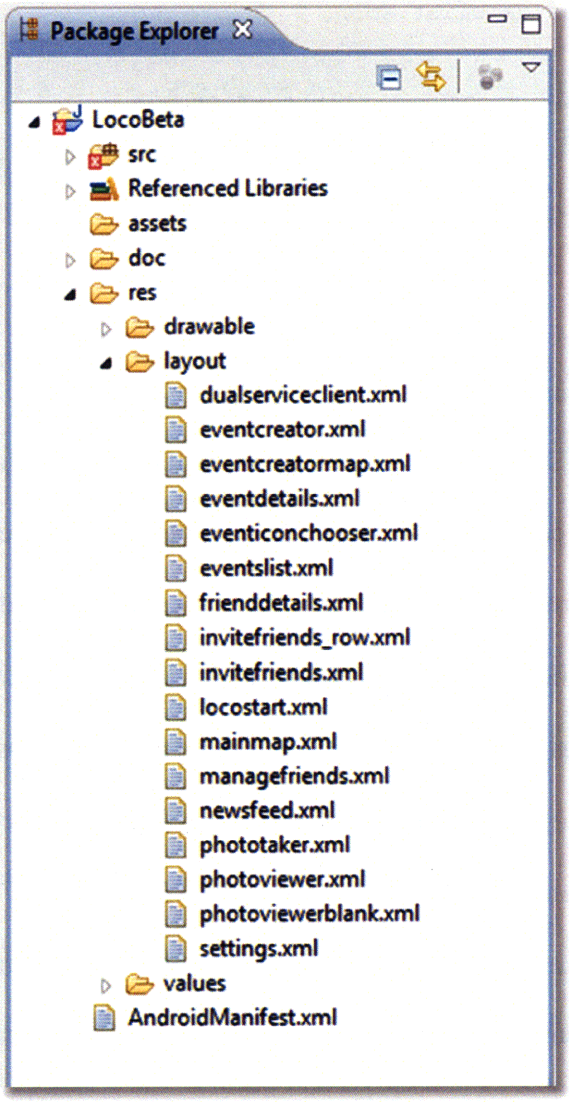


Figure 4: The default folders inside an Android project. The expanded res/layout folder shows XML layouts for activities.

A large share of the GUI work will take place within the `layout` and `values` folders. The XML layout files under `res/layout` define the structure of the view. The `values` folder is for constants, and `drawable` is for images. We explain how to use XML layout via an example.

4.8.1.1 Lists

Consider laying out a vertical list of options (i.e., a simpler version of our list views) in a file called `res/layout/demolist.xml`.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
      <ListView
            android:id="@android:id/list"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
</LinearLayout>
```

This creates a simple vertical list. The list can be filled with arbitrary data, which can be rendered with custom user-defined views. For example, if we wanted each list element to show information about a nearby friend, we define an Adapter (which is a class that extends `BaseAdapter`). This adapter (call it `FriendsListAdapter`) would implement a method called `getView(...)` which returns an object of class `View` (call it `FriendsListRowView`) that can be rendered by Android to show the desired friend information. `FriendsListRowView` could create any type of view. For example, it could create a `LinearLayout` with a horizontal orientation, where the leftmost element is a picture and the element to its right is a `TextView` with the friend's name. It would add

this linear layout for display by calling `addView(...)`, after which that linear layout becomes a row in the master linear layout.

For simplicity, we show how use this layout to fill the list with strings using an `ArrayAdapter`:

```
public class ListViewDemo extends ListActivity {

    /** The strings to show in our list view */
    private final String items[] = {"foo", "bar", "baz", "apple"};

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.demolist);
        setListAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_expandable_list_item_1,
                items));
    }
}
```

This fills the vertical list created by `demolist.xml` (referenced as `R.layout.demolist`) with the strings `"foo", "bar", "baz", "apple"`.

4.8.1.2 Tables

The `TableLayout` class in Android mimics HTML tables in many ways. A table can have multiple rows and each row is a subclass of `LinearLayout`. Rows can span across multiple columns, just like in HTML. One potential disadvantage of using tables is that since each row inherits from `LinearLayout`, elements inside the table are not "selectable". This does not mean that they aren't clickable, but if for some reason in the future, we wanted to let the user "select" an item in the table, we would have to switch to a customized `ListView`.

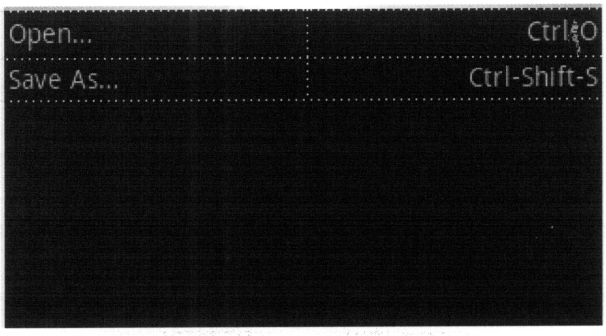
Consider this simple `TableLayout` named `demotablelayout.xml` which creates a table with some rows and columns with text inside.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">

    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Open..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-O"
            android:gravity="right"
        android:padding="3dip" />
    </TableRow>

    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Save As..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-Shift0S"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
</TableLayout>
```

This creates a simple table that looks the following (without the dashed lines):



To display this in an activity, we simply call `setContentView` on this resource inside the `onCreate` method of an activity.

```
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.demotablelayout);
}
```

The cell contents can be filled with whatever. In our case, it should be a custom view that displays an image that reacts to user clicks.

4.8.1.3Maps

Using maps in Android is slight more complicated than other views. First, we must include the Google Maps library, since it's not a standard part of the Android library. This can be done by declaring it in Android Manifest file, which is basically a file that presents essential information about the application to the Android operating system. In other words, it's the major configuration file for each Android application.

We also need to grant the application access to the internet so Google maps can load. Therefore, we place the following inside the manifest file:

```
<application>
    ...
    <uses-library android:name="com.google.android.maps" />
    ...
</application>

<manifest>
    ...
    <uses-permission android:name="android.permission.INTERNET" />
    ...
</manifest>
```

The next step is to define a simple `MapView` XML layout file (name it `demomapview.xml`):

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/demomapview"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <com.google.android.maps.MapView
        android:id="@+id/mapview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:clickable="true"
        android:apiKey="Your Google Maps API Key" />
```

```
</RelativeLayout>
```

`Clickable` is set to true so that the user can interact with the map. Also notice that you must enter an `apiKey`. To get an API key, apply for one here: http://code.google.com/android/add-ons/google-apis/mapkey.html. Now create a `DemoMapView` class that extends `MapActivity`.

```
public class HelloMapView extends MapActivity {

    /** Required to override this. */
    @Override
    protected boolean isRouteDisplayed() {
        return false;
    }
}
```

This should start a map that lets you pan around.

Chapter 5: Contributions and Future Work

This thesis laid out the design and specifications for a demonstration and prototype application called MobileHelp. We determined the best mobile platform to develop based on specifications of what features we need and were available. We discussed how a Bayesian inference engine and Gittins index could be used in a mobile setting to not only infer cognitive style, but also user purpose. This has not previously been tried before.

In the design phase for the demonstration, we created scenarios that would exhibit the potential power a mobile application could unlock from inferring user purpose, and built mocks showing what the user experience would be like. Specifications were created for data flows in both a static demonstration and live prototype with server interactions. A basic primer on the aspects Android development relevant for eventual implementation

was also provided. In the design phase for the prototype, we developed two possible client-server configurations each with their different advantages and disadvantages.

Future work consists of crystallizing the details of the user interface and experience, and implementing the demonstration application we've outlined. Eventually, we would hope to see this move from the demonstration to prototype stage where information with live information, both from the network and phone, is being used to present the user with helpful suggestions.

Chapter 6: Bibliography

Eagle, N., Pentland, A., & Lazer, D. (2009). Inferring Social Network Structure using Mobile Phone Data. *Proceedings of the National Academy of Sciences* .

Farrahi, K., & Gatica-Perez, D. (2008). What did you do today?: discovering daily routines from large-scale mobile data. *Proceeding of the 16th ACM international conference on Mulitmedia*, (pp. 849-852).

Fung, S. (2008). *User Adaptive Web Engine: A Marketing Application in E-Commerce.* Massachusetts Institute of Technology.

Hauser, J. R., Urban, G. L., Liberali, G., & Braun, M. (2009). Website Morphing. *Marketing Science* .

Lee, C. (2008). *User Adaptive Web Morphing: An Implementation of a Web-based Bayesian Inference Engine with Gittins' Index.* Massachusetts Institute of Technology.

Murphy, M. L. (2009). *The Busy Coder's Guide to Android Development.* CommonsWare, LLC.

Riding, R., & Rayner, S. (1998). *Cognitive Styles and Learning Strategies.* David Fulton Publishers.

Siegler, M. (2009, May 15). *Apple Is Indeed Talking About Opening iPhone Background Tasks*. Retrieved May 17, 2009, from TechCrunch: http://www.techcrunch.com/2009/05/15/apple-is-indeed-talking-about-opening-iphone-background-tasks/

What People in Tokyo are Doing on a Tuesday. (2008, October 22). Retrieved May 15, 2009, from Information Aesthetics: http://infosthetics.com/archives/2008/10/tokyos_statistics_right_now.html