NTNU
Det skapende universitet

A Predictive Remote Control for Smart Homes

Audun Gevelt

Master i informatikk
Innlevert: juni 2015
Hovedveileder: Asbjørn Thomassen, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Audun Gevelt

A predictive remote control for smart homes

Master Project, spring 2015

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

Abstract

In this thesis we improve upon the traditional smart phone remote control system for smart homes by adding a predictive component to the user interface. Our goal is to minimize the time it takes for the user to interact with smart home devices through the remote control. To achieve this goal, we locate and compare two prediction algorithms with regards to keeping the system fast and battery-friendly. Additionally, we explore the capabilities of the Android user interface to utilize prediction in a way that minimizes interaction time. We find that we can best utilize prediction in the user interface by keeping the most likely devices on the lock screen, through the use of Android's expanded notification functionality. A separate app screen is accessed when the device can not be found in the expanded notification.

To validate our system, we build a model of our system and a separate commercial system, which we then compare with regard to the time metric. The results show that our system is faster in most cases.

Our work proves that with little extra effort, it is possible to decrease the user interaction time by several seconds in a smart phone remote control system for smart homes.

Preface

This thesis is a project thesis conducted at NTNU. I wish to thank my supervisor Asbjørn Thomassen, for his guidance and support throughout the process; Luan 'Kappa' Tran, for dragging me to the gym; Erik Lothe and Karsten Kjensmo, for their valuable feedback on the report; and all my friends, for their support.

Audun Gevelt
Trondheim, June 8, 2015

Contents

List of Figures

List of Tables

Glossary

ALZ Active LeZi. v, vii, 7, 10–13, 16–18, 20, 40, 42, 45–47

CDU Control and Display Unit. 26, 27

GOMS Goals, Operators, Methods, Selection rules.. 21, 22, 26, 27, 37

IoT Internet of Things. 1

KLM Keystroke-Level Model. vii, 22, 25, 26, 37, 45

NGOMSL Natural GOMS Language. 3, 4, 22, 25, 31, 32, 38, 45, 46

PPM Prediction by Partial Match. 16, 17

SPEED Sequential prediction via enhanced episode discovery. v, 12, 13, 16–18, 45, 46

TDAG Transition Directed Acyclic Graph. 11, 12, 17, 18

Chapter 1

Introduction

The decreasing cost and miniaturization of networked computer chips make it more feasible to embed them into continuously cheaper objects. This is one of the key factors behind a growing trend, called the Internet of Things (IoT). The Smart Home field is closely related to the IoT. It is driven in large part by the promise of increasing comfort and energy efficiency in homes.

As we add networking capabilities to the objects around us, we gain a more fine-grained control of our environment. By increasing the contact area intersecting the physical and the computational world, we amplify the potential of software to reason about, and interact with, the physical world.

This control can be harnessed to automate menial tasks, gather useful data, and make more informed decisions. When our programs can access information about the objects that surrounds us, we in turn, gain the ability to reason about our environment to a much greater degree.

Several vendors have already entered the smart home sector. Most of them use smart phones as a remote control unit for their smart home solution. In addition, many new smart home devices have open standards for communicating with other devices, yielding the potential to control all the devices in the house from a single smart phone app.

Apps are the fundamental way to expand the feature set of a smart phone. They are programs that can be started in much the same way you would start a program from the desktop on your computer. The problem with this approach is that if you only want to perform a small task, having to open an app is a significant overhead compared to the time it takes to complete the task if the app is already active on the phone.

The user already knows what he or she wants when they reach for the remote control. Therefore, the goal of a smart phone remote control must be to minimize

the time spent by the user trying to find the right thing. In the regular app model, too many of the required swipes and button presses are incidental to what the user set out to accomplish.

In the context of a smart home, think of a light bulb in your bedroom. To turn it off manually, you just have to walk over to wherever the light switch is situated, and flip the switch. This is usually faster than pulling out your smart phone, finding the correct app, waiting for the app to load, then finding and pressing the light bulb icon. As a party trick, turning off your lights from your phone might still have its merits. If we want it to replace the light switch on the wall though, it will have to be faster for the user than getting off the couch and flipping the switch on the wall.

Fortunately, we have not yet exhausted the capabilities of the smart phone as a remote control. In the next section, we present a small overview of the motivation for this thesis.

1.1 Motivation

This thesis is motivated by a desire to create a better remote control system for smart homes.

Most remote control systems that exist on mobile phones exist as just another app, with little to no regard for what a user wants from a remote control system. There are some metrics that can and should be optimized for, most important is minimizing user interaction time. When a user interacts with a remote control, he or she already has a goal in mind, and the job of the remote control is to minimize the time required for the user to fulfill his or her goal.

We can optimize for time by simultaneously considering how prediction algorithms can aid the user in finding the right device, and finding a user interface representation that maximizes the utility of such an algorithm.

To create a system that can run on a smart phone, we need to emphasize the run-time and computational complexity of our solution. Since the system runs locally, it has to scale well with the data.

Lastly, we want to create a system that will work primarily on the Android platform.

1.2 Scenarios

The following scenarios shows how the our system can be used to reduce the interaction time.

Finn is going to sleep and wants to turn off the lights in his bedroom. He has already turned off the other lights in his house, which is stored in his recent

history. His smart remote app uses this history to calculate the most likely actions based on the model it has built from his previous interactions, and displays a "Bedroom Light" item along with the other most likely devices on the lock-screen. Finn then taps the "Bedroom Lights" item, which turn off the lights in his bedroom. The system stores this action in the history, updates the model, and displays the most likely devices based on the new information.

Janne wants to turn on the TV as she enters the house after a long day at work. The last thing she did before she left the house was to turn off all the lights. Since Janne normally either turns on her TV or goes to make a cup of tea when she comes home from work, her smart remote app displays "TV" on the top of the list of likely action, just over "Kitchen Light". She taps "TV" and goes to make herself a cup of tea. Since the prediction algorithm updates the most likely items continuously, the item at the top of the list is now "Kitchen Light".

1.3 Goals

Goal *To design a fast smart phone remote control system for smart homes by adding a predictive element.*

Our goal is to improve on smart phone remote control systems for smart homes by combining a sequential prediction algorithm with a new user interface design to reduce the time it takes the user to interact with smart home devices.

Sub-goal 1 *Compare two sequential prediction algorithms with respect to our mobile architecture.*

We compare two algorithms with respect to how well they scale on a platform that is battery-operated and has limited computational resources.

Sub-goal 2 *Design and model a user interface that minimizes user interaction time by using Natural GOMS Language (NGOMSL).*

We build a user interface model to minimize the time it takes the user to interact with our system, focusing on the Android platform.

Sub-goal 3 *Compare our design against a commercial smart home remote control app by using NGOMSL.*

Lastly, we compare our model against a commercial smart home remote control app by using NGOMSL to determine if our design is faster to interact with.

1.4 Research Method

The goals described in the previous section will be approached in the following manner. First, we will expand on the desired algorithm characteristics. We then explore and compare two algorithms that fit our criteria, focusing on keeping the battery drain as small as possible.

Second, we then explore the user interface possibilities afforded by the Android platform to find the best user interface representation that respects the time of the user. We then create a mock-up user interface design and a NGOMSL model that utilizes the Android user interface.

Lastly, we compare the NGOMSL model of our system to a baseline smart remote app, with respect to the time it takes to accomplish a task for the user in each system.

1.5 Thesis Structure

In chapter 2 we explore two different algorithms for sequential prediction that fit our criteria. Then, in chapter 3, we look at a system for modelling a user interface for our system and explore the user interface capabilities of the Android platform.

In chapter 4 we create a model of a system that minimizes the time spent by the user, through merging user interface design with sequential prediction.

Ultimately, we want to know if prediction can improve on existing smart home remote controls. We accomplish this by comparing our proposed system using NGOMSL to a commercial remote control app in chapter 5. Finally, we summarize our results in chapter 6.

The appendices contain the models and NGOMSL model executions traces for our prediction app and the commercial app that is used as a baseline.

Chapter 2

Algorithms

Algorithms that have been proposed for smart homes come primarily from the fields of data compression [Gopalratnam and Cook, 2007], artificial neural networks [Mozer, 1998], fuzzy logic [Anagnostopoulos and Hadjiefthymiades, 2009], Bayesian statistics [Wu and Fu, 2012], Markov logic networks [Wu and Aghajan, 2011] and machine learning. However, only a subset of these are fit to run on a smart phone architecture. We want to use an algorithm that satisfies the constraints we inherit from a system where energy consumption carries a high premium. Our algorithm should be accurate, but it is also important that it does not gut the battery life, which is somewhat of a cardinal sin when it comes to smart phone processes.

The algorithm needs to be capable of online learning, incrementally building the model on which it draws its predictive power. As the user interacts with the devices, we need to update the model to reflect changing patterns of behaviour, if this model is continuously updated, then we do not have to worry about the model getting out of date. An offline prediction algorithm would need to recalculate the model fairly often to stay accurate, which is too costly without tweaking additional parameters.

So we want an online sequence prediction algorithm, that draws as little power as possible, and has high prediction accuracy. Additionally, we will focus only on the binary on and off state of the devices, to simplify the initial model. We further expand upon the desired characteristics of our model in the next section.

2.1 Desired Algorithm Characteristics

[Sun, 2001] characterizes learning models along several major dimensions. We expand upon some of these dimensions with the desired characteristics of our

sequence prediction algorithm.

- Learning paradigm

  Supervised, unsupervised, reinforcement or knowledge based.

  We do not want the user to find the system annoying. The system must therefore be able to learn without feedback from the user. We assume that we are able to gather information about the state of the devices in the smart home without querying the user. As a result, our system only depends on the sequence of state changes in the environment.

- Implementation paradigm

  Neural networks, lookup tables, or symbolic rules.

  Any system that is going to perform on a mobile system must take into consideration the battery drain resulting from computation. The system must effectively balance computational complexity against the utility of the system. Most importantly, the model must update quickly, which rules out slower models. We want an online prediction system.

- Deterministic or probabilistic

  Can the next element be determined.

  Whether human behavior is deterministic or not is out of scope for this thesis. Since we cannot expect to know which device the user will interact with next, the algorithm needs to be probabilistic.

- Markovian or non-Markovian

  Does the Markov property hold for the world.

  The Markov property is assumed to hold for our world, since we have to predict solely based on a sequence of previous events.

- Closed-loop or open-loop

  The algorithm can only predict using the previous history of events in the smart home. This leads to a closed-loop system.

- Action and action policy

  If the model takes action, then is the action policy deterministic or stochastic.

  The model recalculates the most likely devices every time the user interacts with a device. These predictions are then used to update the user interface.

- Applicable domains

  Where can this model be applied.

  We want a model that can be applied to smart home data first and foremost. Preferably something that can take into consideration some of the special characteristics of the data, like time or device state.

2.2 Active LeZi

One algorithm that fits our criteria is the ALZ algorithm. This algorithm was first presented in [Gopalratnam et al., 2003] as an algorithm based on the LZ78 family of compression algorithms, which gives it very good run-time characteristics.

Consider a text being run through a compression algorithm. The goal is to compress the text down to its minimum size while still retaining the information in that text. In essence, this can be thought of as predicting future sequences based on the past observed history.

A compression algorithm consists of both an encoder and a decoder. This means that any sequence encoded by the algorithm has to be able to be decoded into the same sequence by the decoder. For prediction however, we are not interested in being able to retrieve the original sequence. This allows us to shed some constraints on how much of the information contained in the sequence we can use to build a predictor model.

As an example, let us consider the following sequence:

aabcdeeeaaadabccdaaa

If we encode this sequence using the LZ78 compression algorithm [Ziv and Lempel, 1978], we end up with steps seen in table 2.1 and the tree seen in figure 2.1. This tree is an approximation of a order-k-1 Markov model – where k is the longest LZ phrase seen – except it is sparser, since it does not utilize information that crosses phrase boundaries.

Information theory states that optimal predictability occurs if the order of a predictor model grows at a rate approximate to the entropy rate of the sequence source. [Feder et al., 1992] shows that Markov predictors based on the LZ78 compression algorithm family attains optimal predictability. This, along with the fact that LZ78 is an incremental parser, which means it has real-time properties, are the theoretical underpinnings for the ALZ algorithm.

2.2.1 Learning Procedure

ALZ [Gopalratnam et al., 2003] supplements the LZ78 dictionary approach with a sliding window, which expands in relation to the length of the longest LZ

Step	Position	Dictionary	Output
1	1	a	(0,a)
2	2	ab	(1,a)
3	4	c	(0,c)
4	5	d	(0,d)
5	6	e	(0,e)
6	7	ee	(6,e)
7	9	aa	(1,a)
8	11	ad	(1,d)
9	13	abc	(2,c)
10	16	cd	(4,d)
11	18	aaa	(9,a)

Table 2.1: LZ78 simulation of the sequence “aabcdeeeaaadabccdaaa”.

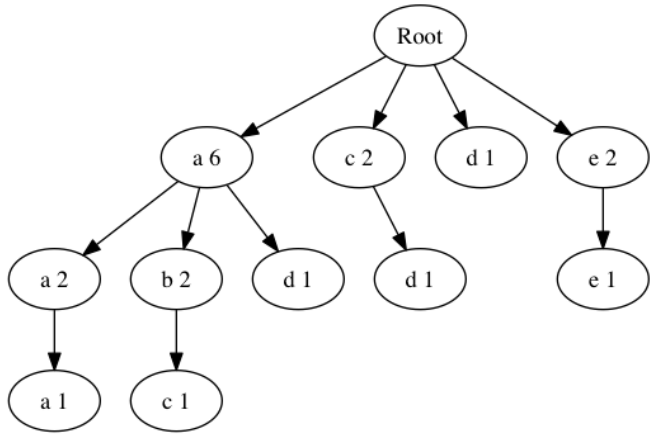

Figure 2.1: LZ78 model after parsing “aabcdeeeaaadabccdaaa”.

sequence. By using a sliding window, we ensure that we capture contexts that cross the phrase boundaries of the regular LZ78 algorithm, thereby constructing a more complete order-k-1 Markov model, where k is the longest LZ phrase seen so far. This window starts out small, and grows with the length of the sequence. This is to ensure that information can be gathered about the relative frequency counts at each order, while still attaining a high prediction accuracy.

Algorithm 1 ALZ algorithm

```
dictionary ← null
phrase w ← null
window ← null
Max_LZ_length ← 0
while v = next_symbol do
  if (w+v) in dictionary then
    w = w + v
  else
    add (w+v) to dictionary
    update Max_LZ_length if necessary
    w = null
  end if
  window = window + v
  if length(window) > Max_LZ_length then
    delete window[0]
  end if
  Update frequencies of all possible suffixes within window
end while
```

As seen in algorithm 1, the learning procedure keeps track of a number of previous symbols in a window, up to max_LZ_length, the length of the longest LZ phrase, and the current phrase. For each input symbol, the algorithm checks if the current phrase plus the new symbol is part of the dictionary. If it is, then it updates the phrase to be the old phrase plus the new symbol. If the phrase plus the new symbol does not exist in the dictionary however, then it is added to the dictionary, the max_LZ_length is updated if the phrase is longer than the current max_LZ_length, and finally, the phrase is reset to contain no symbols. After that has been done, the symbol is added to the window. Next, the algorithm checks if the window is greater than the max_LZ_length. If it is, then it removes the oldest symbol, before it lastly updates all the frequencies of the contexts contained in the new window.

Step	Window	Phrase	Contexts	Max LZ length
1	a	a	a	1
2	a	a	a	1
3	ab	ab	ab, b	2
4	bc	c	bc, c	2
5	cd	d	cd, d	2
6	de	e	de, e	2
7	ee	e	ee, e	2
8	ee	ee	ee, e	2
9	ea	a	ea, a	2
10	aa	aa	aa, a	2
11	aa	a	aa, a	2
12	ad	ad	ad, d	2
13	da	a	da, a	2
14	ab	ab	ab, b	2
15	abc	abc	abc, bc, c	3
16	bcc	c	bcc, cc, c	3
17	ccd	cd	ccd, cd, d	3
18	cda	a	cda, da, a	3
19	daa	aa	daa, aa, a	3
20	aaa	aaa	aaa, aa, a	3

Table 2.2: ALZ simulation of the sequence “aabcdeeeaaadabccdaaa”.

2.2.2 Example

Using the same example as for LZ78, we step through the ALZ algorithm in 2.2 and visualize the resulting tree in figure 2.2. If we compare the two trees in figures 2.1 and 2.2, we can see that figure 2.2 has a more complete and dense structure. This enables ALZ to converge to optimal predictability much faster than LZ78.

2.2.3 Complexity

In the worst case, the number of nodes in the ALZ tree is $\mathcal{O}(n^2/3)$. This situation occurs when each new sequence is a continuation of the previous sequence and one additional symbol. For example, ”a ab abc abcd abcde” (without spaces) would be such a sequence. In reality though, the number of nodes grow at a slower pace. This is because the worst case example above will be interspersed with shorter phrases, that merely fill out the tree. For a k-order tree, and a fixed alphabet size n, the algorithm is completing the n-ary tree of depth k, which

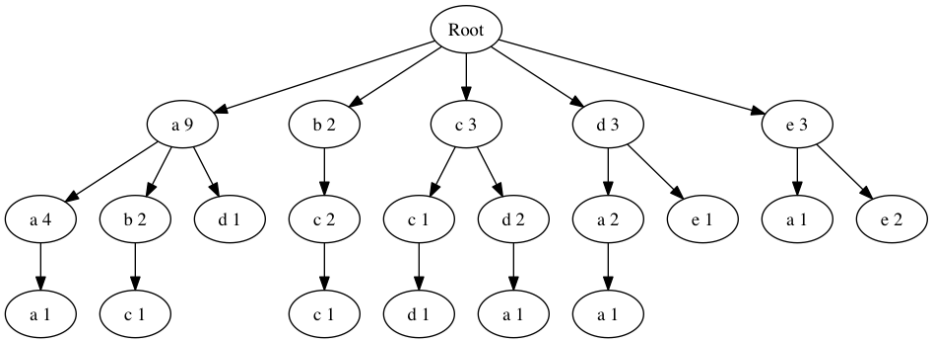

Figure 2.2: ALZ model after parsing "aabcdeeeaaadabccdaaa".

brings the space requirements close to a linear $\mathcal{O}(n)$. This can be seen in table 2.2 where we see the window size increase at a much slower rate than the worst case identified above.

The time complexity is the same as for the number of nodes, $\mathcal{O}(n^2/3)$. This is the sequence that will generate the most frequency updates and node creation out of all possible sequences. Just as for the number of nodes though, the authors observed that the time complexity is closer to $\mathcal{O}(n)$.

2.2.4 ALZ-TDAG

Transition Directed Acyclic Graph (TDAG) is another Markov-based approach to the problem of discrete sequence prediction, first described in [Laird and Saul, 1994]. This algorihm uses user-defined parameters to tweak the run-time and memory characteristics.

TDAG is suitable for online prediction, since we can simply tweak the user parameters to control the run-time of the prediction element. However, it is not sensitive to concept drift, meaning it is not able to register changes in existing patterns. Its predictions are generated from the highest-order reliable model and it does not weigh recent events more heavily like ALZ does. In a smart home setting, this means that the algorithm will perform worse as the inhabitant's patterns change over time. This is a highly unwanted characteristic for a persistent smart home system.

In [Gopalratnam and Cook, 2007], the authors found that the TDAG algorithm can be used for implementing ALZ, by tuning the user-defined parameters to create the same Markov model as ALZ.

The poor concept drift handling of TDAG is alleviated by adopting the tree

depth conditional of the ALZ algorithm. This way, we create the same Markov tree as ALZ, but we keep the run-time characteristics of the TDAG algorithm, which gives us the run-time complexity $\mathcal{O}(\sqrt{n})$, down from ALZs $\mathcal{O}(n^2/3)$. This improvement is possible because while the ALZ algorithm has to consider every node at every level in the Markov tree up to order k-1, the TDAG algorithm stores exactly the nodes that are needed for updating the model and predicting symbols.

Essentially, this means that we create the same result as in ALZ but we reduce the run-time complexity, without adding any any drawbacks elsewhere.

2.3 SPEED

Another algorithm that fits the criteria we identified in section 2.1 is the SPEED algorithm.

To predict well, we have to have a good grasp of the patterns that give rise to the data in the first place. The closer we can model the source, the better our accuracy will be. In a smart home, we will not only have information about the times when the devices are interacted with, but also the state of those devices at that time. So far, the algorithms we have looked at considers the on and off states of a device to be two entirely separate event, with no link between them. This can be rectified by taking advantage of this link between states in our algorithm, specializing the algorithm to this particular kind of data.

[Alam et al., 2012] introduces the sequence prediction via enhanced episode discovery (SPEED) algorithm. As the name suggests, SPEED uses the device state information to find episodes, which are sequences of events in between the on and off state of a device.

The basis for this model is that a user typically has some routines that can be captured by looking at the state of the devices they interact with. For example, when a user enters the kitchen, the first thing they do will probably be to turn on the kitchen light. Then, any activity performed in the interval of the kitchen light turning on and off is part of an episode.

The model created by SPEED consists of these episodes. Say we have a sequence of events “AbcDEda”, where capitalized letters mean on and lowercase letters mean off. Then SPEED will detect two episodes from this sequence, “AbcDEda” and “DEd”, which have boundaries of opposite events.

2.3.1 Learning procedure

SPEED creates a decision tree to keep track of the episodes it finds. The learning procedure keeps track of a window containing symbols and the maximum length of the episodes seen so far called max_window_length. At every iteration, the

algorithm takes in a new symbol from the input stream, adds it to the window, and checks to see if the opposite version of that symbol exists in the window. For example, if the algorithm takes in a "d" and the window consists of "cDe" so far, then the resulting window will be "cDed", which gives an episode consisting of "Ded". The algorithm then updates max_window_length if the current episode is longer than all the previous ones. This is used to prune the window after finding an episode. Next, the algorithm retrieves all the suffixes from the episode, which is how ALZ also builds its model. These suffixes are used to create new nodes and update the frequencies of all possible contexts found in the episode. If the input symbol does not match an opposite symbol in the window, then the algorithm returns and waits until it gets a new symbol.

The algorithm builds a k-1th order Markov model, where the k is the longest possible episode length. The tree that the algorithm creates has a maximum depth of k levels, and contains the frequencies of all the contexts gathered from the episodes. From these frequencies, we can build a probability distribution of all the symbols, which we can use to predict the most likely devices that will be interacted with by the user.

2.3.2 Example

Let us consider an example sequence to simulate the SPEED algorithm:

AbaCdBADceECdb

To signify the on and off states of the devices, we use uppercase symbols to signify on, and lowercase symbols to signify off. For example, "A" means on for a particular device, while "a" means off for that same device.

SPEED constructs a tree with long Markov chains and high density, which according to [Alam et al., 2012] provides a higher prediction accuracy than a tree with less density and shorter chains. The model in figure 2.3 has long Markov chains but has not yet started to grow dense, which will happen when more episodes have been processed. We can immediately see the difference from the ALZ tree in figure 2.2, which produces a shorter tree with fewer branches.

2.3.3 Complexity

The worst case for the number of nodes in the tree is $\mathcal{O}(n^2)$. This will occur when the entire sequence is one episode, with no sub-episodes in between. For example, take the sequence "adiehkbcfA". In this sequence, there is only one episode that spans the entire sequence. The tree created from this episode has $n(n+1)/2$ nodes, which yields the worst case for the number of nodes in the tree.

Time complexity is similar, since the worst case here also occurs when there is a long sequence that forms an episode with no sub-episodes in between. If the

Step	Window before	Episode	Contexts	Max episode length	Window after
1	A	-	-	1	-
2	Ab	-	-	1	-
3	Aba	Aba	Aba, ba, a	3	Aba
4	AbaC	-	-	3	-
5	AbaCd	-	-	3	-
6	AbaCdB	baCdB	baCdB, aCdB, CdB, dB, B	5	baCdB
7	baCdBA	aCdBA	aCdBA, CdBA, dBA, BA, A	5	aCdBA
8	aCdBAD	dBAD	dBAD, BAD, AD, D	5	CdBAD
9	CdBADc	CdBADc	CdBADc, dBADc, BADc, ADc, Dc, c	6	CdBADc
10	CdBADce	-	-	6	-
11	CdBADceE	eE	eE, E	6	BADceE
12	BADceEC	ceEC	ceEC, eEC, EC, C	6	ADceEC
13	ADceECd	DceECd	DceECd, ceECd, eECd, ECd, Cd, d	6	DceECd
14	ADceECdb	-	-	6	-

Table 2.3: SPEED simulation of the sequence “AbaCdBADceECdb”.

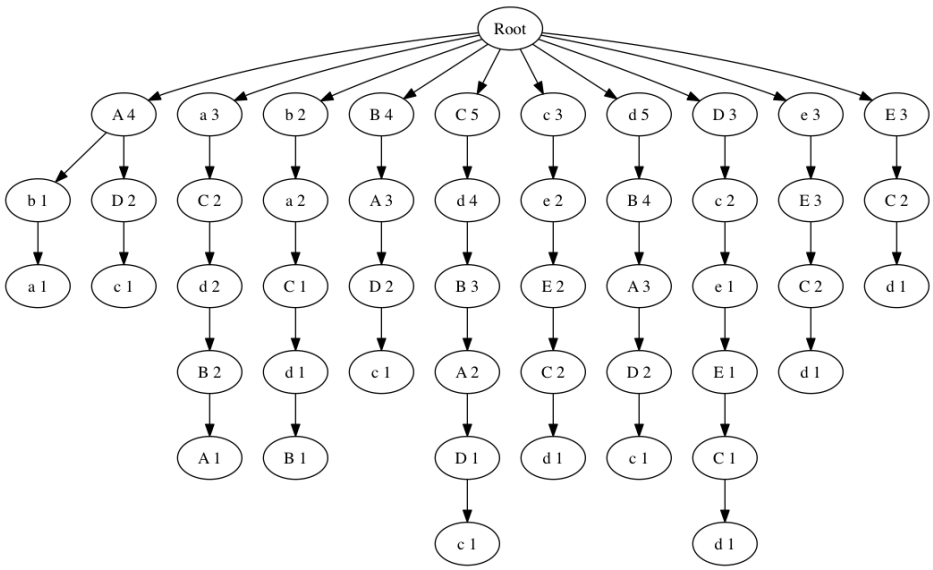

Figure 2.3: Model after parsing “AbaCdBADceECdb”.

sequence consists of an episode with n symbols, then the algorithm has to search all the branches, and update the frequency of every node of each branch. The time required is then n(n+1)/2. Which is the same time complexity as for the space complexity, $\mathcal{O}(n^2)$.

2.4 Prediction by Partial Match

Both ALZ and SPEED construct a Markov tree model, and they both utilize the same Prediction by Partial Match (PPM) strategy to predict the next event in the sequence.

We want to take into account as much information as possible when making predictions. Therefore we want to predict using the longest possible Markov-chain that matches our current context. But the longest chain is also necessarily the one with the fewest branches leading out from it, making it hard to give a good prediction from solely that context.

Say we have a 9-symbol context. Depending on the total number of symbols, it will take a huge amount of input to adequately flesh out the possible 10-symbol contexts branching out from our 9-symbol context. This means that any predictions made from solely the 9-symbol context will have a poor accuracy. We need a way to ensure that we consider symbols that does not yet branch from our current 9-symbol context. This is where PPM [Bell et al., 1990] comes in.

PPM is a general blending scheme. It uses different-order Markov models to build a probability distribution. The main difference between the PPM implementations has to do with how they determine the escape probability, which is the probability that a previously unseen symbol at this context will be the next one in the sequence. ALZ and SPEED both use the PPM strategy of exclusion, which escapes to the k-1 order context based on the ratio of null outcomes to all possible outcomes. In general, the PPM equation is as follows:

$$P(\varphi) = p_k(\varphi) + e_k(\varphi)P_{k-1}(\varphi) \tag{2.1}$$

$$p(\varphi) = \frac{c_k}{C} \tag{2.2}$$

$$e_k = \frac{c_0}{C} \tag{2.3}$$

Where $P_k(\varphi)$ is the final probability, $p_k(\varphi)$ is the probability of the symbol φ in the current context, $e_k(\varphi)$ is the escape probability, and $P_{k-1}(\varphi)$ the probability of the symbol at the k-1 phase. C refers to the total number of sequences of length k-1. c_k is the count of event φ at sequence length k. and c_0 is the number

of null outcomes, which are all the outcomes that do not end in a symbol for the given context.

Let us see an example using the graph we created using ALZ in figure 2.2. We assume the current window is "abc". The contexts that can be used for prediction then are "ab", "a", and the null context. If we want to find the likelihood that the next symbol in the sequence is "c", then we can calculate it using equation 2.1, and the frequencies from the graph in figure 2.2.

$$\frac{1}{2} + \frac{1}{2} \times \left(\frac{0}{9} + \frac{2}{9} \times \left(\frac{3}{20}\right)\right) = 0.516. \tag{2.4}$$

We can also predict the chance of seeing "a" from the same window:

$$\frac{0}{2} + \frac{1}{2} \times \left(\frac{4}{9} + \frac{2}{9} \times \left(\frac{9}{20}\right)\right) = 0.272. \tag{2.5}$$

We can predict all the possible symbols from the current window context using equation 2.1. By using the exclusion strategy, we ensure that the lower order nodes also get taken into account, but with reduced influence equal to the fraction of null outcomes. This, coupled with a gradual increase in the Markov order of the ALZ and SPEED models, means that both algorithms are able to respond to changes in user's patterns over time.

The difference between ALZ and SPEED with regards to PPM prediction is that the TDAG base for ALZ already stores pointers to the nodes that are used for prediction, while SPEED has to search through the tree to find the relevant nodes. This takes longer, and means that ALZ is faster at predicting the next event in any given context.

2.5 Comparison

[Alam et al., 2012] has compared the prediction accuracy, space complexity and time complexity of the ALZ algorithm and the SPEED algorithm on the MavLab data set[1]. This data set was gathered over a month in the MavLab at University of Texas, Arlington. It consists of 750 events, based on the device interactions of six participants during the month of April 2003.

[Alam et al., 2012] finds that the SPEED algorithm achieves prediction accuracy of 88.3 percent, up from the prediction accuracy of ALZ which is 47 percent [Gopalratnam et al., 2003] on the same data set. For the five most likely devices, the accuracy of ALZ rises to 89 percent, about the same as the SPEED accuracy for a single item. We do not have the accuracy of SPEED on the five most likely devices, but it is safe to assume that it is above 90 percent.

[1] http://ailab.wsu.edu/mavhome/datasets/mavlab.zip

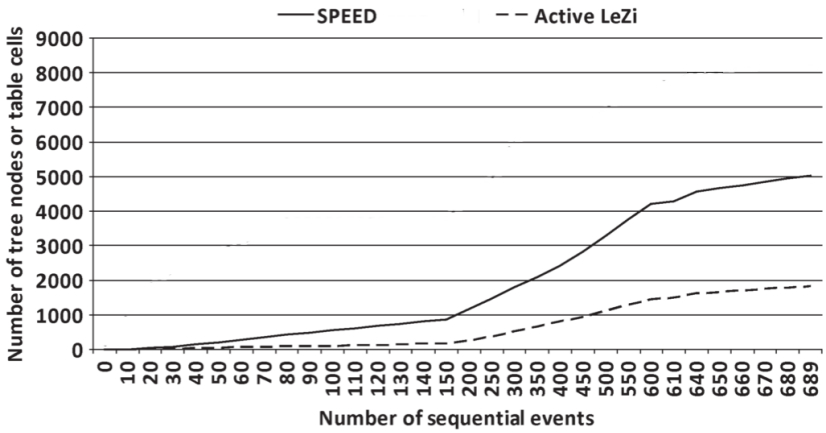

Figure 2.4: Space comparison of ALZ and SPEED (adapted from [Alam et al., 2012]).

If all we are concerned with is the accuracy of our predictions, we can conclude that SPEED is a better algorithm. However, as mentioned in the introduction of chapter 2, we place a high premium on computational power for mobile devices. We can see in figure 2.4 that the space requirements for SPEED is approximately 2.5 times greater than for ALZ when it has been trained on the MavLab data set. As we saw in section 2.3.3 and 2.2.3, the run-time complexity of both algorithms scale in the same manner as their respective space complexity, amounting to the worst case $\mathcal{O}(n^2)$ for SPEED, and $\mathcal{O}(n^2/3)$ for ALZ.

However, as explored in section 2.2.4, by using the TDAG base algorithm for ALZ, we keep references to the exact nodes that are needed to build the model and predict the next symbol in the sequence. This allows the algorithm to predict a symbol in $\mathcal{O}(\sqrt{n})$ time.

SPEED bests ALZ in prediction accuracy, but if we compare the computational footprints of our two algorithms, ALZ scales better than SPEED. The SPEED algorithm has a maximum of 11 percentage points better prediction accuracy on the five most likely devices. This small gain in prediction accuracy does not justify the greater time and space complexity of SPEED over ALZ for our usage. We have to value battery life more than the small increase in prediction accuracy. We will see in section 5.2 that even with 100 percent prediction accuracy, we only get a slight improvement over ALZ with regards to minimizing

the interaction time.

2.6 Related Work

In this section we will look at similar solutions for predictive remote control system. We will look closer at one paper in particular, that explores several algorithms and compares their performance.

2.6.1 Automated Selection of Remote Control Interfaces

[Desai et al., 2002] addresses the "active device resolution" problem. From any number of devices, which one does the user want to interact with next? The authors aims to solve this problem by taking into account the previous remote control access history of the user and the current user context, which is exactly what we are trying to do with our system. Four different prediction algorithms are compared with respect to their accuracy and their run-time performance.

The authors tests Markov processes of the 1st to the 3rd order. Using only the previous event, the algorithm achieves a result of 68 percent. The 2nd order gives an accuracy of 76 percent. The 3rd order decreases to 65 percent however, which according to the authors might be due to the noise in the third previous action of the user, which makes sense given that there are only 4 devices.

Naive Bayes performs with similar accuracy to the Markov processes when using the previous two events. It handles noise in the data better as well, since using the three previous events outperforms the 3rd order Markov process. The run-time of Naive Bayes is fast as well, making it a good choice for a real-time system algorithm.

The nearest neighbour algorithm produces very good results on the test set, achieving an accuracy of 94 percent. Unfortunately, it is also the slowest of the algorithms tested. It is too slow to perform in real-time, which makes it unsuitable for our purposes.

The decision tree accuracy is approximately 20 percent worse than the nearest neighbour algorithm. Although the run-time is better than nearest neighbour, it is still too slow for any real-time use according to the paper.

This paper highlights the importance of choosing the right algorithm. We want high prediction accuracy, since that will decrease the time spent searching for the right device. But we also have to consider the landscape where our solution is embedded. A nearest neighbour algorithm might have the highest prediction accuracy, but if it takes too long to build its model, it will slow down the user interface or use too much power. The difference to our approach is that the authors do not address the user interface, and are not concerned with optimizing for the time of the user.

2.6.2 MavHome

MavHome [Cook et al., 2003] is a smart home project from the University of Texas, Arlington. The goal of the MavHome project is to create a home that acts as a rational agent, maximizing inhabitants comfort and minimizing the operation cost. It does this by predicting and reasoning about its inhabitants. The prediction layer utilizes three algorithms, ALZ among others, to create a meta-predictor that predicts the next action of the inhabitant. This is then used to automate repetitive and routine tasks.

This approach is a little more involved than our own. Where MavHome seeks to create a rational agent that can automate and reason about the inhabitants actions, our system is focused on providing the user with the most direct access to the devices in his home, disregarding automation capabilities and reasoning beyond predicting the next device that the user wants to interact with.

Chapter 3

User Interface

We want to know how our user interface design stacks up against one of the current smart phone remote control systems on the market, therefore we need a technique that allows us to compare two user interfaces with respect to how much time it takes to complete specific tasks with each of them. The best way to do this would be an empirical user test. Unfortunately, this is an expensive technique, and slightly unnecessary since we have an alternative that is good enough for our purposes.

3.1 NGOMSL

In 1983, the Goals, Operators, Methods, Selection rules. (GOMS) model [Card et al., 1983] was proposed. The GOMS model aims to remedy the cost of running an empirical user test, by constructing an engineering model for usability. This allows us to design and test a user interface model without involving several users over multiple design iterations, which takes up a lot of time and resources.

The GOMS model reduces a users actions with a system down to its basic elements. These basic elements – physical, perceptual and cognitive – are then used as the framework in which to study the system.

A GOMS model consists of Goals, which is what the user wants to accomplish. Operators, which are the actions the user performs to reach the goal. Methods, which are the procedures containing operators, and lastly, Selection rules, which can be used to distinguish which method to use when there are several different methods for completing the same goal. The model we create is not only descriptive, but also executable. This means that we can model our system and compare the execution times with another model without running an expensive user test.

Several different GOMS models exist within the GOMS family. The simplest

one is KLM, which describes a user's interaction with the user interface at the "keystroke" level. On the other side of the complexity scale is CPM-GOMS, which is the only GOMS technique that can model multitasking behaviour in experienced users.

A third method that lies in between these two is the NGOMSL [Kieras, 1996] method. NGOMSL is a structured natural language for explicitly representing the methods and selection rules of the user. It adds a natural language syntax on top of GOMS to make it simpler to use. This is the method we will use to model and compare the user interfaces in chapter 4 and 5.

3.1.1 Example

To perform a NGOMSL analysis, we need to identify and describe the model's various goals, operators, methods and selection rules. A critical process involves deciding what to describe and what not to describe. If we go too far down, we will be stuck creating methods that might be too granular for our purpose. For example, a mental process like reading would be very hard to describe properly. Therefore, we can replace it with a simpler operator that approximates the process, instead of trying to create a realistic model with basis in ocular mechanics. As long as we are only trying to model a user interface, the effort is better spent elsewhere.

Let us consider an example NGOMSL model made using the guide [Kieras, 1996]. The user's goals are to delete one file, and duplicate one file in the OS X operating system. First we decompose the goals into methods:

```
Method for goal: delete file.
1. Accomplish goal: delete file using context menu.
2. Return with goal accomplished.

Method for goal: duplicate file.
1. Accomplish goal: duplicate file using context menu.
2. Return with goal accomplished.
```

Both of these methods use the context menu to accomplish their goals. These methods only differ in the icon that is selected from the context menu, which makes it possible to replace both the sub-methods with a generalized method:

```
Method for goal: delete file using context menu.
1. Accomplish goal: perform action on item using context menu.
2. Return with goal accomplished.

Method for goal: duplicate file using context menu.
1. Accomplish goal: perform action on item using context menu.
2. Return with goal accomplished.
```

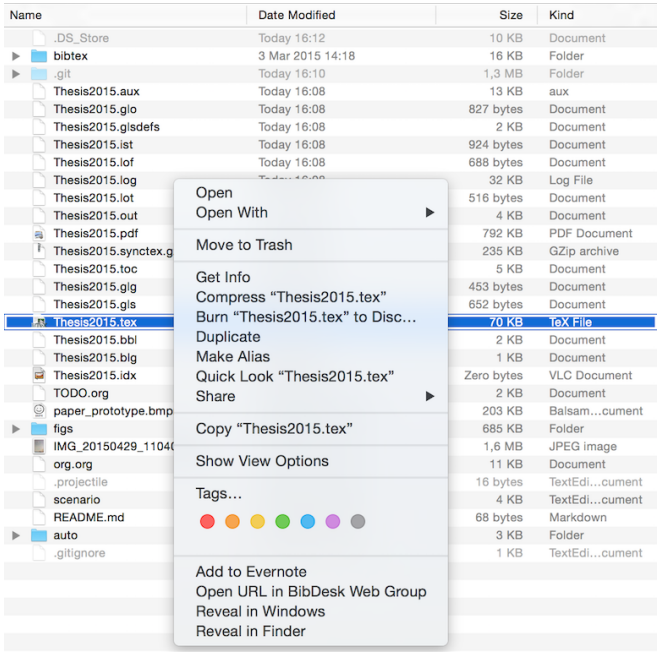

Figure 3.1: A context menu in OS X.

```
Method for goal: perform action on item using context menu.
1. Locate item on screen.
2. Move cursor to item location.
3. Right click item.
4. Locate action item in drop down menu.
5. Move cursor to action item location.
6. Click mouse button.
7. Verify that action was performed on item.
8. Return with goal accomplished.
```

Some times there are several ways to accomplish a given goal. Selection rules can then be used to describe a choice between two or more alternatives. In the example above, we delete the item using the context menu. However, this is also an action that can be done by dragging the item to the trash icon. As an example, let us say that the user is more likely to drag the item to the trash if the trash icon is visible on the screen. Then we can create a selection rule out of the delete file method.

```
Selection rule set for goal: delete file.
If trash icon is visible on screen Then
    accomplish goal: drag item to trash.
If trash icon is not visible on screen Then
    accomplish goal: delete file using context menu.
```

If we want to estimate the time it takes to complete a task using this model, we can simply walk through it. Typically there is a main method that is responsible for retrieving tasks, and a selection rule set that is responsible for delegating the tasks to the proper methods.

```
Method for goal: OS X file operations
1. Get next unit task from task list.
2. Decide: If no more tasks, then return with goal accomplished.
3. Accomplish goal: perform unit task.
4. Goto 1.

Selection rule set for goal: perform unit task.
If the task is deleting an file Then
    accomplish goal: delete file.
If the task is duplicating a file Then
    accomplish goal: duplicate file.
```

Now we have all the methods we need for finding the time it takes to delete a file in our model.

```
Task: Delete item

Assumptions:
The finder application is open on screen.
The Trash icon is not visible on screen.

Trace:
1. Accomplish goal: delete file.
  2. Accomplish goal: delete file using context menu.
    3. Locate item on screen. (M)
    4. Move cursor to item location. (P)
    5. Right click item. (BB)
    6. Locate action item in drop down menu. (M)
    7. Move cursor to action item location. (P)
    8. Click mouse button. (BB)
    9. Verify that action was performed on item. (M)

2 M + 2 P + 4 B + 9 statements
= 2 * 1200 ms + 2 * 1100 ms + 4 * 100 ms + 9 * 100 ms
= 5900 ms
```

Six seconds is an abnormally long time for deleting a file. When this same procedure was tested on a real computer, it took approximately four seconds to complete. The discrepancy mostly comes from the time it takes to find and select the correct item in the context menu. The 1100 ms P operator is from KLM and is estimated based on large moves across the screen to small targets.

When we open a context menu, there is only a small area close to the cursor that is of interest to us, which makes the distance we have to move the cursor much shorter. In addition, for an experienced user, some operations do not really take any time at all, as the user overlaps these operators with other activities. For example, an experienced user knows that the context menu pops up on the right-hand side of the cursor, and that the "Move to Trash" item is located as the second item in the list. Therefore, they do not need to spend time mentally processing where to move the cursor next. By the time the user has opened the context menu, they are already moving the cursor to the correct item.

If we replace the corresponding actions at step 6 and 7 with a simple move of 0.5s, then the resulting time becomes 4000 ms, which is the time it took when testing the same procedure on the author's Macbook Pro.

As we can see from the example above, modeling with NGOMSL can be accurate as long as we take into account how experienced our user is, and choose the best operators for our system. In many cases, a P action will take significantly lower time than the KLM estimate of 1100 ms. If our buttons are closely

Short untargetted swipe	70 ms
Icon pointing from the home screen	80 ms
Half-screen sized zoom	200 ms

Table 3.1: Touch-screen specific KLM operators.

grouped, it may take anything from 200 - 500 ms instead. For a touch screen, the discrepancy between the KLM estimated execution times and actual execution times is even more pronounced.

3.1.2 Touch Interface Operators

GOMS was originally created as a model for desktop computers with a keyboard and a mouse. The challenge when moving to touch-based interfaces is that some of the execution times estimated for a desktop computing model do not translate well to a touch-display.

The discrepancy is more pronounced when pointing to an object on screen. For example, modeling an untargetted swipe using the regular KLM operators designed for mouse and keyboard would require 3 operators: Touch down, Point to a target on screen, and Touch up. Together, this gives an estimated time of 0.1s + 1.1s + 0.1s = 1.3 seconds. This is inaccurate by at least one order of magnitude. Fortunately, [Batran and Dunlop, 2014] explores the KLM model on mobile touchscreen devices, and has come up with 3 new movement operators, seen in table 3.1, to remedy some of the flaws when translating from a mouse-based interface to a touch-based interface.

When using a mouse, we place a level of indirection between us and the interface, which makes it slower than if we were to point directly to the object on the screen using our finger. In addition, a smart phone screen is much smaller than a computer screen, with larger buttons relative to the screen. Therefore the time it takes to perform action related to pointing and swiping are significantly reduced on a touch-based user interface.

3.1.3 Related Work

The GOMS method has been in a wide variety of fields. For instance, it was used to create a training program for a cockpit Control and Display Unit (CDU) in [Polson et al., 1994]. By using cognitive complexity theory and GOMS, the authors identified and modelled specific tasks as a collection of highly detailed rules, containing both the mental operations performed by the user, as well as the key presses needed to complete specific tasks. The resulting model was then able to make accurate predictions about training time, and the time it took to transfer

knowledge from an old cockpit system, to the new CDU system. Additionally, it was also used as the basis for a training program, allowing a pilot to practice various tasks in isolation from the rest of the CDU system.

A different field that the GOMS method has been used for is human-robot interaction [Drury et al., 2005]. The authors created GOMS models for two different interfaces for search-and-rescue robots and compared their performance.

The GOMS method was also used in [Gray et al., 1993] to compare a new work station to an old one. The GOMS model predicted that the new workstation decreased productivity by 3 percent. After gathering empirical data over the period of four months, the conclusion was that the new workstations decreased productivity by 4 percent. The GOMS model not only accurately predicted the decrease in productivity, but also identified the cause, which was that the new workstations did not utilize the workers downtime.

3.2 Android User Interface

To utilize the advantages of that we get from prediction, we need to familiarize ourselves with the possibilities afforded by Android's user interface.[1] We will only consider the stock Android user interface, since that is what most Android phones use.

3.2.1 Apps

In chapter 1 we broached the problem associated with the smart phone app paradigm. Android smart phones build their extension capabilities around the concept of apps. For most tasks you perform on your smart phone, this works just fine. But for short tasks, such as turning on and off a light, you want it to be as fast as possible. An example of an app can be seen in figure 5.1.

A phone can be in three different states when it unlocks. It can open straight to the desired app, a different app, or the home screen. However, we can guarantee that the smart phone opens to a certain app if that app has a notification on the lock screen providing direct access to that app. This is a useful feature if we want to minimize the time it takes to interact with our phone.

3.2.2 Widgets

Widgets are small embeddable application views.[2] They circumvent some of the restrictions placed on apps since they do not need a separate screen, but can instead be embedded in other applications, such as the home screen. This is

[1] https://developer.android.com/guide/topics/ui/index.html
[2] https://developer.android.com/guide/topics/appwidgets/index.html

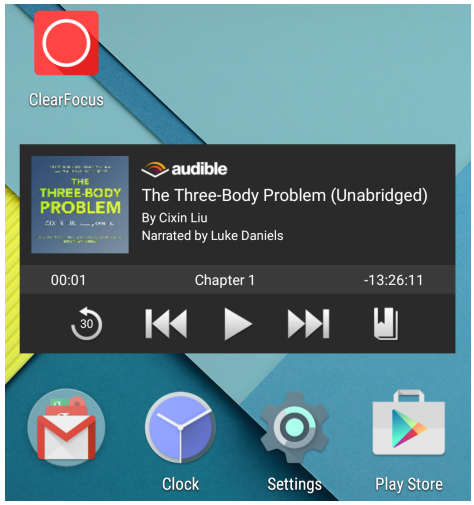

Figure 3.2: A widget situated on the home screen.

much faster than opening an app to get to some specific content. Additionally, the size of a widget can be adjusted to take up more or less space on the home screen.

However, it still has the drawback that the phone does not necessarily open from the lock screen to where the widget is located. If the phone was last locked with an app open, and the widget is situated on the home screen, then we first need to unlock the phone and exit the app before we can interact with the widget.

3.2.3 Notification System

Androids notification system allows apps and background services to push notifications to the notification bar and the lock screen.[3] This means that the first thing the user sees when the screen is turned on is a list of all their notifications.

The notification system is not limited to only displaying messages. Some functionality can be exposed through buttons on the notification itself, as shown in figure 3.3.

In the Android 4.1 update, a new notification system was released, granting notifications the ability to implement an expanded view in addition to the normal notification view. This expanded view can be toggled by swiping down a notification, seen in figure 3.3, which then transforms into an expanded notification,

[3]https://developer.android.com/guide/topics/ui/notifiers/notifications.html

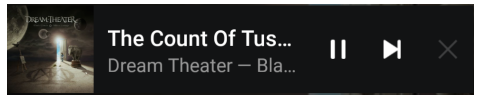

Figure 3.3: Normal view notification

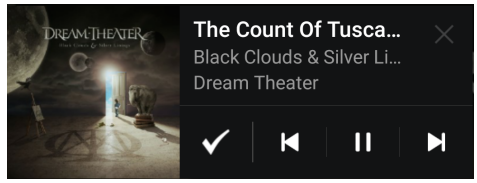

Figure 3.4: Big view notification

seen in figure 3.4. This allows a notification to display additional information, and expose options that do not fit in a regular notification.

The Android API specifies that a maximum of three actions can be exposed through an expanded notification.[4] However, this is a soft limit rather than a built-in restriction, as we can see in figure 3.4, where there are four different actions available. What has a hard limit however, is the size of the expanded notification. The vertical height of the expanded notification can not exceed four times the height of a regular notification, which, if we additionally want to display some text for each action, limits us to five actions stacked vertically.

[4]https://developer.android.com/design/patterns/notifications.html#ExpandedLayouts

Chapter 4

UI Model

In section 3.2, we explored the different user interface aspects of the Android platform, the most interesting of which is the notification system. The main limitation of notifications is the fact that they are small. However, the notification system alleviates this problem to some degree by allowing us to implement an expanded notification right on the lock screen. We will not be able to display all the items that a regular app screen would be able to with the expanded notification, but if we use a prediction algorithm, we can display the devices that the user is most likely to interact with.

In this chapter we present the prototype of our system, which we have nicknamed Prediction app. We have created a mock-up and an interaction model of our system using NGOMSL. This allows us to compare the quality of our system, with regards to the time metric, to a commercial smart remote app.

4.1 Mock-up

A mock-up of the normal sized notification can be seen in figure 4.1. When it is expanded, it transforms into the expanded notification shown in figure 4.2, which reveals the most likely devices according to our prediction algorithm. This list updates continuously as the algorithm receives events from the devices in the smart home. When the user then turns off the bedroom light seen in figure 4.2, the list will update to display the devices that are most likely given the new context. For instance, if turning off all the lights in the user's home is a regular part of their routine before going to bed, the notification list may start filling up with other lights in the house.

In case that the device we want is not in the expanded notification list, we can open the normal app view from the lock screen by tapping the notification

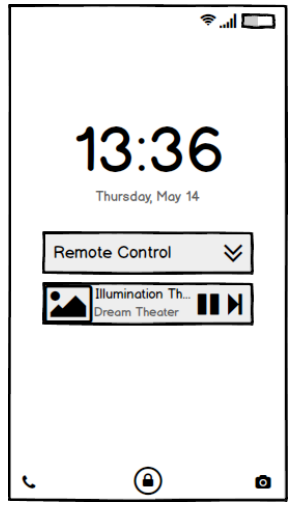

Figure 4.1: Prediction app notification on lockscreen.

header, which is the "Remote Control" label seen in figure 4.1 and 4.2. This opens the app screen seen in figure 4.3. This screen is static and can be arranged to the user's preference.

4.2 Prediction App NGOMSL Model

We created the prediction app model using NGOMSL [Kieras, 1996]. In this section we will walk through our prediction app model, highlighting some methods of interest. We also created a model of a commercial smart phone app that we will use as a baseline for the comparison in chapter 5. This model can be found in appendix 6.3.

```
Method for goal: smart remote prediction app
1. Get next unit task from task list.
2. Decide: If no more tasks then return with goal accomplished.
3. Accomplish goal: perform unit task.
4. Goto 1.

Selection rule set for goal: perform unit task
if the task is expanding the notification then
   accomplish task: expand notification
if the task is finding the device then
```

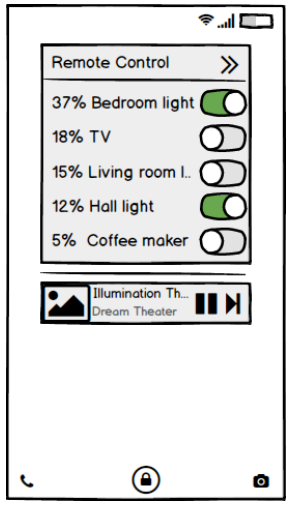

Figure 4.2: Expanded prediction app notification.

Figure 4.3: Prediction app screen.

```
    accomplish goal: find device.
if the task is toggling the device then
    accomplish goal: toggle device.
```

The two methods above are the entry-point to our model. The selection rule set exposes the high-level methods in our model. The task list refers to the list of tasks needed to accomplish the user's goals.

```
Method for goal: expand notification
1. Point to notification.
2. Swipe down.
3. Return with goal accomplished.
```

The expand notification method assumes that the phone is on the lock screen, and that the notification is visible. It transforms the notification into the expanded version, where we can control the most likely devices, as seen in figure 4.2.

```
Selection rule for goal: find device
if we are in the expanded notification then
    accomplish goal: find device in notification.
if we are in the app then
    accomplish goal: find device in app.
```

This is a selection rule that executes different methods depending on the state of the smart phone. It assumes that we are either on the lockscreen with the notification expanded, or in the prediction app.

```
Method for goal: find device in notification
1. Locate device.
2. Decide: If device-found then return with goal accomplished.
3. Accomplish goal: go to app from notification list.
4. Accomplish goal: find device in app.
```

This method either finds the correct device in the notification list, or it opens the app and finds it there. This is where the execution path diverges based on whether the prediction algorithm has predicted the device we are looking for.

```
Method for goal: go to app from notification list
1. Tap notification
2. Wait for system to complete transition.
2. Return with goal accomplished.
```

```
Method for goal: find device in app
1. Locate device.
2. Decide: If device-found then return with goal accomplished.
3. Accomplish goal: scroll list.
4. Goto 1.
```

If the device is not found in the expanded notification, then we have a secondary app screen that functions in much the same way as a regular app, seen in figure 4.3. We do not make use of prediction to rearrange devices on this screen. The reason is because having the user arrange this screen to their personal preference is a more potent time saver than rearranging the tiles on every state change, especially since when the user first opens the app, that means they did not find what they were looking for in the notification list.

```
Method for goal: toggle device
1. Point to device.
2. Tap device.
3. Verify that device was toggled.
4. Return with goal accomplished.
```

When we find our device, we toggle it by tapping the icon representing the device on our smart phone. Making sure that the UI responds before we proceed.

```
Method for goal: scroll list
1. Swipe up.
2. Return with goal accomplished.
```

It is difficult to accurately model scrolling and searching properly, as we both scroll and search at the same time. Therefore we have simplified it to consist of a separate search and scroll method.

Chapter 5

UI Comparison and Results

In chapter 1, we identified the time of the user as the metric we want to optimize for. The goal of our system is to present the devices that the user is most likely to interact with in a way that minimizes this metric. In this chapter, we compare the performance of our Prediction app model, with a commercial smart home remote control app, by using the GOMS models in appendix 6.4 and 6.3

5.1 Comparison Plan

To compare our two models, we choose a task that represents regular use of a smart remote, which we then test on the different models. We trace the task through our two models and compare the resulting times. We can identify the best and worst case times by running the models with different assumptions. The task we will use to compare the two models is the time it takes to toggle the state of a smart home device from the lock screen. We choose this particular task because it embodies the use case we want to optimize for, namely the time it takes to interact with a device when the phone is locked.

5.1.1 Mobile Execution Times

We utilize the touch screen gestures presented in [Batran and Dunlop, 2014] to get more accurate execution times for our models. The remaining execution times are taken from the KLM task execution times in [Card et al., 1983]. The total list of actions that influence time can be seen in table 5.1. We have also determined some execution time averages for the Nexus 5 smart phone in table 5.2 that we use as the transition times between different screens in the model. Note that in

Action	Description	Time
S	Swipe (left/right/up/down)	70 ms
P	Point to icon	80 ms
B	Tapping screen	100 ms
M	Mental act of routine thinking or perception	1200 ms
W(t)	Waiting for system to respond	t ms

Table 5.1: Execution times.

Average app transition time	800 ms
Unlocking phone with 4 digit pin	2300 ms
Unlocking phone without pin	800 ms

Table 5.2: Android specific execution times.

the comparison we assume that the user does not have to enter his pin code to unlock the phone, because this has no significant effect on the model comparison.

5.1.2 Baseline App

As our baseline comparison app we have chosen the Revolv app, seen in figure 5.1. The reason we chose this app is because it has a clean user interface, with all devices clearly visible on the main page of the app. The NGOMSL model for this app can be found in appendix 6.3.

5.1.3 Model Assumptions

For our prediction app model, the assumption that most heavily influences the time is whether or not the intended device is among the ones predicted by the algorithm. If it is, then it will show up on the expanded notification list. If not, the user will have to open the app and find the device manually. Since the app can be opened directly from the lock screen, the app that is currently active on the phone will not influence the time it takes to open the app.

For the Revolv app model, the assumption that most influences the time is which app is active on the phone. There are three different cases when the user unlocks the screen. The first case occurs when the app is already active, so when the user unlocks the phone, the user does not have to waste any time opening the app. The second case is when no app is active, and the phone unlocks straight to the home screen. From here the user must find the app and launch it. The third case is when another app is active. This means the user has to go to the home screen first and then launch the app.

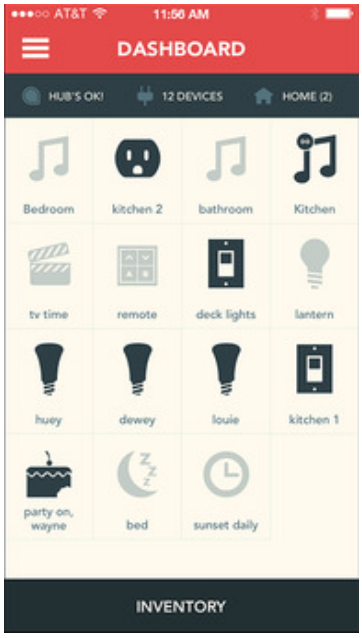

Figure 5.1: Revolv app.

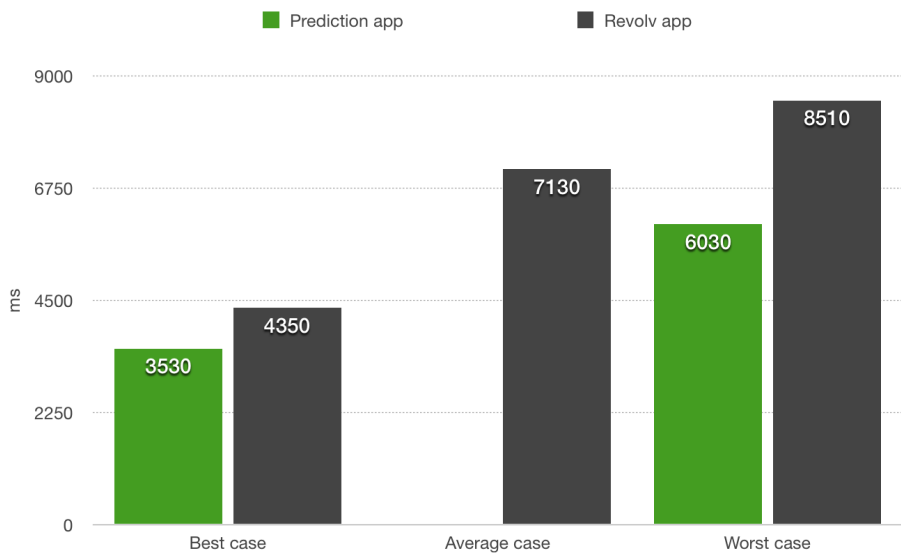

Figure 5.2: Time comparison.

Case	Time
Best case	3530 ms
Worst case	6030 ms

Table 5.3: Prediction app model task trace.

5.2 Comparison

For the Prediction app model, if the algorithm predicts the correct device as one of the top five items, then we get the best case, which means it will take 3530 ms to complete the task, as seen in figure 5.2. If it does not predict the correct device, then we get the worst case, which means it will take 6030 ms instead. If we utilize the results obtained from the algorithm accuracy tests in [Alam et al., 2012], then the best case occurs 89 percent of the time for the ALZ algorithm. The traces of the Prediction app model can be found in appendix 6.6.

We can not say whether the best case is more likely than the average case or the worst case for the Revolv app model. This is because the assumptions that yield the different cases depend on how the user interacts with their phone.

Case	Time
Best case	4350 ms
Average case	7130 ms
Worst case	8510 ms

Table 5.4: Revolv model task trace.

If they use the Revolv app frequently relative to other apps, then the best case happens fairly often. However, most users use their smart phone for chatting and browsing more often than they control the devices in their home. Since how a person uses their smart phone varies greatly, there is little to gain by finding out the percentage split between the different cases. The traces of the Revolv app model can be found in 6.5.

In table 5.4 and 5.3 we see that in general, the Prediction app model is faster. Most of the difference between the Prediction app model and the Revolv app model stems from the way that Android apps work in general. There is a significant overhead to opening apps by going through the home screen, if all you want to do is to toggle a single button in the app. The longer the task is, the less of a problem this becomes.

```
Prediction app model, worst case.
Task: Toggle device.

Expanded for Prediction app:
1. Expand notification.
2. Find device.
3. Toggle device.

Assumptions:
The phone is in the users right hand.
The app is currently on the lock screen.
The phone has no currently active app.
The phone has no lock-screen security.
The device is not on the notification list.
The device is on the first screen of the app list.

Prediction app trace:
1. Accomplish goal: expand notification.
  2. Point to notification. (P)
  3. Swipe down. (S)
4. Accomplish goal: find device in notification.
  5. Locate device. (M)
```

```
  6. Accomplish goal: Go to app from notification list.
    7. Tap notification. (B)
    8. Wait for system to complete transition. (W(0.7s))
  8. Accomplish goal: find device in app.
    9. Locate device. (M)
10. Accomplish goal: toggle device.
  11. Point to device (P)
  12. Tap Device (B)
  13. Verify that device was toggled. (M)

2P + 1S + 3M + 2B + 1W(0.7s) + 13 statements
= 2 * 80 ms + 70 ms + 3 * 1200 ms + 2 * 100 ms + 700 ms + 13 * 100 ms
= 160 ms + 70 ms + 3600 ms + 200 ms + 700 ms + 1300 ms
= 6030 ms
```

The biggest difference between the two models is illuminated by removing line three through six in the Prediction app model worst case trace seen above, effectively removing the prediction aspect from the Prediction app model. This gives us a trace of the time it takes to go directly to the app screen from the lock screen by tapping the notification, yielding a time of 4360 ms. This is almost identical to the best case trace for the Revolv app model, which is 4350 ms. The results are similar because the best case trace for the Revolv app model happens when the phone unlocks straight to the app screen, while for the Prediction app model, we can go directly to the app screen by tapping the notification. This means that both open directly into the correct app. For the Prediction app however, this will happen every time, while for the Revolv app, it depends on which app was last active on the phone.

This means that even without the prediction functionality in our app, we can save significant time simply by providing direct access to the app from the lock screen, by means of a notification shortcut. If we add this shortcut to the Revolv app, then we can always go directly to the app from the lock screen, effectively removing the average and worst case from the equation. In that case, the Prediction app is 830 ms faster than the Revolv app if the device is in the expanded notification, but 1670 ms slower if it is not. If we take into account the ALZ prediction accuracy for five devices, then the best case for the Prediction app will happen 89 percent of the time, meaning that on average, the time it takes to toggle a device is $0.89 \times 3530 + 0.11 \times 6030 = 3805$ ms. This is $4360 - 3805 = 555$ ms faster than if we only use the notification shortcut. If we have 100 percent prediction accuracy, the average case would be equal to the best case for the Prediction app, which is 3530 ms. This is only 275 ms better than the case for ALZ.

Predictive systems make it hard to memorize where devices are located in the user interface. One has to scan through the entire list to find the item one is looking for. Depending on how accurate the prediction algorithm is, this could make it take longer than it would have if the devices were stationary. There is a balance between how predictable the behavior of the user is, and how much time is saved by the user memorizing the location of a particular device.

Our system takes advantage of the notification screen to display the most likely items, saving both space and time. For any smart home remote control app with a limited number of devices however, a predictive list can be detrimental to the performance when the user remembers the location of the devices in the list, since they will have to scan the whole list whenever the algorithm rearranges the most likely devices.

Furthermore, we do not need to use the prediction algorithm when there are five or less devices connected to the smart home, since five is the maximum number of devices that can be shown in the expanded notification list. In that case, the algorithm would only rearrange the placement of the devices in the list, which would nullify the benefits gained from the user memorizing the location of the devices. We can therefore save some time and battery by not running the prediction algorithm until the system has more than five devices connected to it.

On average, the Prediction app model is faster than the Revolv app model for the user. We can shave off 4150 ms from the worst case, and 2790 ms from the average case of the Revolv app model if we simply add a notification shortcut that opens the app directly from the lock screen. This is the main advantage that the Prediction app model has over the Revolv app model in our comparison. This means that the greatest impact on the user's interaction time comes from circumventing the app navigation and going straight to the app screen. This functionality is easy to implement, making it a low-cost step that can save a lot of time for the user.

The prediction functionality is not as impactful, we have estimated that it saves an additional 0.5 seconds on average for the Prediction app.

Chapter 6

Summary

In this chapter we summarize the the results with respect to the goals established in section 1.3.

Our first sub-goal concerns **comparing two sequential prediction algorithms with respect to our mobile architecture.** This goal is achieved by first defining the desired algorithm characteristics of our solution in section 2.1, and then comparing two algorithms that fit the criteria, ALZ and SPEED, in section 2.5. We find that SPEED has better prediction accuracy overall, but worse run-time and space characteristics than ALZ. The difference between the algorithms when considering the most likely five devices is less than 11 percentage points. Since our expanded notification list can show five devices, the greater prediction accuracy of the SPEED algorithm is less of an advantage given its greater run-time complexity, especially considering that our system must run on a battery-powered device.

The second sub-goal, **design and model a user interface that minimizes user interaction time by using NGOMSL**, is achieved by exploring the user interface capabilities of Android in section 3.2, and utilizing this knowledge to create a Prediction app in chapter 4. We have designed a mock-up and a NGOMSL model of our system using KLM operators designed for the smart phone, with transition times tested on a Nexus 5 smart phone. Our model takes advantage of Android's notification system to project the devices that the user is most likely to interact with in an expanded notification on the lock screen. Furthermore, if the device is not found in the expanded notification, the user can open into the app screen directly from the lock screen, circumventing the home screen. The resulting NGOMSL model minimizes the time it takes the user to interact with a device on the phone.

The third sub-goal is **compare our design against a commercial smart**

home remote control app by using NGOMSL. In chapter 5, we compare the NGOMSL model of our Prediction app against the Revolv smart home remote control app in relation to how fast a user can toggle a device in each model. The results show that the Prediction app is faster than the Revolv app in most cases. We also found that the greatest increase in time comes from providing a shortcut to the main app screen on the lock screen. By adding the expanded notification functionality, we shave off an additional 0.5 seconds on average.

Our overarching goal is **to create a fast smart phone remote control system for smart homes by adding a predictive element**. In this thesis, we have designed a Prediction app model that utilizes Android's notification system, together with a prediction algorithm, to reduce the time it takes for the user to interact remotely with smart home devices. We have validated the design by comparing it against the Revolv smart home remote control app. The results show that the Prediction app is faster than the commercial alternative in most cases. Furthermore, we have shown that we can easily improve existing remote control apps by adding a notification shortcut to the lock screen.

6.1 Contributions

In this thesis we have made the case that we can improve upon the commercial smart home remote controls by adding a predictive user interface element to the lock screen. We built a model of our system using NGOMSL, and compared it to the Revolv smart home remote control app. We have shown that by designing a user interface to take advantage of prediction, we can decrease the time it takes the user to interact with smart home devices.

Additionally, we compared the two algorithms, SPEED and ALZ, with respect to our mobile architecture. The results of this comparison showed that while SPEED has a higher prediction accuracy, ALZ is more suited for our mobile architecture given that it has better run-time and space characteristics.

6.2 Future Work

The obvious next step is to implement our system on a Android phone, and do an empirical user test to see how accurate our NGOMSL model is. Additionally, this would allow us to compare our two algorithms on smart phone hardware, which would let us determine the battery impact of the differences in run-time and space characteristics between SPEED and ALZ.

We can also improve the algorithm itself by taking into consideration additional aspects of the data source. For instance, SPEED identifies episodes in the

data stream. An additional aspect that none of the algorithms take into account is the time aspect, meaning that two sequential events happening seconds apart or days apart are considered the same sequence.

Another improvement is to add more nuanced interactions with the devices. For now, our system only exposes the ability to toggle devices on and off. We could for example add intensity sliders for light sources, or channel switching for TVs. This requires that we take into account the qualitative changes that this causes in the data source. For instance, if the ALZ algorithm suddenly registers 100 events generated in the span of a few seconds by light dimmer, it would thereon heavily favour the light dimmer regardless of which events are more likely. The challenge then becomes to align the events detected by the algorithm with the actions of the user. If he dims the light, it should register as a single event by the algorithm, even though it may generate hundreds of events.

A further improvement on this idea is to group the actions exposed through the user interface based on tasks, instead of devices. For example, instead of turning on the TV, dimming the lights, and playing a movie, the user could toggle the “Movie Night” action, which does all these things.

Bibliography

Alam, M. R., Reaz, M. B. I., and Mohd Ali, M. a. (2012). SPEED: An inhabitant activity prediction algorithm for smart homes. *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, 42(4):985–990.

Anagnostopoulos, C. and Hadjiefthymiades, S. (2009). Advanced inference in situation-aware computing. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 39(5):1108–1115.

Batran, K. E. and Dunlop, M. D. (2014). Enhancing KLM (Keystroke-Level Model) to Fit Touch Screen Mobile Devices.

Bell, T. C., Cleary, J. G., and Witten, I. H. (1990). *Text Compression*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Card, S. K., Newell, A., and Moran, T. P. (1983). *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.

Cook, D. J., Youngblood, M., Heierman III, E. O., Gopalratnam, K., Rao, S., Litvin, A., and Khawaja, F. (2003). Mavhome: An agent-based smart home. In *2013 IEEE international conference on pervasive computing and communications (PerCom)*, pages 521–521. IEEE Computer Society.

Desai, N., Kaowthumrong, K., Lebsack, J., Shah, N., and Han, R. (2002). Automated selection of remote control user interfaces in pervasive smart spaces. *in: Proceedings of the Hcic Winter Workshop 2002*.

Drury, J. L., Scholtz, J., and Kieras, D. (2005). Modeling Human-Robot Interaction with GOMS. *Interface*.

Feder, M., Merhav, N., and Gutman, M. (1992). Universal prediction of individual sequences. *Information Theory, IEEE Transactions on*, 38(4):1258–1270.

Gopalratnam, K. and Cook, D. J. (2007). Online sequential prediction via incremental parsing: The active LeZi algorithm. *IEEE Intelligent Systems*, 22.

Gopalratnam, K., Gopalratnam, K., Cook, D. J., and Cook, D. J. (2003). Active LeZi: An Incremental Parsing Algorithm for Sequential Prediction. *FLAIRS Conference*.

Gray, W., John, B., and Atwood, M. (1993). Project Ernestine: Validating a GOMS Analysis for Predicting and Explaining Real-World Task Performance. *Human-Computer Interaction*, 8(3):237–309.

Kieras, D. (1996). A Guide to GOMS Model Usability Evaluation using NGOMSL. (313).

Laird, P. and Saul, R. (1994). Discrete sequence prediction and its applications. *Machine Learning*, 15:43–68.

Mozer, M. C. (1998). The neural network house: An environment hat adapts to its inhabitants. In *Proc. AAAI Spring Symp. Intelligent Environments*, pages 110–114.

Polson, P., Irving, S., and Irving, J. (1994). Applications of formal models of human computer interaction to training and use of the control and display unit. *Final report, System Technology Division, ARD*, 200.

Sun, R. (2001). Introduction to sequence learning. *Sequence Learning*.

Wu, C. and Aghajan, H. (2011). User-centric environment discovery with camera networks in smart homes. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 41(2):375–383.

Wu, C.-L. and Fu, L.-C. (2012). Design and realization of a framework for human–system interaction in smart homes. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 42(1):15–31.

Ziv, J. and Lempel, a. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5).

Appendices

6.3 Revolv model

```
Method for goal: Revolv app
1. Get next unit task from task list.
2. Decide: If no more tasks, then return with goal accomplished.
3. Accomplish goal: perform unit task.
4. Goto 1.

Selection rule set for goal: perform unit task
if the task is finding the device then
    accomplish goal: find device.
if the task is opening the app then
    accomplish goal: go to app.
if the task is toggling the device then
    accomplish goal: toggle device.

Method for goal: toggle device
1. Point to device.
2. Tap device.
3. Verify that device was toggled.
4. Return with goal accomplished.

Method for goal: find device
1. Locate device.
2. Decide: If device-found then return with goal accomplished.
3. Accomplish goal: scroll list.
4. Goto 1.

Method for goal: scroll list
1. Swipe up.
2. Return with goal accomplished.
```

```
Method for goal: go to app
1. Accomplish goal: unlock phone.
2. Accomplish goal: open app task.
4. Return with goal accomplished.

Method for goal: unlock phone
1. Swipe up.
2. Wait for system to complete transition.
3. Return with goal accomplished.

Selection rule for goal: open app task
if another app is in focus then
    Accomplish goal: go to homescreen and open app.
if homescreen is in focus then
    Accomplish goal: open app from homescreen.
if desired app is in focus then
    Return with goal accomplished.

Method for goal: go to homescreen and open app
1. Accomplish goal: go to homescreen.
2. Accomplish goal: open app from homescreen.
3. Return with goal accomplished.

Method for goal: go to homescreen
1. Point to home icon
2. Tap icon
3. Wait for system to complete transition.
4. Return with goal accomplished.

Method for goal: open app from homescreen
1. Accomplish goal: locate app.
2. Accomplish goal: open app.
3. Return with goal accomplished.

Method for goal: open app
1. Point to app icon.
2. Tap app icon.
3. Wait for system to complete transition.
4. Return with goal accomplished.

Method for goal: locate app
1. Locate app.
2. Decide: if app-found then return with goal accomplished.
3. Accomplish goal: go to next screen.
4. Goto 1.
```

```
Method for goal: go to next screen
1. Swipe left.
2. Return with goal accomplished.
```

6.4 Prediction app model

```
Method for goal: smart remote prediction app
1. Get next unit task from task list.
2. Decide: If no more tasks then return with goal accomplished.
3. Accomplish goal: perform unit task.
4. Goto 1.

Selection rule set for goal: perform unit task
if the task is expanding the notification then
   accomplish task: expand notification
if the task is finding the device then
    accomplish goal: find device.
if the task is toggling the device then
    accomplish goal: toggle device.

Method for goal: expand notification
1. Point to notification.
2. Swipe down.
3. Return with goal accomplished.

Selection rule for goal: find device
if we are in the expanded notification then
    accomplish goal: find device in notification.
if we are in the app then
    accomplish goal: find device in app.

Method for goal: find device in notification
1. Locate device.
2. Decide: If device-found then return with goal accomplished.
3. Accomplish goal: go to app from notification list.
4. Accomplish goal: find device in app.

Method for goal: go to app from notification list
1. Tap notification
2. Wait for system to complete transition.
2. Return with goal accomplished.
```

```
Method for goal: find device in app
1. Locate device.
2. Decide: If device-found then return with goal accomplished.
3. Accomplish goal: scroll list.
4. Goto 1.

Method for goal: toggle device
1. Point to device.
2. Tap device.
3. Verify that device was toggled.
4. Return with goal accomplished.

Method for goal: scroll list
1. Swipe up.
2. Return with goal accomplished.
```

6.5 Revolv Model Executions

6.5.1 Best case model execution

```
Task: Toggle device (Best case)

Expanded for Revolv:
1. Go to app
2. Find device
3. Toggle device

Assumptions:
The phone is in the users right hand.
The phone is currently on the lock screen.
The Revolv app is the active app.
The phone has no lock-screen security.
The device icon is on the first screen of the Revolv app device list.

Revolv trace:
1. Accomplish goal: go to app.
  2. Accomplish goal: unlock phone.
    3. Swipe up (S)
    4. Wait for system to complete transition (W(0.7s))
5. Accomplish goal: find device.
  6. Locate device. (M)
7. Accomplish goal: toggle device.
```

```
  8. Point to device. (P)
  9. Tap device. (B)
  10. Verify that device was toggled. (M)

1 S + 2 M + 1 P + 1 B + 1 W(0.7s) + 10 statements
= 70 ms + 2 * 1200 ms + 1 * 80 ms + 1 * 100 ms + 1 * 700 ms + 10 * 100 ms
= 70 ms + 2400 ms + 80 ms + 100 ms + 700 ms + 1000 ms
= 4350 ms
```

6.5.2 Average case model execution

```
Task: Toggle device (Average case)

Expanded for Revolv:
1. Go to app
2. Find device
3. Toggle device

Assumptions:
The phone is in the users right hand.
The phone is currently on the lock screen.
The phone has no currently active app.
The Revolv app is running in the background.
The phone has no lock-screen security.
The Revolv icon is on the first homescreen.
The device icon is on the first screen of the Revolv app device list.

Revolv trace:
1. Accomplish goal: go to app.
  2. Accomplish goal: unlock phone.
    3. Swipe up (S)
    4. Wait for system to complete transition (W(0.7s))
  5. Accomplish goal: open app from homescreen.
    6. Accomplish goal: locate app.
      7. Locate app. (M)
    8. Accomplish goal: open app.
      9. Point to app icon. (P)
      10. Tap app icon. (B)
      11. Wait for system to complete transition. (W(0.7s))
12. Accomplish goal: find device.
  13. Locate device. (M)
```

```
14. Accomplish goal: toggle device.
  15. Point to device. (P)
  16. Tap device. (B)
  17. Verify that device was toggled. (M)

1 S + 3 M + 2 P + 2 B + 2 W(0.7s) + 17 statements
= 70 ms + 3 * 1200 ms + 2 * 80 ms + 2 * 100 ms + 2 * 700 ms + 17 * 100 ms
= 70 ms + 3600 ms + 160 ms + 200 ms + 1400 ms + 1700 ms
= 7130 ms
```

6.5.3 Worst case model execution

```
Task: Toggle device (Worst case)

Expanded for Revolv:
1. Go to app
2. Find device
3. Toggle device

Assumptions:
The phone is in the users right hand.
The phone is currently on the lock screen.
The phone has no currently active app.
The Revolv app is running in the background.
The phone has no lock-screen security.
The Revolv icon is on the first homescreen.
The device icon is on the first screen of the Revolv app device list.

Revolv trace:
1. Accomplish goal: go to app.
  2. Accomplish goal: unlock phone.
    3. Swipe up (S)
    4. Wait for system to complete transition (W(0.7s))
  5. Accomplish goal: go to homescreen and open app.
    6. Accomplish goal: go to homescreen.
      7. Point to home icon. (P)
      8. Tap icon. (B)
      9. Wait for system to complete transition (W(0.7s))
    10. Accomplish goal: open app from homescreen.
      11. Accomplish goal: locate app.
        12. Locate app. (M)
```

```
      13. Accomplish goal: open app.
        14. Point to app icon. (P)
        15. Tap app icon. (B)
        16. Wait for system to complete transition. (W(0.7s))
17. Accomplish goal: find device.
  18. Locate device. (M)
19. Accomplish goal: toggle device.
  20. Point to device. (P)
  21. Tap device. (B)
  22. Verify that device was toggled. (M)

1 S + 3 M + 3 P + 3 B + 3 W(0.7s) + 22 statements
= 70 ms + 3 * 1200 ms + 3 * 80 ms + 3 * 100 ms + 3 * 700 ms + 22 * 100 ms
= 70 ms + 3600 ms + 240 ms + 300 ms + 2100 ms + 2200 ms
= 8510 ms
```

6.6 Prediction App Model Executions

6.6.1 Best case model execution

```
Task: Toggle device (Best case)

Expanded for Prediction app:
1. Expand notification.
2. Find device.
3. Toggle device.

Assumptions:
The phone is in the users right hand.
The app is currently on the lock screen.
The phone has no currently active app.
The phone has no lock-screen security.
The device is on the notification list.

Prediction app trace:
1. Accomplish goal: expand notification.
  2. Point to notification. (P)
  3. Swipe down. (S)
4. Accomplish goal: find device in notification.
  5. Locate device. (M)
6. Accomplish goal: toggle device.
```

```
  7. Point to device (P)
  8. Tap Device (B)
  9. Verify that device was toggled. (M)

2P + 1S + 2M + 1B + 9 statements
= 2 * 80 ms + 70 ms + 2 * 1200 ms + 100 ms + 9 * 100 ms
= 160 ms + 70 ms + 2400 ms + 900 ms
= 3530 ms
```

6.6.2 Worst case model execution

```
Task: Toggle device (Worst case)

Expanded for Prediction app:
1. Expand notification.
2. Find device.
3. Toggle device.

Assumptions:
The phone is in the users right hand.
The app is currently on the lock screen.
The phone has no currently active app.
The phone has no lock-screen security.
The device is not on the notification list.
The device is on the first screen of the app list.

Prediction app trace:
1. Accomplish goal: expand notification.
  2. Point to notification. (P)
  3. Swipe down. (S)
4. Accomplish goal: find device in notification.
  5. Locate device. (M)
  6. Accomplish goal: Go to app from notification list.
    7. Tap notification. (B)
    8. Wait for system to complete transition. (W(0.7s))
  8. Accomplish goal: find device in app.
    9. Locate device. (M)
10. Accomplish goal: toggle device.
  11. Point to device (P)
  12. Tap Device (B)
  13. Verify that device was toggled. (M)
```

```
2P + 1S + 3M + 2B + 1W(0.7s) 13 statements
= 2 * 80 ms + 70 ms + 3 * 1200 ms + 2 * 100 ms + 700 ms + 13 * 100 ms
= 160 ms + 70 ms + 3600 ms + 200 ms + 700 ms + 1300 ms
= 6030 ms
```