Master Artificial Intelligence, University Utrecht

Master Thesis

Predicting App Launches on Mobile Devices

Using Intelligent Agents and Machine Learning

Supervisor Utrecht University:
prof. dr. John Jules Meyer

Author:
Niels Denissen, BSc.
(4203313)

Second Examiner:
dr. Mehdi Dastani

Supervisors Avanade:
Boaz Pat-El, MSc. (Technical)
Gijs Ramaker, MSc. (Process)

Almere, June 12, 2015

Abstract

Data rich applications often have to load large amounts of data upon launch. The launch times for these applications, e.g. Facebook and NU.nl, can be improved by prefetching their data prior to use. This requires reliable predictions on what applications the user will use in the near future. In order to perform successful predictions, this research utilizes intelligent agents and reinforcement learning. With it, the devised system is able to successfully predict 45.6% of all applications launched by a user. The intelligent agent framework Jadex provides the communication between the agents and Q-learning is used along with the time-of-day as a reinforcement learning algorithm. The results are obtained via simulations with the LiveLab dataset which contains phone usage from 24 users over about a year time. The flexible MAS allows for many improvements in future work that promise even better results.

Acknowledgements

This thesis concludes the 2 years I spend finishing the master Artificial Intelligence at the University of Utrecht. It was written at Avanade starting November 1st, 2014 and was finished in June, 2015. I would like to thank Avanade Netherlands and a number of people that helped me realize the obtained results, as the slogan of Avanade reads.

My weekly supervisor at Avanade was Boaz Pat-El. He fulfilled the role of technical supervisor and has been a very helpful counselor throughout the entire project. The weekly meetings were always very pleasant and helpful. During the few usual struggles of a project, Boaz was always a good partner to discuss ideas with. From the university the role of first supervisor was fulfilled by prof. dr. John Jules Meyer, whom I would like to thank very much for his advice and trust at the start, and his enthusiasm and feedback during every meeting in the process.

Furthermore my thanks go to the process supervisor at Avanade, Gijs Ramaker, who provided the interns at Avanade with helpful meetings and workshops, and the interns at Avanade with whom many coffee-breaks and lunches were happily shared. Not to forget my family, friends and girlfriend who provided help and many welcome distractions during the past months. Finally I would also like to thank the second examiner from the University of Utrecht, dr. Mehdi Dastani, in advance for his evaluation of this thesis.

Almere, June 2015

Contents

List of Abbreviations

ALC	-	Application Launch Count
API	-	Application Programming Interface
BDI	-	Beliefs Desires Intentions
BI	-	Business Intelligence
BPG	-	Bayesian Policy Gradient
BQ	-	Bayesian Quadrature
BQL	-	Bayesian Q-Learning
CRM	-	Customer Relationship Management
CSV	-	Comma Separated Values
DA	-	Distinct Applications
DROP	-	Goals (D), Roles (R), Subject (O) en Procedures (P) (Dutch abbreviation)
ERP	-	Enterprise Resource Planning
FALCON	-	Fast App Launching with CONtext
FIPA	-	Foundation for Intelligent Physical Agents
GPTD	-	Gaussian Process Temporal Difference Learning
IDE	-	Integrated Development Environment
JADE	-	Java Agent DEvelopment framework
LTE	-	Long-Term Evolution (4G)
MAS	-	Multi Agent System
MC	-	Monte Carlo
MDP	-	Markov Decision Process
MEI	-	Motivated Embodied Intelligence
OS	-	Operating System
PREPP	-	PREdictive Practical Prefetch
RL	-	Reinforcement Learning
SARSA	-	State-Action-Reward-State-Action
SQL	-	Structured Query Language
TD	-	Temporal Difference (Learning)
URL	-	Uniform Resource Locator
XML	-	eXtensible Markup Language

List of Figures

List of Tables

Chapter 1

Introduction

Mobile phones and tablets have become increasingly popular over the last few years. They provide the possibility to quickly and easily connect to others, search information on the internet, play games, navigate your route, etc. Because of this extensive use of smartphones by both the consumer as well as the business world [55, 81], research into improving these devices has become increasingly important.

Besides aspects like battery life, speed is one of the most important features of a mobile phone [83]. People rely on their device to provide them with quick information that they can request on demand. Speed is of crucial importance here as this is one of the main advantages of using a smartphone. Improvements can of course be realized by placing fast processors in the devices and devise new techniques that transfer information from and to the internet (think of LTE and WiFi). But along with improvements of these techniques, more and more data will be shared (from high quality photos to entire databases with information). Instead of improving on hardware only, we could try to make phones a bit smarter, let them predict our actions so that they can prepare some work for us.

This research will attempt to predict which application will be used by a user at what time. This information can eventually be used to prefetch these applications and reduce their load times. In this introduction first a more elaborate problem statement will be given along with the scope for the research in section 1.1. This research was conducted at the company Avanade which will be introduced in section 1.2. Then the research questions that will be answered are treated in section 1.3 along with the methods used to answer each of them in section 1.4. Finally an overview of the rest of this thesis is given in section 1.5.

1.1 Problem Statement

In order to improve the response time of applications, techniques like caching that are common for websites can be used. Most browsers retain certain web pages that have been accessed by the user at an earlier time in their cache. Whenever a user requests the same page that has been requested before, the local cache can be consulted instead of fetching the page again from the internet. This greatly reduces load times for websites.

The same principle is already applied to mobile devices. Numerous mobile apps cache data when someone is using the app in order to provide a quick launch when the application is reopened. An example is Facebook. This application caches the stories that are shown on the app whenever it is exited. Whenever a user reopens it, these stories are shown immediately, after which the newer ones are loaded. This gives the user immediately something to look at, but doesn't show any new data faster. In order to reach that goal, new data should be prefetched before the user opens the app. But to do this, predictions should be made about what application will be used when.

This research aims to find a good way to predict the use of such applications. As will be shown in chapter 2, some overlapping and related work has been done in this area. In these papers, intelligent agents have not been used yet, even though they seem to provide a suitable framework for flexible learning. Also machine learning is not used in these papers and has in fact not been used that often in combination with intelligent agents at all. Since there is a lack of research in these promising areas, the combination will be investigated in this thesis.

1.1.1 Scope

This thesis focuses on investigating the use of intelligent agents in combination with machine learning to predict application launches on mobile devices. It will not cover the actual reduction in load times that will be acquired. Furthermore the actual battery drain and data usage that comes with the prefetches will not be considered. Focus lies on what applications are used when. An investigation will be done to how intelligent agents can be used and what type of machine learning can best be applied. With these results a system will be created that predicts the application launches of users. Finally the quality of the predictions will be assessed using simulated data.

1.2 Avanade

The research in this thesis is conducted at the IT consultancy company Avanade [8]. Avanade was founded in 2000 as a joint venture between Accenture and Microsoft. Because of this, they focus on Microsoft technologies in the projects they do. Lately, Microsoft is investing more and more in the field of machine learning and predictive analytics, offering products like Azure Machine Learning and Revolution Analytics[1]. Avanade is following this trend and because of it has the ambition to provide services in these areas as well.

Predictive algorithms and machine learning thus form an interesting new field for Avanade. This thesis uses machine learning and can enable Avanade to perform real-time predictions. Appendix C will eventually discuss the practical usability of this thesis. A description of the company follows below.

[1]www.zdnet.com/article/microsoft-finalizes-its-revolution-analytics-acquisition/

1.2.1 Company Description

Avanade has over 22,000 employees from all over the world. It offers global IT consultancy, serving customers at more than 70 locations in 20 countries worldwide, dedicated to using the Microsoft platform to help enterprises achieve profitable growth through solutions that extend Microsoft products. Avanade has a comprehensive portfolio of IT and business solutions and has deep industry experience, technology, assets and proven implementation approaches. Avanade provides solutions in the following fields:

- **Application Development**: Gain advantage over the competition by using custom application solutions to drive business processing and data. Avanade delivers custom enterprise applications based on the Microsoft .NET Framework. As a 2014 Microsoft Mobility Partner of the year, this involves enhancing employee productivity across touch and mobile devices. Also the integration of data sources and implementation of cloud, private cloud and traditional systems as a preferred partner for Microsoft Azure is part of Avanade's Application Development solutions.
- **Data and Analytics**: Craft a BI strategy that fits your unique business requirements, integrates with your existing infrastructure, and aligns with your companys goals. Uncover fundamental business value from your corporate data and transform it into a strategic business asset with Avanade Business Intelligence solutions. Avanade Touch Analytics is one such component of a much larger set of capabilities, it provides analytics on mobile devices to empower real-time, online and offline decisions.
- **Collaboration**: Evolve your business workplace to the next generation, using Microsoft-based technologies, where people, processes and technology are connected using collaboration tools that fuel information-sharing and productivity, driving value, profitability, and innovation.
- **Infrastructure Services**: Avanade's business expertise, technology and assets will set up Microsoft-based infrastructure solutions to maximize existing investments, including a line of business applications.
- **Managed Services**: Keeping your business-critical software up and running and up-to-date is of major importance for any business. Avanade Managed Service support, maintain and evolve your enterprise business software.
- **CRM**: It is vital for all businesses to be able to effectively maintain customer relations. Avanade has expertise in designing, implementing, and deploying Microsoft Dynamics CRM solutions to optimize any companys return of investements.
- **ERP**: Avanade's enterprise resource planning solution addresses challenges that industries encounter, like changing economics, shifting industry trends and evolving customer expectations. As the largest partner of Microsoft in this field, Avanade uses the Microsoft Dynamics AX platform to implement projects across the globe.

1.3 Research Goals

From the described problem statement and scope discussed in section 1.1, the research question for this thesis is formulated as follows:

RQ: Can intelligent agents combined with machine learning be used to predict launches of applications for mobile devices?

This question gives rise to a number of smaller questions. These questions will have to be answered first as they are crucial in the choices to be made about the system. Firstly, the use of intelligent agents requires an investigation as various types of agents exist. There exist many frameworks that support these objects and thus the first subquestion arises:

SQ1: Which framework for intelligent agents can best be used?

The other aspect of this research covers machine learning. This area of research is quite large and contains a wide range of different kinds of machine learning. To learn more about machine learning and which of these techniques are most suitable, the following question will be answered:

SQ2: What kind of machine learning should be used in the agents?

In order to implement the techniques that are developed, a mobile platform has to be chosen. Since deployment will not be done in this research, this question is not crucial for this work immediately. In order to actually use the solution eventually though, support for the functions that are simulated is needed and thus an investigation into the various mobile platforms is important. The following question will thus be treated:

SQ3: Which mobile platform should be used?

Finally the devised system has to be tested. Testing can be done in multiple ways and by using various different data sets. As proper testing is crucial in evaluating the created system, the last subquestion is:

SQ4: Which way can the system be tested?

The research will use a different chapter to answer each one of the subquestions. Using the answers to those, the research question will finally be answered.

1.4 Research Methods

Before the research started, some interviews were conducted with various employees of Avanade. Using these interviews the exact direction and scope of the research was determined. The interviews were conducted following the DROP-model. DROP is a dutch abbreviation and stands for Goals (D), Roles (R), Subjects (O) and Procedures (P). It provides a structured way to conduct a successful interview.

For most research questions posed in the previous section, a literature study was needed. Literature was gathered via Google Scholar, which contains a vast database of papers that are accessible using the subscriptions available through the University of Utrecht. The subquestions as well as the section on related work were answered using these techniques. First a set of relevant keywords was used to get an overview of the literature. A selection of most quoted and recent papers was gathered. Several papers that were cited by these papers were investigated as well as papers that referred to the most interesting ones found. This way the most important and relevant works for the question at hand were identified.

In order to design the system, the methodology Tropos [32] was used. This choice is argumented in section 3.3. Furthermore an Agile [3] way of working was adopted. This methodology results in intermediate working versions of the system with limited functionality which allow for easier modifications during the implementation process.

1.5 Overview

In this thesis, first an investigation into related work is described in chapter 2. Then SQ1 will be answered in chapter 3 using an investigation to the frameworks existing for intelligent agents and how they can be used in the application. Machine learning is required to detect patterns in user behavior, the various techniques existing in this field are evaluated in chapter 4, that answers SQ2. Then chapter 5 shortly discusses the mobile platforms that can be used and with it SQ3. Chapter 6 shows the design process of the system and the implementation of this design. The last subquestion SQ4 is answered in chapter 7 in order to define how the results will be gathered. Since multiple parameters have to be investigated prior to the gathering of these final results, chapter 8 describes a short research into the setting of these parameters. After these preparations the actual results can be gathered in chapter 9. Finally the results will be discussed in chapter 10 and a conclusion will be drawn on the research questions in chapter 11. Lastly some suggestions for future work are given in chapter 12.

Chapter 2

Related Work

Before diving into the various sub questions that were posed in the introduction, previous work done in related fields will be discussed. Over the last few years there has been a lot of research to mobile devices as they have become more capable of complex action and are used more often by people. They have various limitations, e.g. battery life and internet connection, that allow for improvements. Furthermore, a lot of interesting information can be gathered from mobile devices, as people tend to bring their device everywhere and use it for numerous actions. All this results in a vast amount of research done in the field of mobile devices very recently.

Research into mobile phone usage has increased with the popularity of these devices. Some research in this area will be discussed in section 2.1. Building on this analysis, various researches have been conducted to extract user patterns from these mobile devices. The learning of these patterns is treated in section 2.2. Using these patterns, others have already tried to prefetch applications on mobile phones. These studies are most relevant for this research as their research questions largely overlap. Section 2.3 treats the papers found on this. Besides using the data gathered from mobile devices for app prefetching, other applications exist as mentioned in section 2.4. This research focuses partially on the use of intelligent agents in mobile devices, related work in this field is described in section 2.5. Finally some related systems are treated in section 2.6.

2.1 Analysis of Smartphone Usage

Quite some papers have been written about the way people tend to use their smartphones and how this data can be used in predicting behavior or improving usability of these devices. Especially this first aspect is of interest. Several papers suggest that the list of recent apps used and thus the sequence of app usage is one of the most influential factors in determining what app will be launched next. A lot of researches focused on the use of other contextual information as well, with varying rates of success. Here a deeper look into what information determines what apps users will use is given.

Smartphone usage differs heavily between users [87], this level of diversity suggests that mechanisms to improve user experience or energy consumption will be more effective if they learn and adapt to user behavior [40]. For the same reasons, patterns will differ greatly between people as well. Possibly what sensors are important indicators of behavior also differs per person.

Besides this diversity, some studies have a clear overall verdict on what context features are most important. The analysis in [36] for instance concludes that correlations between smartphone usage, location and social context (Bluetooth) are most interesting. Others concluded that contextual information, such as time, location, user profile and latest used app, can be used to predict mobile app usage [53]. The correlation between sequentially used apps has a strong contribution to the prediction accuracy. Add to these results the fact that humans operate different on their phones and may have different triggers to use them. For these reasons it might be more interesting to let the system decide which context features to use for itself.

An elaborate study has been done on a large range of mobile phone usage related matters [17, 18]. These include the launching of apps, housekeeping apps on mobile devices (also discussed in [51]), discovering new apps and multitasking on phones. The most interesting work for this research is the launching of apps. To extract information about this from mobile devices, the author created AppSensor. With it he gathers results on when apps are launched, in what order, where etc. AppSensor was used as a basis to implement the app Appkicker. AppKicker includes prediction technology based on PREPP [92]. This system will be discussed further in section 2.3.

2.2 Learning User Patterns

One of the most important tasks that has to be completed when reducing load times for apps, is the elicitation of user patterns on mobile devices. To successfully reduce the load time of an app, the one that will be launched next needs to be predicted and for that user patterns need to be elicited. During the last few years, a lot of research has been done in learning these user patterns. This section will discuss a range of papers that were published in this area and closely related areas and that can provide more insight in how the learning process is done up until now.

2.2.1 AccessRank

AccessRank [42] is an algorithm that predicts re-visitations and reuse in many contexts. It predicts the next most likely action based on past results. Essentially this concept can be applied everywhere, though in this domain AccessRank can be used to determine what app will likely be used next based on the past. The algorithm says nothing about the next time an application will be launched though and thus is incomplete for predicting app launches. PREPP, as will be discussed in section 2.3, uses the same notion but along with a prediction of the next time the app will be used.

2.2.2 Reflection

Reflection [65] is an Android service API that developers can use to predict what users will most likely use next in an application. It is an engine that provides machine learning functionality for apps and can be used by developers. A set of event prediction features is combined using online learning. For the learning process, a large range of algorithms can be used. The authors argue that some applications perform better with a certain predictor than others. Thus diversity in predictors seems to be desirable.

2.2.3 CondOS

CondOS [30] is a concept for an extension of currently available mobile OSs. It proposes a way to learn user behavior on OS level so that OS level functionality might benefit from this as well. The authors propose a design for such a system and discuss topics on what to improve and how privacy can be preserved when using the sensors.

2.2.4 Context Model for App Prediction

In [105] a new context model for app prediction is proposed. It collects a wide range of contextual information in a smartphone and makes personalized app predictions based on the naïve Bayes model. The authors created a home-screen that presents the most probable apps to be used next and showed that this reduces the time users have to search for applications. Through an analysis of data, they found that several contexts such as last application, cell ID and hour of day were important influences. With cell ID, the identifier of a broadcasting tower for cellular phones is meant. This can be regarded as a way to determine one's rough location.

2.2.5 Mobile Miner

Using limited phone resources, Mobile Miner [110] efficiently generates behavioral patterns. The authors find behavior patterns for individual users and across users, ranging from calling patterns to place visitation patterns. This is done by combining features into baskets and use of a weighted rule mining algorithm to generate association rules that represent state transitions. With their results the authors finally created a phone UI that shows the most probable next apps used to the user.

2.2.6 Call Predictor

Call prediction is the topic studied in [95]. It aims to learn a model that predicts when calls will arrive or go out. Call prediction can be useful in planning daily schedules, avoiding unwanted communications and resource planning in call centers. The paper suggest probabilistic ways to determine when someone will receive and make a new phone call. At any given time, this model can generate a list of most likely contacts to be called, creating an intelligent address book. Observations are clustered per hour of day and predictions are done per hour. Call prediction relates closely to this research at this aspect. The agents for app prediction will likely poll for information as well, though at a smaller interval and using machine learning instead of probability theory.

2.2.7 Multi-faceted approach to predicting App Usage

Besides using solely context or preferences in predicting apps, one could combine these facets and create a multi-faceted approach to predicting smartphone app usage patterns [134]. The authors formulate smartphone app prediction as a classification problem. To classify app usage they use App Bags. Phone features are classified with app usages and from this the N most likely apps to occur are extracted along with a confidence level.

In the system all features extracted are combined in an app bag to classify app usage. Then these app bags are compared to other users' app bags. For this the similarity of users' patterns is used to weigh their influence, and the confidence of their predictions is taken into account as well. A limitation of the system is that only the prediction of the first next app is taken into account. Furthermore, these predictions aren't groundbreaking in their accuracy with respect to a very simple prediction based on Most Recently Used. Nevertheless the idea is valuable and is likely to perform well. In section 2.1 it was concluded that users behave in a diverse way and every app could well have a different optimal learning scheme. Using multiple facets in learning could thus prove useful.

2.3 Application: App Prefetching

Some researches conducted fairly recently have been about the same topic as this research is covering, prefetching applications to reduce their load times. This section will discuss and evaluate FALCON and PREPP, each of which show promising results. Furthermore a short comment is made on this technology existing in iOS and Windows Phone 8.1 already.

2.3.1 FALCON

The research leading to the program FALCON [136] uses different features to determine what app will be launched next. Firstly the authors distinguish between trigger apps and follower apps. Trigger apps tend to trigger other (follower) apps in a session, like an SMS might trigger a browser search. Besides that, location is used. The location where apps were used is clustered and used as a feature. Furthermore the time can be of influence, both as the time of day or using yearly and monthly timelines.

Using all of these features, FALCON trains a Cost-Benefit learner using a Knapsack model and tries to decide what app to launch next. It then prefetches apps by calling the prefetch process of an app, this process needs to be created for each app. If it is not available, FALCON loads the default page of the app.

Evaluation

The idea of FALCON is very closely related to the approach in this research. Nevertheless some improvements might be made. FALCON only starts predicting app launches once the first app on the phone has been used, it thus isn't constantly monitoring the phone and predicting what app might be used next. These predictions are only done after a certain launch. Furthermore a single model is learned that combines all of the features. Using different models might turn out to be more accurate as every application and user might have their own characteristics.

Nevertheless, FALCON seems like a very good technique to predict app launches and reduce their load times. Using agents that constantly monitor the system and actively predict even for the first use of an app could improve results. With machine learning better predictions than the statistical methods of FALCON can possibly be reached.

2.3.2 PREPP

In [91, 92] the authors propose a system called PREPP that solves the problem as follows: Given a sequence of content-based apps that a user has used, and the times when the user has used them, can we prefetch content in a timely manner while keeping overall network prefetch costs low? They predict which app will be launched next and after what time using the past launch sequence of apps. Crucial in PREPP is the time at which the next app will be launched. This time has to be short enough to ensure freshness of the data. It is undesirable to prefetch an app an hour before it's used, since the data will be old.

Evaluation

PREPP is improving launch times a lot, but it only learns what will be the next used app. It doesn't learn a user pattern with respect to other features like for instance time and location as FALCON did. For each time period and location, the pattern of apps used can differ. Say a user always uses Google Maps at 17:30 to drive home. Then it doesn't matter what sequence of apps was used before, only the temporal aspect counts here. The sequence is only one aspect that contributes a good prediction, other aspects might improve this further. PREPP most closely relates to this research as it predicts applications at any given time. Besides that, it reports some important results of FALCON as well. Therefore it can be used to compare results with.

2.3.3 iOS and Windows Phone 8.1

In iOS some form of background fetch already exists [131]. This function can be used by apps to specify that iOS should try to prefetch specified data whenever it predicts the app will be openened. iOS thus determines when an app is prefetched. Besides the information via the link in the footnote, no information could be found regarding this. It is expected that agents aren't involved here. Besides Apple, Windows recently offers developers the chance to prefetch content as well[1]. With regard to literature, the same holds as for iOS. With this lack of information, it will be assumed no agent technology was used here either. These techniques can thus not be used for evaluation, but are not expected to outperform recent literature on this topic.

[1]`http://blogs.windows.com/buildingapps/2014/05/01/`

2.4 Application: Other

Besides predicting user behavior in mobile devices, other fields are interested in finding user patterns as well. As these are closely related techniques, it might be beneficial to look at some of these fields. Some applications of these techniques in other areas will be discussed here.

2.4.1 Extract Mental States

Several studies have been done to infer mental statuses of the user from usage data on a mobile device. This could prove as an interesting intermediate step in inferring what a user will do. These mental states might prove to be good predictors for app launches as well.

In [102] a system is proposed to retrieve user traits from a limited amount of applications present on a mobile device. Another usage of these user patterns is discussed in paper [50]. The authors propose a way to infer logical status of the mobile device from this data. Logical statuses are statuses that say something about the state of the user, like isAlone or isBusy.

2.4.2 Privacy

Besides all the good things that come from using user patterns, there are also privacy concerns. In [29] the authors use the found patterns to protect the privacy of a user by allowing them to reveal a limited amount of information to a requesting service. This topic takes a very important place in Artificial Intelligence and thus also in this research, especially where information is heavily shared among users as is done in community learning.

2.4.3 Predicting Electricity Consumption

Recently, a study [58] has been done into predicting load on the electricity net for the next day, considering patterns of previous days. The data on input is first clustered using a Self Organizing Map (SOM) and predictions are done using an Artificial Neural Network.

2.4.4 Network Load

A study has been done to the diverse usage patterns of smartphone apps via network measurements on a national level in the US [133]. From this the authors conclude that a considerable fraction of apps is used in particular regions. This information can be used to optimize content fetching time in LTE and WiFi.

2.4.5 Webpage Prefetching

PocketWeb [66] describes a technique to predict user browsing for mobile devices. It tries to predict what website the user will visit next and load these pages before the user opens them. 60% of the URLs could be predicted within 2 minutes for 80-90% of the users. This is only done though, for the 2 most used URLs. The authors argue this is because there is always a small set of URLs that defines most of the usage. PocketWeb is an extension of Pocket Cloudlets [60] that prefetched these pages at night while charging the phone. The biggest improvement is that now it also works with dynamic content.

2.4.6 Media caching

The user experience on mobile devices can be enhanced largely by efficiently caching media. These are large files that mostly need a lot of time to load. A study [2] has been done to how the load of networks could be reduced. Besides that studies [27, 77] to cache advertisement videos on mobile devices have been done as well. These show that keeping a cache for these videos pays off well, predicting them could be beneficial in this area.

2.4.7 Storage

Although this research is seeking a different approach to improve load time of apps, storage performance can greatly affect performance of several applications on smartphones as well. This storage should be revisited to improve the performance as is reasoned in [59]. Another study related to storage [82] and smartphone application launch was done by utilizing information about I/O behavior. By studying reads and writes, a system is implemented that reduces application delay by prioritizing reads over writes.

2.4.8 Energy Savings

Even though outside the scope of this research, a lot of research has been done to energy savings in mobile devices [119]. For future work these considerations could be taken into account.

2.4.9 Smart Home

A more advanced approach to using user patterns in practice are smart homes. These homes will learn the behaviors of its inhabitants and adjust all sorts of actions to it. These include climate control, lights, music, etc. Intelligent agents are being applied to smart homes already. MavHome [31] is proven to predict user behavior in homes accurately. This research could eventually prove useful in the mobile domain as well.

2.5 Intelligent Agents in Mobile Devices

Already in 2005 researchers anticipated on the predicted importance of mobile devices and devised the idea of using intelligent agents with them [86]. The authors investigate the possibilities of agents and implement an application called Genie on a mobile device. This application uses BDI agents to create a context-sensitive mobile tourist guide application. Various agents communicate with each other in the system to provide predictions on what a user will want to know and anticipate on that. The idea of using agents in mobile devices has thus successfully been implemented, which emphasizes the usefulness of this notion in this research.

Another study implements intelligent agents for the Android platform [4]. The Andromeda

Platform is a result of this study and provides a platform for simple agents to be used in Android. More on intelligent agents used in this research can be found in chapter 3.

2.6 Related Systems

Several works that relate to this work in a range from doing somewhat the same to being only remotely related to the topic have been discussed. Besides this, other related systems that perform similar tasks exist as well. Anticipatory systems and Recommendation systems are two of these that could prove interesting to this research.

2.6.1 Anticipatory Systems

In [94] the difference between a predictive and an anticipatory system is made. An anticipatory system is defined as follows: ”A system containing a predictive model of itself and / or its environment, which allows it to change state at an instance in accord with the model's predictions pertaining to a later instant”.

The use of sensor data in context elicitation is discussed with a lot of examples of existing anticipatory systems. A description of Machine Learning techniques used in various subjects is available as well as an overview of what sensors can be used to identify which features. In the article, context sensing domains and relevant machine learning techniques for it are evaluated. For Activity Classification a lot of ensemble learners as well as Bayesian networks are suggested. Besides that, for scene classification, K-means clustering is the most used approach. This may give some insight in what to use when creating learning agents. The classification task could classify which prediction agent works best.

2.6.2 Recommendation systems

Somewhat related as well are recommendation systems. Systems that recommend material based on what a user has used in the past. It differs slightly because it predicts the possible interest in new content, not existing content. Recommendation systems have been developed for mobile devices, like AppJoy and GetJar. AppJoy [135] recommends applications to install based on app usage of a user. GetJar [104] does the same thing, but in addition the authors created an entire app store that utilizes this functionality to recommend new apps.

Collaborative Filtering is a dominant approach in these systems. The technique can be used user-based (evaluate similarity among users) or item-based (evaluate similarity among items) [17].

Chapter 3

Intelligent Agents

In order to create a system that predicts application usage and prefetches apps, a certain degree of autonomy is needed. This autonomy is required since the system has to decide for itself when to prefetch applications. Therefore the system needs to act proactively and has to react to changes of its environment. All of these aspects hold for the notion of an agent as opposed to objects we know from Object Oriented Programming. Wooldridge provides an introduction to what Intelligent Agents are and how they can work together in a Multi-Agent System [128, 129].

The first subquestion posed in the introduction will be treated here:

SQ1: Which framework for intelligent agents can best be used?

This chapter will discuss what intelligent agents are. What different types of agents exist will be treated in section 3.1. A framework to implement the system in is needed, since creating a multi-agent system from scratch requires a lot of work. Various frameworks are discussed in section 3.2. To create a multi-agent system in an appropriate way, a design methodology can be used. Some of these are discussed in section 3.3. Finally the posed question will be answered in section 3.4.

3.1 Types of Agents

In the literature on agents there is no universally accepted definition of an agent. Nevertheless an attempt at it is made by Wooldridge [129]:

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.

An agent though, needs not to be intelligent. A thermostat is an often used example of an agent that decides autonomously whether to switch the heater on or off. Although it is an agent, it is not generally called intelligent. This notion requires some more capabilities that are characterized by Wooldridge and Jennings [129]:

- *Reactivity* ensures that agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives.
- *Proactiveness* means that agents are able to exhibit goal-directed behaviour by taking the initiative in order to satisfy their design objectives.
- *Social ability* states that agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

To use agents, at least some intelligent behavior is needed. The agents need to react to their environment which is the user of a phone, they need to be proactive in making decisions about what apps to launch when and they have to be social to communicate with each other on their predictions.

There are thus different kinds of agents available. The thermostat as discussed earlier is said to be a purely reactive agent, one that responds to its environment directly. The more intelligent agents are often realized using the intentional stance [35]. These agents are called BDI agents and since they are the most sophisticated type of agent they are most interesting to use. BDI agents will be discussed next.

3.1.1 BDI agents

The notion of BDI agents is based on the idea that there exist beliefs, desires and intentions in practical reasoning. People have believes about the world, they have desires they would like to achieve and a set of intentions that represent desires with a form of commitment. A procedural reasoning system as depicted in Figure 3.1, can represent this model and summarize how a BDI agent works.

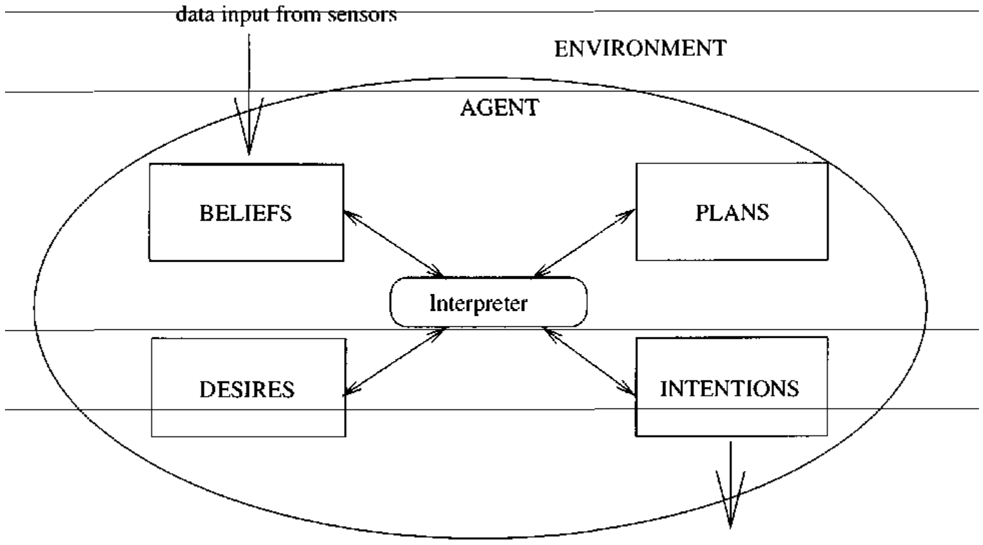


Figure 3.1: A Procedural Reasoning System [128]

Via input that is retrieved using sensors the belief base of an agent is updated. Using these beliefs and the desires the agent has, the interpreter selects a set of intentions that will be executed. The interpreter can use the plan base to select plans needed to achieve intentions.

Being the most sophisticated agent at the moment it is researched a lot. Researches have already devised an improved method to select plans in BDI agents [88] and to better validate them [113]. Using BDI agents as the main agent for the system, there are not really any mistakes that can be made. BDI agents can be used as purely reactive agents where necessary and are capable of being extended to fully reasoning agents when desired.

3.2 Agent Frameworks

When implementing agents, a platform on which they can run is needed. There already exist numerous platforms on which all sorts of agents can be implemented. Most of these frameworks are JAVA-based to provide cross-platform functionality.

There is a lot of literature available on languages, tools and applications used in multi-agent programming [20] and a survey to MAS programming languages and platforms [19]. These papers were written a couple of years ago. This means that some of these are outdated, especially for mobile device considerations. Besides these older ones, some have been expanded and improved and they will be considered in this section.

First a number of assessment criteria will be devised on which the various frameworks that are taken into consideration will be tested. Then the frameworks will be discussed, evaluated according to the assessment criteria and explained to some extent. Finally everything is evaluated and conclusions will be drawn on which framework to use.

3.2.1 Assessment criteria

In order to evaluate the various agent frameworks that exist, some criteria to assess them on are required. These criteria will be listed here and can also be found in table 3.1 where the comparison is made. Differences are made between crucial and desired aspects, the latter are displayed italic in the table.

BDI
As seen in section 3.1 on Types of Agents, a BDI agent is currently one of the more advanced reasoning agents. Since this type of agent could be used, support for it is required.

Mobile support
Since the goal of the research is to reduce app load times on mobile devices, these mobile devices would ideally be supported directly by the framework. As the research doesn't include any live tests, but only a simulation, this is not a hard requirement.

Debugging
For quick and easy development, the right debugging tools are valuable. Since some frameworks

utilize multiple programming languages, some aren't easily debugged. Again, this is a desired property and thus not a hard requirement. If necessary, debugging can always be done by printing the information needed to a console.

Community
A large community and widely used framework often provides more support for development in the form of tutorials or internet fora. These communities can be important in the development of the application when help is needed. A community is also a soft constraint.

Free
Of course the costs have to be taken into account as well. A free and open source solution is preferred. This is considered a hard requirement as software tends to be expensive.

FIPA compliant
FIPA [54] is a developed standard for agent frameworks. This ensures that multiple frameworks can work with each other. As only a single framework will be used it's not crucial to have a language that is compliant to the FIPA standards, but it is always a plus to have. Apart from the expansion possibilities it brings, the standard ensures some rules are followed that are known to be working and have been verified extensively.

3.2.2 JACK

JACK [6] is a popular agent platform that has been developed for quite some time already [52]. It comes with an entire own development environment(IDE). BDI agents are fully supported, unlike mobile devices. A lot of tools are provided for debugging purposes and the community for JACK is fairly large. Unfortunately it is a commercial product and thus not freely available. Also, it isn't FIPA compliant by default, but can be made this way using a plugin called FIPA JACK.

3.2.3 Janus

Janus [56] is developed as the Virtual Machine for SARL. SARL is an agent programming language designed to make agent development faster and easier [100]. It fully supports Java via which other APIs can be accessed eventually. BDI agents are supported, but no explicit support for mobile devices is available. Furthermore debugging can be done via java when the agents are created using the SARL language. The community is small though and the product has been developed only recently. Finally it is freely available and FIPA compliant.

3.2.4 JADE

JADE [117] was the first framework to be developed and is the most used one in the research field as well. A comparison of various JAVA-based agent platforms done in 2003 [122], concludes that JADE is the best option to use. Furthermore some lessons learned from using JADE are discussed in [12].

Basically JADE provides a platform for agents to run on, communicate on, find each other and their services and all that in a cross platform manner. BDI is not supported in JADE, but several expansions on it exist that do support this and will be discussed below. Mobile devices are also supported via an extension called JADE-LEAP. Using this, agents can run on the device itself the same way as JADE runs on PCs, or it can be used as a front end with a PC as back end. Support for Android phones [15] and Windows Mobile .NET is provided. The Windows Mobile platform is deprecated though and replaced by Windows Phone which is not explicitly supported. Besides that, agents here cannot reside on the device but the app can only connect to an agent network externally.

Various debugging tools are available to JADE developers and the community is quite big. A lot of tutorials, internet fora and also a book on Jade from 2007 [13] is available. JADE is freely available and completely FIPA compliant. Besides all this various extensions have been developed for it, these include JESS, BDI4JADE, 2APL and JADEX. Most features of JADE hold for these extensions as well, only exceptions will be mentioned. Over the years, JADEX has been separated from JADE and now runs in its own environement. Because of this JADEX will be discussed in a separate section.

JESS

Jess [101] is a rule-based reasoning engine that can easily be used to add intelligence to JADE [10]. It only provides the reasoning part of a BDI. Jadex actually uses the RETE functionality used in JESS for the implementation of its production rule system.

BDI4JADE

BDI4JADE [11] is exactly what the name says, a BDI layer built on top of the JADE framework [85]. It uses Java only, just like the newest version of Jadex. Since its initial launch this framework was updated with capability relationships as described in [84]. Unlike Jadex, BDI4JADE still works with JADE itself. Its community is small and the framework is used by only very few people, partly because it was released fairly recently.

2APL

2APL [1] is a BDI multi agent programming language built on the JADE platform [33]. The framework seems to be a bit more popular than BDI4JADE, but isn't as popular and broadly used as Jadex. Besides that, it has no explicit support for any mobile devices apart from via JADE.

3.2.5 JADEX

Jadex is a framework that provides for the implementation of BDI agents in Java [24, 25]. It can work in a standalone version as well as with JADE by using it as a middle layer. For a long time, Jadex worked with XML to specify its agents' beliefs, goals, etc. This reduced the ease with which other existing applications could be used from within the agent and made testing more difficult. Recently though, Jadex has been upgraded to work with Java only [96], therefore simplifying the use of external API's inside the agents. (This update also resulted in Jadex only being able to run as a standalone package, thus there is no integration with JADE possible anymore.)

Jadex is being developed for quite some time already and because of that provides many tutorials and support for development. It has been used in various applications and can be used with Android. For this purpose other tutorials are available for use with the standalone version. It also has a fairly large community and many users based on the amount of downloads on SourceForge[1], the website they distribute their software on, and the fora found on the internet regarding this topic.

3.2.6 Jason / AgentSpeak

AgentSpeak has been one of the most influential abstract languages based on the BDI architecture. Jason has been created as the first fully-fledged interpreter for a much improved version of AgentSpeak [57]. A book has been written about programming a MAS in AgentSpeak using Jason [21]. This combination thus provides BDI functionality and via the possibility to run Jason on JADE has the possibility to run on mobile devices. Furthermore a plugin to Eclipse or even a separate IDE called jEdit is provided via which debugging is made more easy. Also FIPA compliancy can be guaranteed via JADE and the community for Jason is quite large.

3.2.7 Evaluation

Table 3.1 shows all discussed aspects for each of the frameworks mentioned. The first two columns, not written in italics, represent the hard constraints while the others form the softer ones. Jadex and Jason are the only frameworks that support all demands that were required and desired. They have a larger community which gives them a slight advantage over BDI4JADE and 2APL. From the two, Jadex is chosen to provide the BDI agent functionality in the system.

	BDI	Free	*Mobile support*	*Debugging*	*Community*	*FIPA compliant*
JACK	√	X	X	√	√	√
Janus	√	√	X	√	-	√
JADE	X	√	√	√	+	√
JESS	X	√	√	√	√	√
BDI4JADE	√	√	√	√	-	√
2APL	√	√	√	√	O	√
Jadex	√	√	√	√	√	√
Jason	√	√	√	√	√	√

Table 3.1: Comparison of agent frameworks

[1]http://sourceforge.net/projects/jadex/

As it is difficult to assess the various measures for each framework, the provided table is to be considered a guideline. The aspects measured are all taken from the corresponding literature or via an internet search on the topics.

3.3 Development Methods

Since the design of a multi agent system can be a very complex task, development methods exist that are often very useful in constructing these new systems. A lot of these methods are discussed in the Handbook on Agent-Oriented Design Processes [32] by Cossentino et al. Some of the most interesting methodologies for our purpose will be discussed here, along with another called Prometheus, which is not described in the book. Finally a short evaluation will be given.

3.3.1 Methodologies

The most appealing methodologies will shortly be discussed. These include GAIA, Tropos, Prometheus and an Agile Multiagent Software Engineering methodology.

GAIA
Gaia is the oldest methodology described. It was developed by Wooldridge et al. and is applicable both in the macro-level (societal) and the micro-level (agent) aspects of systems [130, 137]. The original method has been expanded to work with agent design and iterative development by Gonzalez et al. [46]. Besides this expansion, a lot of work has been done in the usage of Gaia for engineering JADE agents [78, 79, 80, 16], as we have seen in the previous section. As is the case with a lot of existing software solutions, chances are that a multi-agent system has to be adapted after deployment. How to handle this in the Gaia methodology is discussed by Cernuzzi et al. [28].

Tropos
Tropos as discussed in [32] provides a way to develop BDI agents including a tool to create them in Jadex. Jadex has been the framework of choice, making Tropos a good match. Tropos also provides support for a macro and micro-level of designing. It goes even further by enabling the developer to generate Jadex code.

Prometheus
The Prometheus methodology [89] can be used to develop BDI agents. It provides three design phases: system specification phase, detailed design phase and architectural design phase. Prometheus is applied in the development of agents for the JACK agent platform. The idea of this is similar to Tropos. Prometheus though was developed for JACK, a framework that isn't used.

Agile Multiagent Software Engineering
When working in a team on a single software product, agile is an increasingly popular development method. This method can also be applied to multi-agent systems as is done by Domann et al. [37]. This doesn't provide a true alternative to the other methodologies as there will not be

a team of developers. The agile way of working though is interesting nevertheless and can be applied in the creation of this system.

3.3.2 Evaluation

All discussed methodologies provide a good guideline for agent development. Since the system developed will not require a very complex configuration, most development methods will suffice. Since an Agile approach provides a flexible way of developing the system, this will be adopted at implementation. Taking into account that Jadex is the framework of choice (see section 3.2), Tropos is considered the most suitable choice as it is designed to work directly with it.

3.4 Conclusion

This chapter was concerned with determining how agents can be used in the research. There exist various types of agents, from reactive to intelligent. The type that was chosen, BDI, supports the intelligent aspects of agents, but is also capable of use in a reactive way. Next an evaluation was done between various different agent frameworks. Most of these frameworks provided the most important aspects required. Only two of them though, Jadex and Jason, satisfied all constraints that were identified for the platforms. From these two, Jadex was chosen. Finally Tropos will be used to design the macro level of the agents and the more detailed design of each separate agent. Among other options for these design methodologies, Tropos fitted best with the choice for Jadex.

Chapter 4

Machine Learning

In order to create a system that learns from past behavior of a user, some form of artificial learning is needed. Different types of learning that exist in the literature on machine learning will be discussed and some algorithms from relevant sub-fields reviewed. Besides the standard ways of machine learning, there exist many interesting alternative methods. Also, some great improvements can be obtained when combining multiple learning entities in a form of cooperative machine learning. Finally, in order to implement these techniques, some existing programs might be used, reducing the programming effort required.

The second subquestion posed in the introduction will be treated here:

SQ2: What kind of machine learning should be used in the agents?

To discuss all of these topics this chapter is organized as follows. First the 3 types of machine learning will be discussed in section 4.1. From this a type of machine learning is identified as relevant for the system, namely reinforcement learning. This form of learning will be treated more elaborately and some aspects needed will be identified here. Having discussed which type of learning is required, various reinforcement learning algorithms are treated in section 4.2 and some interesting alternative methods in section 4.3. Furthermore, since cooperating agents might prove very useful, section 4.4 discusses cooperative machine learning. Finally some software libraries regarding machine learning are mentioned in section 4.5 and an evaluation of the chapter and discussion of what will be used can be found in section 4.6.

4.1 Types of Machine Learning

In the current literature, a distinction can be made between three types of machine learning: supervised, unsupervised and reinforcement learning. In supervised learning, the learner receives example inputs along with their outcomes and should learn to map the inputs to the correct outputs. In unsupervised learning the desired output is not known and thus the learner has to find a structure in the input by itself. Finally reinforcement learning requires a learner to achieve a certain goal in a dynamic environment, without some teacher explicitly stating what is correct but only giving an indication of whether it has come close to this goal.

When predicting app launches, the correct output for the given input is unknown beforehand, rendering supervised learning not applicable. Even stronger, because the behavior of the user is changing there will never be a universal truth as to what prediction is correct. Thus data can never be labeled, the correctness can only be guided by evaluating a measure of accuracy. Unsupervised learning is not applicable either since it works without any label, error or reward signal. There is a way to evaluate the results though, by using the time an app was launched by the user. In summary there is a clear goal, the environment is dynamic and while there is a way to evaluate the results, the outcomes aren't explicitly stated. From this it is concluded that reinforcement learning needs to be used to learn the behavior of a user. A short explanation of it will be given next, followed by an overview of its aspects.

4.1.1 Reinforcement Learning

As discussed before, reinforcement learning can be defined as learning without the existence of a teacher that provides 'training examples'. Only experience can be used to evaluate results. In reinforcement learning agents [99] a mathematical formalism is often used to ease the analysis of systems. This formalism is called a Markov Decision Process (MDP). In an MDP the environment is modeled as a set of states and actions can be performed to control the system's state. The goal is to control the system in such a way that some performance criterium is maximized [125]. This performance criterium can be expressed as the reward received from performing an action in a certain state.

Even within the field of reinforcement learning, numerous different types of learning exist. The most important aspects that can be identified in these various algorithms will be discussed next, along with an evaluation of the aspects needed.

4.1.2 Aspects of Reinforcement Learning

In order to effectively evaluate what algorithms within the domain of reinforcement learning are needed, some aspects of these will be discussed. For every aspect a short description is given as to whether it will be useful or even necessary. These aspects will eventually be used to assess the algorithms discussed in section 4.2.

Markov Condition and Uncertainty

Reinforcement learning often assumes that a condition called the Markov Condition holds on an MDP. This condition states that any observation made by an agent must be a function only of its last observation and action (plus some random disturbance). When observations made by the agent are not sufficient to summarize all information about the process, the history of observations and actions has to be taken into account as well, resulting in a non-Markovian condition. Furthermore there can be uncertainty present in state transitions. Some algorithms are able to explicitly model this uncertainty.

It's likely that the Markov Condition will not hold for this system. The reading of some sensors will never be sufficient to represent the entire state of a human using a smartphone. This results in uncertainty in state transitions. It is thus desired to be able to deal with uncertainty, mostly because of the lack of observability. This observability will be treated next.

Full vs Partial Observability

Full observability of an MDP is reached when an agent always has the ability to distinguish a certain state over another. When the environment cannot be observed fully by an agent though, because of a lack of decent sensors for instance, the environment is said to be partially observable and an agent might not be able to distinguish between all states. Such processes involving partial observability are called Partial Observable Markov Decision Processes (POMDP) [109] and they imply non-Markovian observations.

The action of opening an app will by far not be the only action that results in the environment entering a certain next state. Starting a browser search for instance could follow from something read on paper and have nothing to do with any observations made on the phone. Because the environment thus is only partially observable, algorithms that require full observability of the MDP are less preferable as they will most likely perform worse.

Model-free vs Model-based

In reinforcement learning it is useful to have a model, so that you know which states exist and how state transitions work. When this is available, model-based algorithms can be applied. These algorithms assume that such a model is present. In many complex systems though, such a model is not available. Whenever this is the case, model-free reinforcement learning techniques have to be used.

An explicit model of the environment is not available initially in this system. This requires one to focus on model-free reinforcement learning. Since the system will be only partially observable, such a model will never be able to be determined either. From this it is concluded that being model-free is a crucial aspect of the algorithm.

Offline vs Online

Learning can be done in an online or offline fashion. Online means that the agent alters his decisions while performing actions in the environment, so he is learning while acting. Offline learning means a simulation is run for the agent to learn from. This requires knowing the model of course. Whenever possible offline learning can be very practical since a simulation can be done numerous times without having to use the actual system yet.

Since a real-world entity has to be modeled in this system, online learning is required. As there is no model of the system available, it was concluded before that learning has to be model-free. Offline learning isn't even possible when there's no model available, as no simulation can be run in this case. Being able to learn online is therefore a crucial aspect.

Exploration vs Exploitation

A very important aspect of reinforcement learning is the trade-off between exploration and exploitation. Of course the learned information should be exploited to predict future events more precisely, but especially in a dynamic environment it is very important to keep exploring other options. This is needed to keep the policy up-to-date, but also to be able to improve the policy. Without exploration, the system is generally not able to improve.

This trade-off between exploration and exploitation is ever present and discussed broadly in the domain of reinforcement learning. Because of the dynamics of this system, exploration will be very important. The used policy has to be able to alter quickly. Nevertheless, exploitation is necessary to be able to learn patterns in behavior. This leads to believe no explicit conclusion as to what is more important can be reached beforehand. Since all algorithms account for this trade-off, this aspect will not be considered in the evaluation of the algorithms.

Fixed, Indefinite or Infinite Horizon

A distinction is often made in the types of tasks that can be performed by an agent in an MDP. A type means the number of actions to be performed before the goal is reached. One type is finite, fixed horizon tasks. These use a fixed number of steps to reach a goal state of the MDP. In indefinite horizon tasks, action profiles can have arbitrary lengths but by using goal conditions they can come to an end. Finally in infinite horizon tasks the system does not end at all.

For the system it is fairly clear to see that infinite horizon is the way to go. A goal state or goal condition upon which the system ends is not known. Predictions on user behavior are never finished as the user keeps using his phone.

On-policy vs Off-policy

Another aspect of reinforcement learning details how policies are investigated by an algorithm. A policy is simply the rules used by an agent to select an action from a certain state. Learning can happen on-policy or off-policy. On-policy means that only the current policy of the agent is evaluated. To ensure exploration, this policy should be altered from time to time. A benefit of this technique is that it will run fast, but a downside is that exploration is less involved as in off-policy techniques. Using off-policy namely, besides only the action belonging to the current policy, other actions are investigated as well. This ensures a broader investigation of possibilities, but requires more computing time.

Whether on- or off-policy learning is more desirable is not clear entirely. On-policy ensures that the algorithm runs fast, resulting in small computing overhead and quick decisions. On the other hand off-policy learns far faster as it evaluates multiple actions at the same time. Since computing overhead is outside the scope of this research and the speed of learning is of great importance in the fast changing environment of mobile phones, off-policy learning will be preferred.

Continuous vs Discrete Space

An MDP has various states and actions that can be performed from these states. These states and actions can be represented by continuous or discrete variables. When continuous variables are used, the state space is often infinite. An algorithm either can or cannot cope with these continuous or infinite spaces of actions and states.

Since states are defined from sensor data retrieved from the phone, they are represented by somewhat continuous data (the digital sensor discretizes the values technically). Besides continuous states, continuous actions are used as the actions consist of an app and time to launch it (time is continuous). Nevertheless, all of these input variables can easily be discretized so that finite states and actions are the result. An example of discretizing can be given in time. All time units that happen in one minute can be classified to belong to that minute per week. So everything between 10:20 and 10:21 on a Wednesday belongs to a single state. Besides discretizing the space there are other methods to deal with continuous state and action spaces [120]. Thus these aspects of continuous and infinite spaces aren't crucial in our system. Being able to handle these spaces can be desired though, as values then do not have to be discretized.

Conversion Speed

The conversion speed of an algorithm denotes the time it takes for the algorithm to converge to a hopefully optimal policy. Algorithms ideally would search the entire space elaborately to find the optimal policy and converge quickly as well, but this will always be a compromise to a certain extent.

In this application it is desirable to have a high conversion speed. Smartphone users may often delete or add applications and expect apps to work quickly. This implies that it's required to learn patterns as soon as possible. Nevertheless, conversion to an optimal policy will never be possible, since the optimal policy will most likely not exist. Or if it exists, the agent will not find it due to a lack of observability of a smartphone user. It is thus required to have a fast conversion speed.

4.2 Reinforcement Learning Algorithms

The previous section concluded that reinforcement learning will be used. Thereafter a description of the various aspects was given. Now the various algorithms that exist within this field will be discussed. Figure 4.1 shows a general algorithm that holds for these features just discussed. This gives a very rough outline of how the algorithms mentioned in this section work. In the algorithm all occurrences of the 'S' stand for state, 'T' for time, 'R' for reward and 'Q'/'V' for quality/value of a state.

First the most basic techniques, namely those using Temporal Difference Learning will be discussed. After that some other interesting reinforcement learning algorithms will be treated, some of which are taken from [125] and [115]. Finally an evaluation using the aspects discussed in section 4.1 is done to conclude on the found algorithms.

```
for each episode do
  $s \in S$ is initialized as the *starting* state
  $t := 0$
  repeat
    choose an action $a \in A(s)$
    perform action $a$
    observe the new state $s'$ and received reward $r$
    update $\tilde{T}$, $\tilde{R}$, $\tilde{Q}$ and/or $\tilde{V}$
    using the experience $\langle s, a, r, s' \rangle$
    $s := s'$
  until $s'$ is a goal state
```

Figure 4.1: A general algorithm for online Reinforcement Learning [125]

4.2.1 Temporal Difference Learning

The main idea of temporal difference (TD) learning [114] is that learning values can be updated while not finished with a trial yet. These TD methods thus learn their value estimates based on estimates of other values, a process called bootstrapping. No model of the MDP is needed and they can be applied online without having to sweep the entire state space. The most basic and popular method to estimate Q-value functions in a model-free fashion is by using Q-learning. Q-learning is an off-policy learning algorithm, algorithms like SARSA and Actor-Critic Learning are variants on it that work on-policy. These thus learn the Q-values of the policy the agent is actually executing. Benefit of these is that there is no need to evaluate all actions' Q-values.

Q-learning

Using Q-learning [123], an agent can estimate a model of state transition probabilities of the environment, but the state transition probability must be fixed (so the environment should be a Markov Decision Process). In order for this to apply, the behavior of a user on his phone has to be modeled by a Markov Decision Process. There exist successful attempts at predicting human intent using MDPs [69], so there is good reason to believe Q-learning can be used. Q-learning then estimates Q-values for actions based on the reward it receives for these in a certain state. Basic Q-learning does not support partial observability or dealing with uncertainty and it works only with finite state/action spaces.

There have been several extensions developed for Q-learning. Some that focus on the balance between exploration and exploitation, like Bayesian Q-learning [34] and SA-Q-learning [49]. Other examples of extensions are Speedy Q-learning [9] that seeks to speed up convergence and GQ(λ) [67] that addresses policy selection. As Q-learning learns values for each possible action simultaneously, learning can happen quite fast, depending also on how the parameter for learning rate is set.

SARSA

SARSA is closely related to Q-learning, but it incorporates on-policy learning. So the agent will always follow the current policy and exploration has to be done on a higher level by changing the policy from time to time. SARSA is said to be especially useful in non-stationary environments where one will never reach an optimal policy. It is also useful if function approximation is used, since off-policy methods can diverge in these cases [125].

Because of its off-policy learning, only a single action is evaluated per state. It takes SARSA a long time to evaluate all possibilities and thus learning is quite slow. This is the case in this problem domain assuming no prior knowledge is present. With prior knowledge implemented, SARSA could perform better.

Actor-Critic Learning

Actor-critic learning represents another class of algorithms that precedes Q-learning and SARSA. This type of learning separates the policy from the value function. An advantage of this is when there are many actions or the action space is continuous, there is no need to consider all actions' Q-values to select one. Furthermore they can learn stochastic policies naturally and can use a priori knowledge about policy constraints [125].

Actor-Critic learning again uses off-policy learning, which learns slower than on-policy as explained in the section on SARSA.

4.2.2 Batch Reinforcement Learning

Originally, batch RL required a set of transition samples to be known a priori from which the learning system then derives a solution [61]. It was originally intended for supervised learning only. Later though it was revisited to work without an a priori fixed set of training experience. The benefits coming from this are stability and data-efficiency. Compared to Q-learning, batch learning converges significantly faster and thus has more use in real-world systems.

One of the benefits of batch learning is that it handles the problem of exploration overhead via experience replay. In learning Q-values, when the value of state s_t changes, the value of s_{t-1} changes only when this state is visited again. So it takes a long time for this updated value to propagate back through the state space. To speed this process up, 'experience replay' was introduced. This involves replaying state transitions using observed data as if they were new observations.

Other problems that are overcome using batch RL are inefficiencies due to stochastic approximation of values and stability issues when using function approximation. Besides this the algorithm is reminiscent of the typical temporal difference learning algorithms as it uses these, only then more often by learning in batches.

4.2.3 Bayesian Reinforcement Learning

When determining the behavior of a user on his smartphone, a lot of uncertainty in state-transitions arises. Probably many of these transitions will be correct (e.g. traveling from work to home) but they will always remain uncertain. To deal with uncertainty, Bayesian networks can be used. "Since Bayesian learning meshes well with decision theory, Bayesian techniques are natural candidates to simultaneously learn about the environment while making decisions." [121]. As a sidenote it should be mentioned that uncertainty in state-transitions is not present when working with time for example. The fact that time progresses is deterministic and as this feature will be the only one used in this research, dealing with uncertainty is not yet a crucial feature. It could become so when for example location is used.

Multiple algorithms have been developed in this field. These can be categorized as value-function based or policy gradient algorithms. Some algorithms for each of these categories will be discussed. As these are more sophisticated methods, they can become more difficult to implement and fully understand. This research covers more than just machine learning which could make these algorithms unfeasible to use due to time constraints. They will be covered shortly nevertheless, possibly for future work.

Value-Function Based Algorithms

Value-function based algorithms search the space of value functions to find an optimal value (action-value) function.

An algorithm in this class that works with discrete state and action spaces is Bayesian Q-Learning (BQL). As the name hints, it is a Bayesian approach to the popular Q-learning algorithm. Here exploration and exploitation are balanced by explicitly maintaining a distribution over Q-values to help select actions. In its original form BQL algorithms can only be applied to MDPs with finite state and action spaces.

As an extension to BQL that can handle continuous state or action spaces, Gaussian Process Temporal Difference Learning (GPTD) was proposed. Where the sum of discounted rewards for a state-action pair is modeled by a Normal distribution in BQL, a Gaussian is used in GPTD. Furthermore using a Gaussian process, infinite state and actions spaces are accomodated using infinitely many Gaussians over the Q-value of a state-action pair.

Policy Gradient Algorithms

Policy gradient algorithms maintain a parameterized action-selection policy and update the policy parameters by moving them in the direction of an estimate of the gradient of a performance measure. They are proven to work well with partial observability. The largest problem in this field though is the high variance of the gradient estimates, mostly due to the reliance on Monte-Carlo (MC) techniques. As a Bayesian alternative to this, Bayesian Quadrature (BQ) was introduced which outperforms MC in terms of mean-squared error by orders of magnitude.

An algorithm in this domain is the Bayesian Policy Gradient (BPG) method. This method casts the problem of estimating the gradient of expected return as an integral evaluation problem, and then uses the BQ method. The algorithm starts with an initial set of policy parameters and updates these parameters in the direction of the posterior mean of the gradient of the expected return, as calculated by the BPG evaluation procedure.

4.2.4 Evolutionary Reinforcement Learning

Evolutionary algorithms use the process of natural selection to solve optimization problems. These algorithms can be used for discovering high-performing reinforcement-learning policies. [47, 124]

The evolutionary methods are evaluated to perform well with partial observability and continuous action spaces. Furthermore with the use of hybrid methods, the computation power needed is less than for temporal-difference methods. The only downside is that these techniques are mostly used in offline learning. There has been promising research in the online learning area, but there are still some critical challenges ahead for this. Since these techniques are not profound enough, they will be assumed as only available in offline learning.

4.2.5 Evaluation of Algorithms

A selection of reinforcement learning algorithms has been discussed and shortly explained. Since no model is available, all methods evaluated work without one and are thus model-free. A comparison chart was made in Table 4.1 with the criteria that were discussed before.

	online learning	**learning speed**	off-policy	supports partial observability	*deal with uncertainty*	*infinite state/action space*
Q-learning	√	+	√	X	X	X
SARSA	√	-	X	X	X	X
Actor-Critic	√	-	X	X	X	√
Batch RL	√	+	√ / X	X	X	√ / X
BQL	√	+	√	X	√	X
GPTD	√	+	√	X	√	√
BPG	√	+	X	√	√	
Evolutionary	X			√		√

Table 4.1: Comparison of RL algorithms

The evaluation was set up as such that all green fields are desirable options. Meaning that one can simply refer to the table and search for green fields. The yellow fields for batch reinforcement learning mean that these class of algorithms depend on the underlying algorithm used. When reading the table though, keep in mind the importance of a field. The fact that learning is done in an online fashion is crucial and also the learning speed is considered crucial in the mobile domain, therefore these fields are made bold. All of the other factors are desirable and thus negations here might be permissible via a workaround. The italic features are of least importance as they either are not applicable yet when only learning based on time (uncertainty) or they are very easily worked around. Some gaps exist where papers didn't state clearly whether this was or was not a feature of the algorithm.

From the table it can be concluded that 3 algorithms (SARSA, Actor-Critic and Evolutionary) do not satisfy the most important bold constraints. Furthermore one can see that the Bayesian algorithms (BQL, GPTD and BPG) have a few more green italic fields. These though are of very minor importance for this research. Besides that Bayesian Reinforcement algorithms are far more complex and used very little in recent literature yet. This was explained in section 4.2.3 on Bayesian Reinforcement Learning. As this research considers more than just machine learning and the advantage is only very minor, the more proven methods are deemed favorable.

This leaves only Q-learning and Batch RL. Batch RL though, only has a green field for off-policy if Q-learning is used underneath instead of Actor-Critic. So basically both algorithms will use Q-learning, where the only difference is the fact that Batch RL more often repeats samples gathered that Q-learning only processes once. Since Q-learning is the core of this and will be used anyway, this algorithm is chosen for this research. When deemed valuable it can easily be expanded to Batch RL by reentering old data entries in the algorithm. In future work the more complex but possibly slightly better bayesian algorithms can be considered.

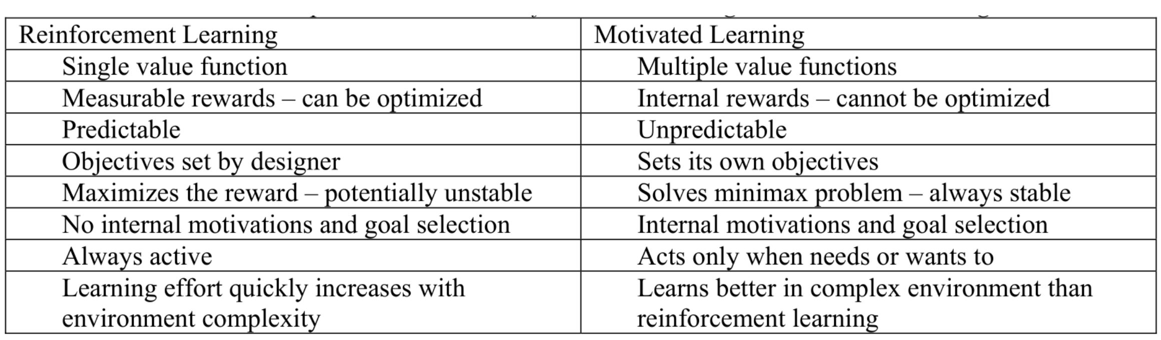
4.3 Alternative Learning Methods

Besides the regular machine learning techniques, alternative methods exist to let agents learn about their environment. This chapter will discuss Motivated Learning as an alternative to reinforcement learning. Then learning policies in BDI-agents will be discussed and finally a short comment on the transfer of learning experiences is made.

4.3.1 Motivated Learning

In reinforcement learning, the learning effort and computational cost increase significantly with the complexity of the environment. This turns optimal decision making intractable in these complex environments. Using a network of interdependent motivations, goals and values that the machine learns while interacting with the environment, a new learning strategy is described, Motivated Learning [111]. From Motivated learning, embodied intelligence is produced. An agent that uses motivated learning is called a Motivated Embodied Intelligence (MEI) agent. These agents resemble BDI agents a lot. The main significant difference between the two is that while the motivations of BDI agents are predetermined by the designer, MEI agents create their own motivations to act and learn how to implement their goals.

In 2013 the author performed a successful simulation in a 3D game with an MEI agent [112]. Furthermore a comparison chart with reinforcement learning is provided in Figure 4.2.



Reinforcement Learning	Motivated Learning
Single value function	Multiple value functions
Measurable rewards – can be optimized	Internal rewards – cannot be optimized
Predictable	Unpredictable
Objectives set by designer	Sets its own objectives
Maximizes the reward – potentially unstable	Solves minimax problem – always stable
No internal motivations and goal selection	Internal motivations and goal selection
Always active	Acts only when needs or wants to
Learning effort quickly increases with environment complexity	Learns better in complex environment than reinforcement learning

Figure 4.2: Comparison between reinforcement learning and motivated learning [111]

4.3.2 Learning Plans in BDI Agents

Learning in BDI agents often is about learning what plans are appropriate. When the environment changes, different plans might become more interesting and thus it is desired that the agent learns to use different plans. [48, 107].

Previous work [108] has combined decision trees with BDI agents to enable BDI agents to learn from past experience. This way the effectiveness of their plans can be evaluated and altered over time. This approach will probably not be useful in this research, since the problem will not contain a vast plan base from which an agent can learn different plans. The only type of plan an agent can choose is to launch an app at a certain time, with variables being what app and at what time.

4.3.3 Transfers in Reinforcement Learning

Since the system might have multiple predictors acting at the same time, they can share their results and experience with each other. Research has been done to these types of transfers in reinforcement learning [63]. The agents can share parameters they've learned that work well with each other for example. This has been done in community learning as discussed in related work. This research will not involve any of this, but future work could take it into account.

4.4 Cooperative Machine Learning

Besides predicting app launches based on various input types, this research also wants to combine multiple predictors and evaluate their performances in some kind of assessment agent. Since the performance of agents will likely alter with changes in the behavior of the user, these evaluations differ constantly. When an app launch is predicted by a predicting agent, the accuracy measure is known as soon as the user acts. The quality of such a prediction agent can then be learned using reinforcement learning. This learning via multiple agents can improve reinforcement learning [14].

In their paper on cooperative machine learning [90], the authors describe two types: team learning and concurrent learning. Team learning applies a single learner to search for behaviors for the entire team of agents. Concurrent learning uses multiple concurrent learning processes.

Within team learning, a distinction can be made between homogeneous and heterogeneous agents. Homogeneous means that all agents are assigned identical behaviors, while not being identical perse. Heterogeneous agents have a certain skillset and thus differ in behavior. Homogeneous is especially interesting when no specializations of agents are required, or when the search space is too large to use heterogeneous agents. Finally hybrid team learning provides a way to combine both.

In concurrent learning, typically each agent has its own unique learning process to modify its behavior. Concurrent learning may be preferable in those domains for which some decomposition is possible and helpful. The question then is whether this holds in the problem domain of this research.

Concurrent learning using homogeneous agents seems to be the best approach in this domain. As all agents eventually perform the same tasks, but all in a different way, homogeneous agents are needed. The following section will elaborate on the choice of concurrent learning. When using multiple learners, credit can be assigned to them according to their work done or performance reached. This will be discussed in the section on credit assignment. Whether this is needed is doubted as each predictor learns how well it performs on its own using reinforcement learning, but the principle is closely related and might be used. Furthermore, ensemble learning is discussed as an example of dividing the learning task.

4.4.1 Concurrent learning

Concurrent learning projects a large joint team search space onto separate smaller ones, thus reducing computational complexity. Furthermore, breaking the learning process into smaller chunks permits more flexibility in the use of computational resources to learn each process, because they may be learned independently of one another.

Most research in finding user patterns recently focuses on learning independent aspects of a user. For instance only keep the application launch sequence in mind like PREPP [92] or learn multiple aspects concurrently and combine them later [134]. This would give rise to the idea that the learning process can be decomposed.

Benefits of this technique are that mobile resources are sparse and can then be used more flexibly, disabling them when the mobile device is running out of battery. Furthermore the computational complexity would decrease, which again is preferable considering the limited resources of mobile devices.

Usually in concurrent learning, the problem of multiple learners occurs. This means that when multiple agents are acting in an environment, the fact that one of them learns can affect the others as well. This problem is avoided in the current domain since the environment will technically not be altered by the agents. The agents learn behavior of a user, but they do not enforce or change any of this behavior.

4.4.2 Credit Assignment

There exist various ways to assign credit to the different learners in concurrent learning. Global reward gives all learners equal reward, regardless of how they contributed. But in order to reward good learners and punish others, reward can best not be divided equally.

A possible way to do divide the credit is by using local reward. It assesses each agent solely on its individual behavior. This way it discourages weak agents because they will receive few reward. A downside is that agents will have no rational incentive to help others, which will result in the development of greedy behaviors. This doesn't have to be a problem in this research though, since each learner should be best in its own case and its not preferable for an agent to perform worse in order to help others. That is because the predictors do not depend on each other for their performance. In problems where this is the case, it could be preferable for agents to let some of their performance drop in order to improve that of others. Here this is not applicable.

It is argued that local reward increases homogeneity in teams, which suggests that the choice of credit assignment should depend on the desired degree of specialization. Since in this research the agents are basically required to do exactly the same (namely predict behavior, only in different ways) homogeneity is desirable and thus local reward is a good way to go.

Besides local reward, many other techniques have been developed. One other way to evaluate agents' performance is to evaluate how the team would have performed if the agent had never existed [127].

Assessing Agents

Besides dividing the credits well, the agents have to be assessed in order to determine their value. In [64] a way to assess which of multiple criteria influence the real user's assessment is presented. It proposes an Adaptive Multi-Agent System and compares it to other standard learning algorithms like Neural Networks and Support Vector Machines. A desired way to assess the quality of cooperating agents in the final result is described. If an agent (criteria) performs well, its weight goes up. When results turn out negative, the agents (criteria) that had most influence on the outcome are punished most by having their weights reduced.

Various studies have been done to different kind of comparison techniques [44, 62]. For instance, eigenbehaviors present a way to compare different behavior patterns between users [39]. Since they are vectors, they can easily be evaluated using euclidean functions. In this research, local reward can be used based on the error the agents made in their predictions. To possibly improve the system, other techniques as described here could be tried.

4.4.3 Ensemble Learning

In ensemble learning, multiple models are learned independently of each other and later combined to create a better hypothesis as opposed to a single model [76]. This is most common in supervised learning. Some sort of ensemble learning is done by Darwin Phones [74]. The paper is about classifying context on multiple devices and combining these by using model pooling (sharing models between devices) and let them perform collaborative inference. For this research it doesn't seem as interesting though, mainly because ensemble learning is often used for supervised learning. The application found in Darwin Phones differs in that it combines models from different devices. There do exist efforts though that apply ensemble learning in RL, mainly because of it's often proven improvement over learning with a single model.

Ensemble Learning with Reinforcement Learning

Ensemble learning and reinforcement learning have been combined in the past. The general idea is to let several homogeneous or heterogeneous reinforcement learning algorithms run at the same time and combine their efforts. In [68] it is shown that this tactic can be very effective and result in rewards that are better than every separate RL algorithm.

Research has been done where five different RL algorithms (Q-learning, Sarsa, Actor-Critic, QV-learning and ACLA) have been combined using ensembles in four different ways [126]. Besides this one there exist several others that have succesfully applied this approach [41, 93]. This research thus has good reasons to believe the combining of different prediction agents will have a positive effect.

4.5 Libraries

To use machine learning, one could of course implement the algorithms themselves, but this will not always be necessary as there exist numerous libraries that implement these learning techniques. To ensure cross platform operability, most researchers and with it most of these techniques are implemented in Java. This section will briefly discuss three of these, WEKA, RL-Glue and Azure ML.

4.5.1 WEKA

WEKA is a machine learning API, written in Java. The manual for WEKA 3-7-8, which is a developers version, can be found in [22]. Furthermore some experiences of WEKA by the authors have been listed in [23]. From the libraries discussed it is one of the oldest and contains most algorithms for all sorts of machine learning techniques.

4.5.2 RL-Glue

RL-Glue [116] is a library with Reinforcement learning methods for internal use in C# and Java. This means that the library can be used in these programming languages directly. Nevertheless it is available via external use for all platforms.

4.5.3 Azure Machine Learning

Azure is the cloud solution of Microsoft. With Azure, an application resides in the cloud and can be accessed from anywhere on the internet. Inside Azure recently the possibility to run machine learning has been provided [70]. Unfortunately though, Azure provides no reinforcement learning methods at the moment. Besides that, for research purposes a local machine learning library seems preferred for quick setup and easy testing. Nevertheless by offloading this to the cloud some computational work could be relieved from the mobile device.

4.6 Conclusion

In this chapter many approaches to learn artificially have been discussed. Firstly it was concluded that reinforcement learning is the type of learning that is applicable in this domain. These algorithms have been researched for a long time and thus are proven to work well. Nevertheless some alternatives like Motivated Learning might be applied to compare results eventually. From the reinforcement learning algorithms, Q-learning was chosen to provide artificial learning. Various other algorithms that are supposed to perform better though were treated as well. As these techniques are more sophisticated, difficult to implement and aren't sufficiently backed up by results, Q-learning will be used at first. Again in future work or if time allows, using other techniques might be considered. The reason for not delving further into this, is that the research depends on a lot more factors (the multi-agent system for instance) than reinforcement learning alone. Because of this the decision was made to start off with a proven technique for reinforcement learning, so to avoid unnecessary uncertainty to the solution.

As the predictors each are reinforced individually, no cooperation techniques are required initially. When gathering more of these predictors though, the techniques discussed here can become important in assessing what agents to believe. When this is done, local reward will be the first choice, in order to have homogeneous agents. As Q-learning is present in the RL-Glue library, this library will be used.

Chapter 5

Mobile Platforms

Now that every technique has been chosen, a platform to implement the solution on has to be picked. Various platforms exist for mobile devices, but by far the most used are Android, iOS and Windows Phone. Each of these platforms have their own limitations and their own benefits.

The third subquestion posed in the introduction will be treated here:

SQ3: Which mobile platform should be used?

It should be mentioned that the mobile platform is of particular importance when deploying the system to a device. This research will not include a deployment as it will only use simulations to verify the techniques used. The question is mostly answered for Avanade. Since they focus on Microsoft technology, the question rose on whether the Windows Phone platform is suitable. Furthermore it is deemed useful to know which platform and development environment is most interesting, particularly for any future work.

First the different platforms are compared with the abilities needed kept in mind. Then some development environments are introduced and finally a conclusion is drawn about what will be used.

5.1 Comparison

The three different platforms mentioned before all have their own perks. For this research it is particularly important that at least features of the phone can be monitored, apps can be launched from another app and Java is supported. This support is important since all agent frameworks found in section 3.2 work with Java at the moment. The assessment criteria used along with the results can be found in table 5.1.

Java is only supported by Android[1] which gives it a very important advantage. The parts of the agent framework that are needed could be rewritten in a different language, but it should be avoided if possible. Another important issue arrives at the launching of apps from within

[1]`http://en.wikipedia.org/wiki/Comparison_of_mobile_operating_systems`

	Java	App launching	Monitor system
Android	√	√	√
iOS	X	√ / X	√ / X
Windows Phone	X	√ / X	√ / X

Table 5.1: Comparison of mobile platforms

another app. In iOS[2] and Windows Phone [73] it seems to only be possible to start some default apps like Mail or SMS from within other apps, but not any kind of app. Android phones are able to perform this though[3]. Besides that, in iOS[4] and Windows Phone the system can monitor all sensors like GPS or the Accelerometer, but is unable to monitor which apps are opened. These problems are due to the nature of apps being separated from each other by design. Android is a bit more open and is able to monitor this[5] [6].

5.2 Development Environment

There are multiple development environments available for mobile devices. As Android is the chosen platform, Eclipse is the most likely choice. That is because Eclipse provides development tools and support for Android development. Also the Tropos design method works with Eclipse as seen in section 3.3.

Besides Eclipse though there is an interesting other option, Xamarin. Xamarin is a development tool that provides 100% coverage of the API's of iOS, Android and Windows Phone and makes sure an app can be developed for all 3 platforms using only .NET. It will shortly be treated below.

5.2.1 Xamarin

There have been studies in the past regarding which mobile OS is easiest to develop in [45]. These discussions though are somewhat in the past when using Xamarin. Xamarin is a development tool that makes it possible to develop mobile applications in the .NET framework. These applications can easily be ported to all 3 mobile platforms, thus removing the need to code an app 3 times.

Java can be integrated in Xamarin so that existing java libraries can be used when coding in the .NET environment of Xamarin. For this Java bindings can best be used [132]. When this is done, the application can only run on Android though, because it still needs a Java Virtual Machine to perform the Java libraries on. Xamarin thus doesn't enable us to use iOS or Windows Phone, but the application could be adjusted more easily to work with either one of the platforms by replacing or rewriting the java libraries used.

[2]http://stackoverflow.com/questions/419119/
[3]http://stackoverflow.com/questions/3872063/
[4]http://stackoverflow.com/questions/19452696/
[5]http://stackoverflow.com/questions/11346557/
[6]http://stackoverflow.com/questions/3290936/

5.3 Conclusion

This chapter discusses Android, iOS and Windows Phone as possible platforms to eventually deploy the system to. Mostly because of the need for Java considering the agent framework used, Android is shown to be the best choice for a mobile operating system. In order to develop the code, Xamarin and Eclipse were proposed. Xamarin promises some interesting features by providing the possibility to create apps for all mobile platforms while coding them only once. Unfortunately the binding of Java libraries turned out to be too time consuming. Because of this, Eclipse was used to develop and test the system with.

Chapter 6

Implementation

Now that all of the literature is discussed and an idea is formed on how to fix the problem at hand, the implementation of this idea comes next. As was concluded before, multi agent systems can provide a flexible way of designing programs. Tasks can easily be divided over active components that execute their tasks in parallel. Such a system can be realized in numerous different ways, which is why a design methodology should be used. As was concluded in section 3.3, Tropos will be used in the following sections to design the MAS and further specify the tasks of the system. The actual code resulting from the described design can be found via appendix D.

Firstly the overall design of the MAS will be described, using Tropos as a design methodology, in section 6.1. Then each identified agent will be further specified in section 6.2. This concludes the basis of the system. Nevertheless, several expansions are possible and these will be discussed in section 6.3.

6.1 Multi Agent System Design

Tropos [32] is a software engineering methodology that can be used to design software particularly existing of agents. It provides a guideline to develop agent systems. There exist some tools that can be used in the design process of Tropos. TAOM4E [106] is one of these, which was used because of its integration with Eclipse and Jadex. The tool provides a way to design Tropos models in Eclipse, the same environment in which the application will be developed. Furthermore, it supports exporting the model to Jadex code. This shows that the way agents are modeled in Tropos and Jadex coincide, which is an important benefit of Tropos. All diagrams used in this section were created using TAOM4E.

This section will first walk through the 2 requirement phases that Tropos offers, the early and late requirements. Once these have been determined, the architectural design will be made. The specifications of each agent identified here will be deferred to section 6.2, which essentially covers the detailed design phase of Tropos.

6.1.1 Early Requirements

The early requirements phase of Tropos concerns the understanding of the organizational context within which the system-to-be will eventually function. To understand this context, domain knowledge is acquired from experts. With this knowledge, actors that need to interact with the system-to-be are modeled along with their goals. These goals will finally be made more specific.

Domain Knowledge Acquisition

The proposed ways of acquiring domain knowledge involve talking to stakeholders and analyzing documents. Using these techniques, information on required and desired functionality of the system-to-be can be obtained. The present system though, will not provide any practical functionality that accomplishes actual goals of other actors. It will simply improve usability of a smartphone. Because of this the organizational context of the system is very simple and thus does not require elaborate domain knowledge acquisition. Later on, the system will be further specified. For these decisions the related work section of this research (Chapter 2) can be regarded as domain knowledge. Furthermore, interviews with employees at Avanade were conducted to extract useful applications of the system.

Actor Modeling

This phase concerns the modeling of desires, needs and preferences for every stakeholder involved. Since the only stakeholder of the system-to-be is the user of a smartphone, the resulting model is fairly simple and doesn't include any relations. The resulting diagram can be seen in Figure 6.1 and is discussed below.

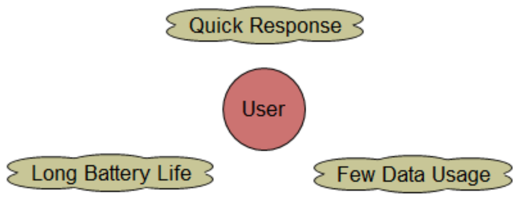


Figure 6.1: Early Requirements: Actor Diagram

Tropos differentiates between 2 types of goals. Functions that the user needs from the system-to-be are called 'hard goals', while functions that are desired or preferred are called 'soft goals'. They are depicted using round circles and cloud-like circles respectively.

A mobile phone is assumed to have only a single user. This user will use the phone to open applications that provide the user with the desired functionality. The system-to-be will not provide any of the functionality like opening apps on the foreground, but only properties that are desired. The user will want a quick response from his mobile phone. Furthermore the battery of the phone should last as long as possible and the system-to-be should use as few data as possible. These 3 goals form the soft goals that can be identified.

Goal Modeling

For each goal identified in the previous step, a decomposition into sub-goals can be made. The only way a user himself can alter any of the soft goals would be by upgrading his phone to a newer model. This would probably reduce the response time and hopefully prolong the battery life. Using fewer data could only be influenced by using the phone less when on paid connections, but this should not be considered a goal of the user. For our system-to-be the goals in the early requirements cannot be decomposed, but to illustrate how this step works, the upgrade of a phone is included in the model in Figure 6.2.

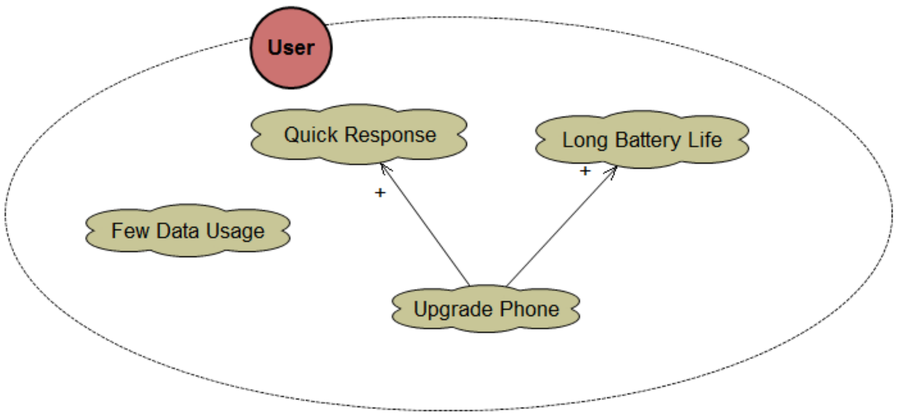


Figure 6.2: Early Requirements: Goal Diagram

6.1.2 Late Requirements

The Late Requirements Phase is concerned with the definition of functional and non-functional requirements for the system-to-be. The new system will be treated as another actor (or a small number of actors) who are dependers or dependees in dependencies that relate them to external actors. The top goals assigned to the system will then be refined and finally the capabilities of the system-to-be are identified and modeled as plans. Since the goals of minimizing battery usage and data usage are outside the scope of the project, these will not be specified further for now.

Actor Modeling

The system-to-be will first be introduced as an actor. The goals of the stakeholders will be assigned to the new actor by establishing goal dependency links from stakeholder actors to the system actor. Figure 6.3 shows the resulting diagram.

All of the soft goals identified in the previous phase will depend on the system for their completion. Furthermore the system will be given a hard goal that enables it to fulfill some of the soft goals. This goal will be to prefetch applications before use, so that opening them will require a smaller load time.

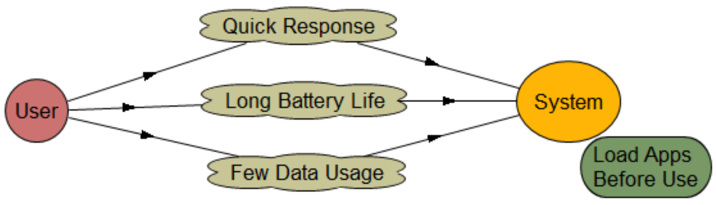


Figure 6.3: Late Requirements: Actor Diagram

Goal Modeling

As done in the previous phase, the goals specified can now be refined. This time, goals are analyzed from the system actor's perspective. Figure 6.4 shows the decomposition of the top-level goal into lower level ones.

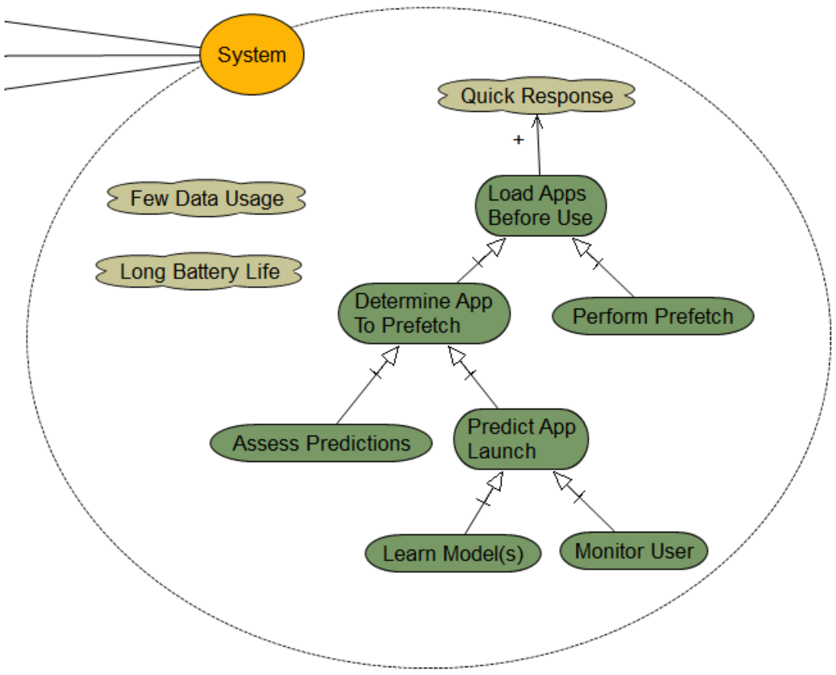


Figure 6.4: Late Requirements: Goal Diagram

The main goal of the system will be to prefetch data before the user accesses it, this will improve the load time of applications and thus provide a quicker response of the phone. This goal can be decomposed in one that determines what application will be prefetched and one that performs it. The goal that determines the app to prefetch can be further decomposed into a goal that predicts what app will be launched next and one that assesses the quality of this prediction. The reason to have such an assessment is to be able to combine multiple predictors later on in the modeling process. Having multiple of these predictors can greatly enhance performance of the system. Finally the prediction of an app launch can be decomposed into a goal to learn a model for these launches and to monitor the user in order to feed this model.

Plan Modeling

The last phase of the late requirements turns the goals of each actor into concrete plans. Every plan that fulfills a leaf-level goal is modeled by a means-end relation to that goal. These plans are straightforward for each leaf-level goal in our example and are depicted in Figure 6.5.

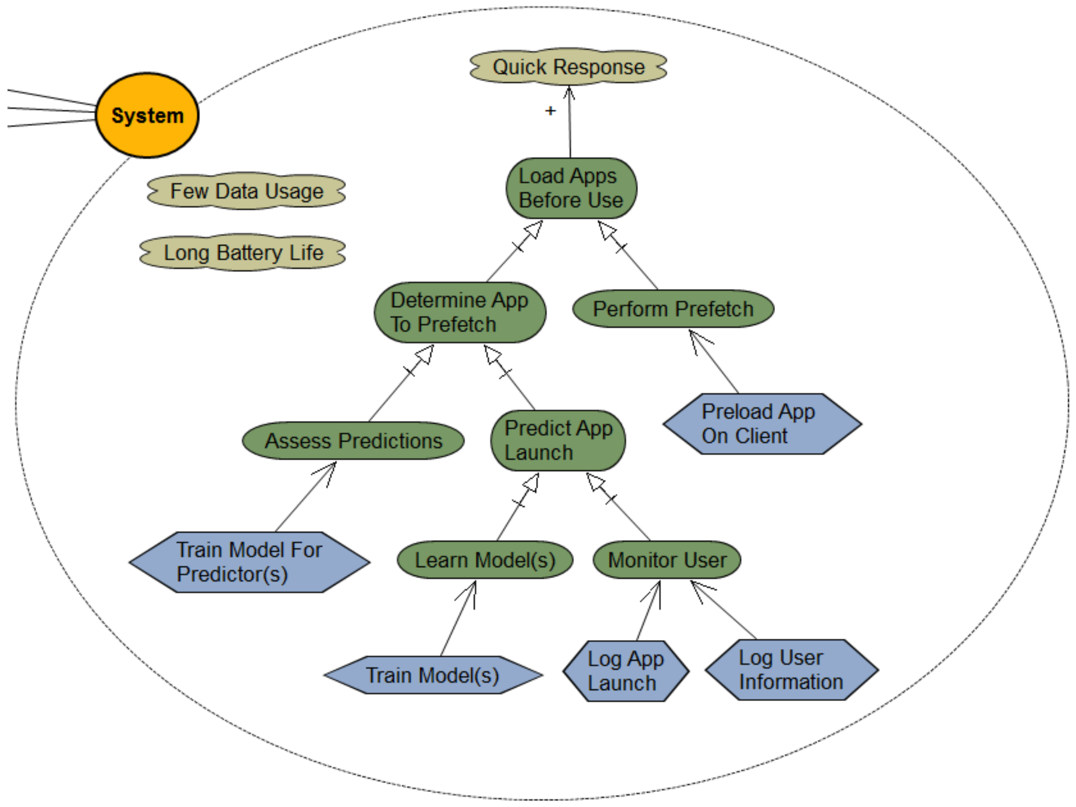


Figure 6.5: Late Requirements: Goal Diagram (extended with plans)

6.1.3 Architectural Design

Now that all requirements for the system are modeled, the actual system architecture can be made. This phase consists specifically of designing the multi-agent system structure and further specifying the capabilities for each individual agent. First the various agents that together form the system will be identified. Then the goals specified earlier will be delegated to these new agents. Finally for every agent the goals and capabilities are further specified.

Identify Agents

Considering the goals that were identified earlier, the system will be broken down into smaller subsystems (agents) that are all responsible for part of the job. When considering the process of prefetching apps, a first step to be made is the gathering of information. This information includes the times that a user actually launches an app and possibly information on the location, duration, activity, etc.. This task can be decoupled from the total system in the form of an Input agent. As we will have multiple predictors, this will prevent the system from gathering the same information multiple times.

The input gathered by this agent is needed to predict what applications to prefetch next. For these predictions, models need to be trained and consulted when applicable. Since it might be interesting to train multiple models based on different classifiers, multiple of these predictors can be allowed. This introduces our next agent, the Prediction agent.

Now that there are multiple predictors making predictions in parallel, an agent is needed that brings this information together. This agent can learn the performance metrics of each individual predictor and decide who to believe based on this. The task of gathering predictions and assessing them can thus be separated in the form of an Assessment agent.

Finally the conclusions of this agent consist of what application to prefetch. This prefetch of course still has to be carried out, for which a Launcher agent can be used.

Delegate Goals

After identifying which agents are present in the system, the previously modeled goals will be divided. Figure 6.6 shows the goals assigned to the agents and the dependencies between them.

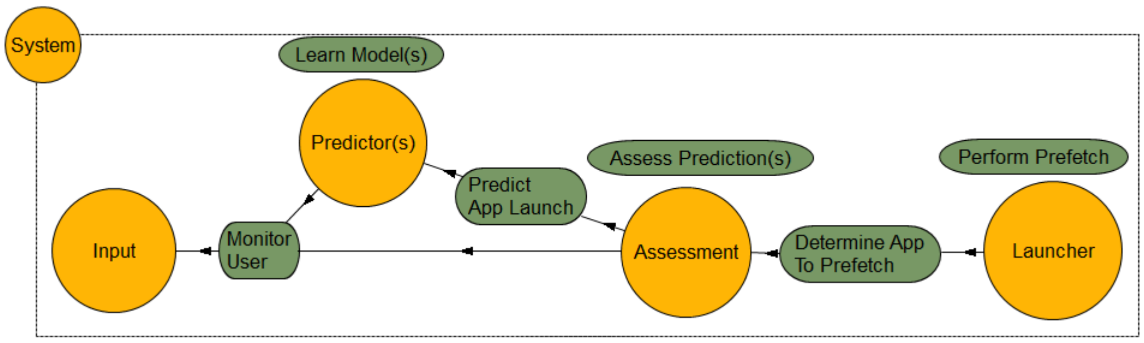


Figure 6.6: Architectural Design: Delegations

In short this system works as follows. The Launcher agent performs a prefetch of apps determined by the Assessment agent. This Assessment agent makes this choice based on predictions gathered from Predictor agents and the assessment it learns about these predictions using app launches logged by the Input agent. The Predictor agents are able to make these predictions based on the information they receive from the Input agent and the machine learning models they learn on this information. Finally the Input agent gathers the information needed by the Predictors and Assessor from sensors in the smartphone.

Goal and Capability Modeling

As has been done in the requirements phases, the goals for each individual agent can now be specified further along with their capabilities. The result of this phase is depicted in Figure 6.7.

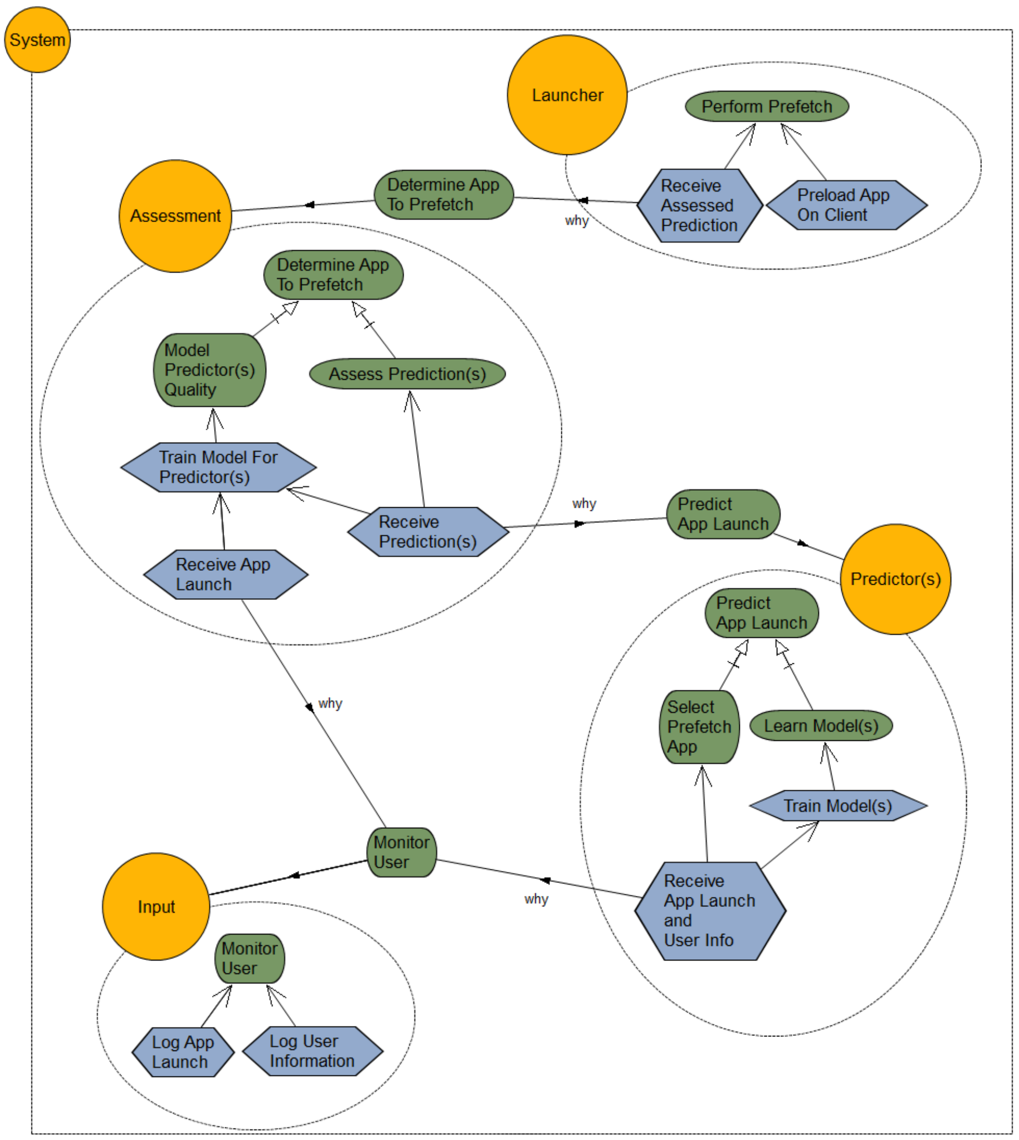


Figure 6.7: Architectural Design: Complete

After the architectural phase comes the detailed design phase and implementation and testing phase. This elaborate design of individual agents will be discussed in the next section on agent design, section 6.2.

6.2 Agent Design

Now that the various agents in the Multi Agent System are identified, each agent has to be further specified. These specifications will be given for the basic system, so without any expansions done yet. This will provide the framework upon which various improvements are possible. These improvements will be discussed in a separate section, called Expansions (section 6.3).

For every agent identified in the previous section, a description of the way it works and the messages it sends will be given. This includes the possible machine learning models that were used and the way they work within each separate agent.

6.2.1 Input Agent

The task of an input agent is to retrieve all information about the user (apps launched, location, time, etc.) that is requested by the prediction agents. This means that capabilities of this agent involve polling the user location, logging his application launches and everything else that can be determined about the user.

The input agent works on demand, so it reacts to questions from prediction agents. Initially, it will not do anything but wait for messages from these prediction agents. A prediction agent can now send a message to the input agent that specifies what information it would like to receive at what interval. Say a prediction agent wants to know the users location every 5 minutes, then the input agent will start a process that polls this location for the requested interval and immediately sends the information to the predictor. Besides this, the input agent will send information about an application launch to every prediction and assessment agent for every launch performed by the user.

All the applications that are launched are given a unique ID so that the other agents don't have to bother checking what input belongs to what exact application. In the end the launcher agent can use the mapping between IDs and applications to launch the appropriate application. The IDs start at number 1, reserving 0 for not launching any application.

6.2.2 Prediction Agent(s)

A prediction agent's task is to predict what application will be launched next by the user based on information about past behavior. These predictions can be found using a large range of different techniques. As concluded before (section 4), reinforcement learning seems to be the most promising form of learning for this problem and because of this it will be the current focus for the prediction agents.

The prediction agent takes an active role in determining its predictions. First it will announce to the input agent what data it will need. The agent will then receive the requested information on the desired intervals and can learn its machine learning model using this. This model will be used to predict what is the most likely application to be launched next, along with a quality of this prediction. If the quality of this result from the model is above a certain threshold, the agent will send this prediction to the assessment agent.

Reinforcement learning can be applied in a lot of different ways. This research though will focus solely on learning patterns based on time of day. This allows to evaluate the system, but it should be kept in mind that a lot more options are possible. They are left for future work and described in section 6.3.

Learning Based On Time

A prediction agent learning solely based on time is considered. This agent uses the Q-learning [125] algorithm for reinforcement learning, thus learning the quality of a state-action pair. As a state this agent uses time and the possible actions are the different applications that can be launched. Only one application can be launched in an action, thus limiting the prefetch to one application per timestep used by the agent. Furthermore it should be noted that action number 0 is reserved for not prefetching any application.

There are 2 different messages that can be received. For each, the actions that are performed after will be treated. Firstly the agent can receive a message notifying him that the user launched an application (Algorithm 1). And he can receive one that notifies him of a time update (Algorithm 2).

Algorithm 1 Receive an application launch

```
1: Update launched applications                ▷ For later use to determine reward
```

Algorithm 2 Receive a time update

```
 1: Remove outdated prefetches                  ▷ When their lifetime is expired
 2: for Every possible prefetch do
 3:     Calculate reward                        ▷ Proportional to number of correct prefetches
 4:     Update old state/action value           ▷ Using parameters for learning
 5: end for
 6: Set current state of model                  ▷ As specified by time update
 7: Select next prefetch                        ▷ select action with highest state/action value
 8: if value > quality threshold then
 9:     Send prediction to assessment agent
10: end if
```

In these algorithms, several parameters can be identified that alter the performance of the agent. For each of these algorithms an optimal value has to be estimated:

- Learning time span:
  The agent can learn within different time spans. Say for instance an agent learns per day. This means that 14:00 on Monday is the same time to him as 14:00 on Tuesday. If he were to learn per week, 14:00 on Monday would be different from 14:00 on Tuesday, but again the same as 14:00 on Monday next week. In case of per day learning, weekends can be treated as a special case as they tend to have different patterns.

- Polling interval:
  With this parameter the interval with which the agent requests state updates from the input agent is meant. This influences the precision with which the agent learns. For example, a polling interval of 5 minutes allows the agent to update its state every 5 minutes and to possibly prefetch an application every 5 minutes. If this interval would be too small,

say 10 seconds, the agent would distinguish between app launches that are more than 10 seconds apart. As a deviation of 10 seconds from a pattern is highly likely to occur with human users, this will result in a failure to learn such a pattern. Too large a polling interval, say 1 hour, is undesirable as well. This causes applications to be predicted only once an hour and will thus reduce the actuality of a prefetch.

- Quality threshold:
  Determines when a prediction done by the model is good enough to be send to the assessment agent.

- Lifetime prefetch:
  This parameter determines how long a prefetch will prevail and thus is deemed relevant. A fast changing application like a news app will have a short lifetime, as its retrieved data is outdated quickly.

- Q-learning parameters:

  – Reward:
    There are several ways to determine the reward. One way is to only give positive reward when a prefetch was correct. It is also possible though to reduce reward when an incorrect prefetch was done, or when an application launch was not predicted.

  – Discount factor:
    The Q-learning algorithm works with a discount factor that determines the influence of future states. The higher this value, the more influence the next state will have on the state/action value.

  – Learn rate:
    This parameter is used to determine the learning speed. The higher this value, the more influence immediate reward and the value of the next state have. When 0 for instance, nothing will be learned.

6.2.3 Assessment Agent

The task of the assessment agent is to assess the quality of each prediction agent. Using this quality it decides what prefetches to do based on the predictions of the agents. This assessment can be done in numerous ways using supervised learning. As only the basis is described here, a simple statistic is used.

The agent will receive messages from both the input agent and the various prediction agents. Using this it will calculate two statistics: The percentage of correct prefetches out of the total number of prefetches (precision) and the percentage of correct prefetches out of the total number of application launches (recall). Everytime a prediction comes in from a prediction agent, the assessment agent decides whether to send this to the launcher agent or not based on these statistics.

There are again 2 different messages that can be received. For each of these the actions performed after will be specified. Firstly a prediction can be received from a prediction agent

(Algorithm 3) and secondly an application launch can be received form the input agent (Algorithm 4).

Algorithm 3 Receive a prediction
```
1: Add prediction to agent's list of predictions
2: if Quality of agent > quality threshold then        ▷ Quality measure decides on quality
3:     if Protocol of consecutive predictions then     ▷ Protocol decides on same predictions
4:         Send prediction to launcher agent
5:     end if
6: end if
```

Algorithm 4 Receive an application launch
```
1: for Every agent known do                 ▷ Known means: a prediction was received in past
2:     Remove outdated predictions                      ▷ When their lifetime is expired
3:     if application was predicted then
4:         Raise number of correct predictions
5:     end if
6: end for
```

The assessment agent also has parameters that can alter the way it works:

- Quality measure:
  This measure specifies the quality an agent should have before its predictions will be send to the launcher agent. Whenever an agent's quality is below a certain threshold, its predictions are too inaccurate. This measure can be calculated using the precision or the recall statistics of the agent. Also, a weighted function between these measures can be used.
- Protocol consecutive predictions:
  Whenever a prediction is done by a prediction agent while the same prediction was already done by the assessment agent, a choice has to be made. The new prediction can either be carried out or ignored.
- Lifetime prefetch:
  This parameter has the same meaning as the one described at the prediction agent. Probably it's best to keep these the same as this should be an attribute for the application that is prefetched.

6.2.4 Launcher Agent

The launcher agent's purpose is to launch the applications that are predicted by the rest of the network. It receives predictions from the assessment agent and makes sure the appropriate prefetch function for this application is initiated.

6.3 Expansions

The basis of the system as designed in the previous sections allows for a lot of expansions. The multi agent nature ensures a lot of flexibility for the system. Most agents can be expanded with varying levels of intelligence. The first and most basic expansion of the system would be to let prediction agents learn on increasingly different facets. These might include location or even calendar events belonging to the user.

Besides this it is possible to identify far more improvements on the system. These will be discussed in the following sections. Firstly functionality that can be added to the overall system will be discussed. Then every separate agent will be mentioned with possible extensions within their own behavior.

6.3.1 Overall System

There exist various ways of altering the overall system in order for it to predict more accurately. The ways discussed in this section include learning the lifetime of applications, community learning and the use of 2 new agent types: context and pattern matching. Finally a concept is explained that allows developers to use this system.

Lifetime of Applications

As of now, the lifetime of an application is a preset number, the same for each application. This lifetime determines the time a prefetch of an application is useful. When it expires before an actual launch is done, the prefetch will be considered useless. A news app for instance will have a short lifetime as a lot of new data will arrive in a short period of time.

A possible expansion is to let the system learn the lifetime of each application instead of setting it manually. The system could learn the amount of data that still needs to be gathered in between the time the app was prefetched and it was opened. The larger this amount, the shorter the lifetime of an app would become.

Community Learning

As previous research has concluded, many patterns of users are the same within various communities. It would thus be interesting to let agents of a certain user, learn from agents of other users. Applications or web pages that are often visited at a certain location for example can then be shared, this way a community can learn certain patterns instead of users themselves.

A related field of this technology is distributed sensing [94]. This technique allows systems to communicate their sensory data, so that they do not have to gather this all individually. In our system, the input agent could learn the current location from other input agents in the neighborhood.

New Agent: Context

In order to improve the system, a context agent is proposed. This agent could infer context that can be used by the prediction agents. There has been done a lot of related work into extracting user traits from sensory data on mobile phones. An example of how this data can be used is to determine whether someone is walking or driving by using the accelerometer. This information might possibly be used to determine the chance that a user will open an application.

This context agent could be used for far more purposes though. Patterns could be learned for the location of the user. This way the location of a user can be predicted whenever the input agent decides not to monitor this because of battery issues. Predicting this location can also be done on the server, this way offloading the computation for machine learning. Furthermore, patterns in charging behavior or wifi connection can be learned. This can help in prefetching ahead. Possibly some data can be prefetched earlier when a wifi connection is expected to be lost in order to save money on data charges.

Furthermore this agent could check when irregular patterns occur, like in the event of a holiday or weekend. It is likely that these days possess different patterns and thus for these days agents should stop their learning process.

New Agent: Pattern Matching

Another agent could be added to the system in order to improve its performance. This pattern matching agent can be used to match patterns in behavior of different apps. This way apps can be categorized based on their usage patterns. Whenever a new application is installed, the system could quickly learn its pattern by matching it with another application. This feature can turn out very valuable as users tend to install new apps quite often. Besides letting new applications find their pattern quickly, these apps in the same category might learn from each other as they have similar patterns.

Platform for Developers

The flexibility of the system gives rise to the opportunity for developers to create their own prefetching techniques. By providing an easy way for developers to create prediction agents for their own applications, machine learning can be added more easily. This idea is closely related to the framework Reflection [65]. As the system assesses the quality of each prediction agent, the created agent can automatically be disregarded if it doesn't function.

6.3.2 Input Agent

The input agent is responsible for data gathering. This logging of the system will use some battery life and thus is not always desired. The goal of not using too much of the battery could be added to the input agent. This way the agent will have conflicting goals and needs to utilize the true BDI potential in order to choose between these goals. Prediction agents could be ranked based on battery usage and whenever needed the most demanding ones can be shut down. In order to determine when to start shutting down agents, charging patterns can be used as was proposed in the addition of a context agent.

6.3.3 Prediction Agent(s)

The prediction agent forms the most important part of the system, as it generates the actual predictions that are needed to prefetch applications. Because of this these agents are involved in most improvements to the system. As said before the first thing to mention is that these agents can be used to learn using different sensory input data, from location to calendar events and many more. In addition to using different sensor data, the resolution with which this data is gathered will greatly alter performance. An agent can learn per day, meaning 10:00 o'clock results in the same state every day, or per week, meaning 10:00 o'clock is only the same state as 10:00 o'clock on the same day in another week.

Whilst learning, the agents use various parameters that determine their success. These parameters were discussed in the previous section. As of now, they have to be set prior to starting the agent, but these values could possibly be learned as well. The assessment agent knows the score of each prediction agent. It could feed this back to each of these agents which allows the predictors to adjust their parameters based on this. For this goals can be used within the prediction agents to keep performance at a certain level. To maintain these goals, plans can be adopted that alter parameters and report on the effect of this.

The agents could communicate with each other about good performing parameters. This idea was shown in community learning, where agents of different users communicate, but might also be useful for agents belonging to the same user. Possibly the assessment agent could notify agents that are performing bad and tell them to retrieve some parameters from other well performing agents.

6.3.4 Assessment Agent

The assessment agent determines the quality of each prediction agent and determines whether to believe it or not based on that. A first improvement of this model would be to combine the predictions of the different predictors and determine which one to prefetch based on their combined quality. This way 2 prediction agents with a low quality can complement their predictions and thus reach the threshold of the assessment agent.

When combining these predictions, supervised learning can be used. There exist various techniques in this domain. Some will probably prove useful in determining the quality of each agent and in combining the predictions of different agents.

Finally the assessment agent could take an active role in altering the predicting agents. It could kill agents that have been performing very bad over a certain period of time and even start new agents that are similar to good performing ones.

6.3.5 Launcher Agent

The launcher agent right now is only responsible for the actual prefetch of an application. Its functionality could be expanded though by letting it consider data usage and battery life. When one of these gets critical, it might prove better not to perform any prefetches. The user will rather have a working phone till the end of the day than a faster application launch upon use.

Another improvement on this agent would be to keep track of the times that are saved when using the prefetch capability. The launcher agent could keep track of the improvements obtained whenever a prefetch was successful. Prefetches that have practically no influence in loading times could be dismissed.

Chapter 7

Test Setup

The system as it was proposed in chapter 6 now has to be tested on its performance. This chapter will describe everything there is to know about the setup for testing.

The fourth subquestion posed in the introduction will be treated here:

SQ4: Which way can the system be tested?

In order to assess the system, a setup is needed that supports gathering the needed results. This section describes how user data was gathered, what adjustments had to be done to the original implementation and finally the performance measures used and how they will be measured.

7.1 Gathering User Data

For testing purposes, various alternatives were considered. One way the system can be tested, is by deploying it to a mobile device and conducting a real life user study. This was not feasible unfortunately due to the timespan available for this research. Besides this, using simulated data makes gathering statistics more easy and allows for an easier investigation into the possibilities. Simulating was thus the method chosen.

The data needed to run these simulations has to be retrieved somehow. This sensor data can be gathered by using monitoring apps or by using existing data sets. Both options will shortly be treated below.

7.1.1 Monitoring Sensor Data

There exist many applications that are capable of monitoring a smartphone. Benefits of this is that gathering sensor data can be tailored and log exactly the information that is required. There exist some frameworks that provide this functionality out of the box, without the need to implement every sensor log manually. One of these is called SCDF and is described in [7]. With SCDF one can choose what sensors to log and export it to an XML file. There is even a possibility to let the app send the statistics to a server periodically.

Funf [43] is another but more advanced application that provides the logging of sensor data [5]. Using Funf, one can specify what sensors to log, at what interval rate and for how long if that is applicable. Furthermore results can be send periodically to dropbox and can be exported to an SQL database file from there. The creation of the application that logs the sensors can be done via a web interface where one can easily specify the loggers needed and send the automatically created application to collaborating people. The app automatically sends the gathered statistics to the specified dropbox folder, where the developer can retrieve them from. A side note is that this application only works up until Android 4.4. Since Android 5.0 new functions were made to retrieve sensor data and the old ones that Funf still uses were deprecated.

7.1.2 Existing Data Sets

In order to simulate and test the predictions offline, existing data sets can be used. These are mostly limited though to information on location and application launch times and do not include much more sensor information. As the implementation without extensions only considers the app launch times, most of these will be sufficient for now. Livelab, AppSensor and Reality Mining are datasets that contain this information and will shortly be discussed

Livelab [103] is a dataset that has been developed as a way to retrieve usage data from iOS devices, specifically the iPhone 3GS. Since the iOS platform is very closed, jailbreaking (hacking) the phone was necessary to gain access to the various sensors on the phone. The application logs most of the sensors available on the phone [98].

AppSensor [18] is an event logger for Android and with it, data was collected from over 4000 users. The authors only looked at the location, time and sequence of apps launched.

Furthermore, researchers at MIT gathered a large dataset of user data. This project was called Reality Mining [75]. It has been described in [38]. As the data collected is from 2004, this dataset is the oldest of the three. This doesn't mean it can't be used, but the more recent the better as this reflects usage nowadays best.

7.1.3 Chosen Dataset

The Livelab data set has been chosen as this project contains most sensory data and was collected most recently. It contains a lot of data gathered from 34 different users from February 19th 2010 to April 25th 2011. These users were all affiliated with Rice University in Texas, USA. Due to time constraints, monitoring phones would result in significantly smaller datasets. The benefit it gives of having more recent data representative for phone usage nowadays, is deemed of less importance. Assuming phone usage, and with it patterns in this usage, has only grown since 2010, the program can be assessed on this dataset. Also, PREPP uses the same set of data, which allows for good comparisons.

The 34 users of the Livelab dataset along with some statistics on their application usages are shown in appendix A.

7.2 Adjustments to Implementation

In order for the simulation to be run, some minor adjustments have to be done to the implementation. First of all the simulation has to be able to run on maximum speed. As time is simulated, it would thus be ideal if the agents let each other know when they are done and the process can continue. How this process works will be described here.

Firstly the input agent will send information about the environment and about application launches in batches. The interval of these batches will be determined by the smallest interval a prediction agent uses to predict. Say there is an agent that learns and tries to prefetch every 5 minutes, and one that does so every 10 minutes. The input agent will then send both agents the required information every 5 minutes. The second prediction agent will ensure itself that it acts as if it received this information every 10 minutes. Whenever the prediction agents and the assessment agent are done, they send confirmations to the input agent so that it knows it can continue sending the next batch. This ensures a speedy simulation. In real life this would not form a problem as learning would happen at sufficiently large intervals for the algorithm to compute everything needed.

Furthermore the system is tuned a little bit by excluding weekends from learning. Whenever the system detects the simulated time falls in a weekend, it will stop learning and thus stop updating its state-action values. As people tend to have different schedules in weekends, the destructive results of this when included are prevented. Also, the application called 'SpringBoard' is excluded when retrieving application launches of a user. This application is nothing more than the homescreen of an iPhone which isn't regarded as a genuine application. The next section will reveal the performance measures used to assess our system and thus include the final adjustments made to support this.

7.3 Performance Measures

Finally after having gathered the results, they need to be evaluated. In order to determine how the system performs and to be able to compare results, some performance measures are needed. For all of these measures it holds that they have to be averaged out using a number of different users. A system might perform very good with a certain user, while not being able to detect patterns of others. The performance measures used are listed here, followed by a description of each of them:

- Precision (%):
  The percentage of predictions that was correct (followed by a corresponding launch).
- Recall (%):
  The percentage of launches that was correctly predicted (preceded by a prefetch).
- Freshness (CPD):
  A Cumulative Probability Distribution (CPD) of the number of seconds between the current launch and the previous load time. This previous load time can be a past launch

or a prefetch. The median freshness is a less elaborate indicator that will be used in determining the parameters first.

- Effectiveness:
  The proportion of the number of launches that was predicted to the number of predictions done.

Precision, recall and freshness are measures that are also used by PREPP [92]. Precision and recall tend to have a negative relation, when one rises, the other drops. When a lot of predictions are done, the chances that they are correct become lower, but the chance that a launch is predicted will become higher. Freshness is a measure proposed by the authors of PREPP to determine the relevance of a prefetch. The longer a prefetch is done before a launch, the less value it has as the data will become older. Finally effectiveness was added to the list. This value helps to determine the number of double prefetches. Whenever a single launch is successfully predicted 3 times for instance (which can happen when the lifetime of a prefetch is longer than the polling interval of an agent), this is of course redundant, but not noticed by any of the other measures.

In order to retrieve the data needed, the assessment agent saves logs to various CSV files. For each prediction agent and for itself it keeps track of the number of correct predictions and the number of launches predicted. Furthermore it determines freshness by keeping track of the last load time for each application for each agent. It also tracks a benchmark for this freshness by determining it when no agent does any prefetching.

7.3.1 Data Visualization

To draw conclusions from the performance measures, it is often useful to visualize the obtained data in a certain way. This way it can be easily seen what parameters lead to the best results. For this purpose two programs are considered, R and Excel.

R [118] is a statistical programming language that makes manipulating data and altering CSV files easy. Besides this, scripts can be created for it in order to be reused. As the experiments will tend to require a lot of the same evaluations, creating these scripts once can save a lot of time. The preparation of the data will thus be done using R, and more specifically Rstudio [97]. The scripts that were used to prepare the data properly are described in appendix B.

After having prepared the data, it will be visualized using Microsoft Excel [71]. This program allows for easy creation of graphs like the way the performance measures evolve and provides many ways to alter the look of a graph.

Chapter 8

Parameter Setting

The system deals with a number of different intelligent agents, each of which have parameters that can be set to a range of values. Before the results of this research can be obtained, these parameters have to be set or at least evaluated to know which ones to choose. As this requires a lot of different tests to be run and each of these tests have to be evaluated immediately, this separate section contains a small research into appropriate values for these parameters. The raw data gathered can be found in appendix D.

Because the entire dataset, as is concluded in section 7.1, is quite large and would take a long time to process for all combinations of parameters, a subset will be used. This selection of data is treated in section 8.1. Then the parameters for the prediction agents and the assessment agent will be tested in section 8.2 and section 8.3 respectively.

8.1 Test Data

In order to determine the parameters for the agents, data will be gathered from a selection of different users. It is important to use multiple users and average the results so that these results cannot coincidentally hold only for a specific case. Nevertheless due to time constraints on the research it was not possible to run these tests for all 34 different users present in the LiveLab dataset. As this phase only considers estimating the parameters, only a subset of the users is used to reduce the influence of exceptional cases.

The users are picked using the table on the LiveLab data in appendix A. From this set, only users that cover the maximum timespan were considered for parameter setting. These users include those with identifiers A and B. From these users, those with an exceptionally low Application Launch Count (ALC) were not considered either (B00, B11), as these sporadic users are not likely to benefit from this system. The users that were picked for this phase are listed below, along with a short motivation why they were included:

- A10, B10:
  These users have a high ALC and a low measure for Distinct Apps (DA). This means they use the same apps often and are thus assumed to be best candidates for the system.

- A09:
  This user has a high ALC and a high DA. He/she is still a good candidate for the system, but uses a larger amount of different applications.
- A08:
  This user has a low ALC and a low DA. Therefore he/she will probably not benefit a lot from the system and is expected to yield lower results.
- B08:
  This user has a low ALC and a high DA. For him/her the system is expected to yield even lower results. Apart from not using the phone often, a lot of different apps are used as well.

With the LiveLab data of these users, this chapter will determine what parameters will be used for each agent.

8.2 Parameters Prediction Agent

In order to determine the optimal parameters for the prediction agents, first a list of all possible values for these parameters will be given. From this range, a selection of values to test with are chosen and some presumptions on the expected best outcome will be discussed. Finally the described order will be used to determine the optimal value for each parameter.

8.2.1 Parameters

A list of possible parameters for the prediction agent, along with their possible values, those used for testing and the expected best one, is given in table 8.1.

	Possible values/options	Values to test	Expected
Learning time span	1: Day 2: Day (excl weekends) 3: Week	2	2
Polling interval	<0 sec, 30 min]	5, 10, 30 min	10 min
Quality threshold	[0, 1]	0, 0.1, 0.2, 0.3	0.1
Lifetime prefetch	<0 sec, 1 hour]	30 min	30 min
Q reward	1: 1.0 per correct prediction 2: 1.0 per correctly predicted use 3: -0.1 per incorrect prediction 4: limit reward [-1.0,1.0]	1, 2, 1+3, 1+4, 2+4	2+4
Q discount	[0, 1>	0, 0.1, 0.3, 0.8	0.1
Q learn rate	<0, 1]	0.1, 0.2, 0.5, 1	0.2

Table 8.1: Parameters for Prediction agent

The described parameters are explained more elaborately in section 6.2 about the implementation of a prediction agent. For each, an explanation for the values chosen is given.

Learning Time Span

Theoretically, the learning time span can be varied in all time ranges. The most interesting ones though will align with user cycles. With this a certain cycle of a user is meant. A day is an example of this, a user will behave the same to a certain extent in this time range, such as sleeping every night. As weekends tend to be an exception, these can be excluded from the learning process. Per week learning can be interesting as patterns might return on a weekly basis as well. Learning per month is considered too large an interval as learning will be excruciating slow.

For this parameter, only learning per day excluding weekends will be evaluated. Patterns per day are learned quickly and can adapt to changes far more quickly than weekly patterns. Furthermore, excluding weekends is expected to perform better or at least similar to including these.

Polling Interval

The polling interval can be anything above 0 seconds up until 30 minutes. The reason for choosing this limit is that predicting app launches each half an hour is estimated as a threshold for interesting predictions. When predictions are done with a longer interval, they are done even fewer than once per half an hour. It is likely that not much will be predicted and freshness of this data will remain very low. Furthermore, the lifetime of a prefetch should be longer than the polling interval. Otherwise it would not be possible to prefetch an app that occurs at the end of an interval, since the lifetime of the required prefetch will be expired when the launch occurs.

Polling at a low interval is expected to yield the best results. The lower this measure, the higher the recall as more predictions can be made. On the other hand this will lower precision for the same reason. This parameter should thus be small, but too small will lead to the agent differentiating between too narrow time intervals and not clustering the launches appropriately. Because of this phenomenon, 10 minutes is expected to perform best. 5 and 30 minutes are chosen also to test near the boundaries of this value.

Quality Threshold

The quality threshold can be any value between 0 and 1. 0 means that everything resulting from the Q-learning algorithm will be prefetched by the agent (except when negative reward is possible). 1 is considered an upper-bound, resulting in nearly no prefetches done. To be precise, the quality of a prediction can be higher than 1, depending on the maximum reward set and also the discount factor and learn rate. The other values will be chosen as such though, that a quality of 1 results in a very high confidence for the agent.

The best quality threshold is expected to be 0.1 for the polling interval of 10 min. The reason for this is that the threshold is nothing more than a cap on the predictions done, when no action seems to provide a good enough prediction. A low threshold yields more predictions and probably a higher recall, but a lower precision. 0.1 is expected to be a good value in-between, a blend between precision and recall. Besides it, some surrounding values lower and higher are tested. The reason that the polling interval is mentioned with this parameter is that the quality measures depend on this. The lower the polling interval, the more precise predictions and probably the lower the qualities of each prediction. These lower qualities might require a lower threshold in order for the agent to perform well.

Lifetime Prefetch

The lifetime of a prefetch was briefly mentioned at the polling interval. Setting it too high will result in agents believing a prefetch was correct even though it was done long before the actual launch. This phenomenon results in very long delayed results. If these results are delayed too long, the agent will not be able to determine the correct source of the high reward. So the lifetime will be set to 30 minutes always, as this is regarded reasonable. In short this means that if a prefetch is more than half an hour apart from its launch, it is regarded as incorrect.

Q Reward

The reward for the Q-learning reinforcement algorithm can be determined in multiple ways. 4 different types are considered here. The first option is by giving a reward of 1.0 for each prediction that was correct. Secondly, a reward of 1.0 can be given for each app launch done by the user that was correctly predicted. Furthermore a negative reward of -0.1 could be given for each incorrect prediction and finally the total reward can be limited. These options form the most basic ones. It is possible though too think of more complex ways to assign reward, but this will not be subject to this research.

The best option for reward is expected to be 1.0 per correctly predicted app launch in combination with limiting the total reward. This research focuses on maximizing the number of correctly predicted launches, not necessarily on maximizing the number of predictions that were correct. Of course this last measure is important and related, but the first is leading. The limit on reward ensures that these rewards do not become too large. This in its turn ensures that quality values for state-action pairs do not become too high and thus makes the system quicker in adapting to changes in user patterns. It should be noted though that this comes with a reduction in robustness when a temporary deviation of this pattern occurs.

Q Discount Factor

The discount factor can be set from 0 upto but excluding 1. A discount factor of 0 means that future states are not considered when valuing state-action pairs. If this would be 1, the qualities of those pairs could diverge in theory.

Too high a discount factor would also result in future states being more important than immediate reward, which is not expected to be desirable. Future states directly mean future times and even though future times can be of importance, observations at the current time will best predict what application to launch. Because of this the selected values are all in the lower section of the possible values and 0.1 is expected to perform best.

Q Learn Rate

Finally the learn rate is limited from 0 to 1, excluding 0 itself. When the learn rate is 0, nothing is learned and the old quality of state-action pairs always remains. When it is 1 though, the old value is not considered anymore. This means that only immediate reward and future state determine the quality.

The discount factor and learn rate are estimated with low values. The reason for this is that it makes the system more robust to deviations from user patterns. A high learn rate will quickly adapt to changing patterns, but also be quickly altered when an exception occurs. Since this short term learning can be beneficial, the entire range of values will be tested, but a lower one like 0.2 is expected to perform best.

8.2.2 Results Per Parameter

Determining what values to choose for the parameters is quite an elaborate task. Since trying all combinations is infeasible as the number of combinations is very large, an order in which parameters are chosen has to be determined. First the Q-learning parameters will be set, as they form the basis of learning and will likely perform approximately equal for different thresholds and polling intervals. From the Q-learning parameters, the reward is expected to have the most impact on the results, followed by the discount factor and the learn rate. This order will thus be followed to determine their values.

Q Reward

The parameters that are used for the simulation are listed below. Finally table 8.2 shows the results of the simulation. These are the averages of the 5 users selected before. As all users where tracked for approximately the same amount of time, these averages are not weighted.

Fixed parameters:		*Variable (Q reward):*
Polling interval:	10 min	option 1
Quality threshold:	0.1	option 2
Lifetime prefetch:	30 min	option 1+3
Q discount factor:	0.1	option 1+4
Q learn rate:	0.2	option 2+4

Q reward	Precision	Recall	Effectiveness	Freshness (median)
option 1	21.7%	32.9%	46.0	588.6
option 2	21.4%	33.7%	44.3	574.2
option 1+3	23.3%	25.4%	50.4	698.2
option 1+4	21.7%	31.9%	45.4	593.2
option 2+4	21.7%	31.8%	45.3	593.7

Table 8.2: Result Q reward

From the results gathered, the options can be evaluated. Option 3 gives a slightly best precision, because it gives negative reward for incorrect predictions. This increase comes with too large a drop in recall though. Furthermore option 4 limits the maximum and minimum reward. It doesn't seem to influence precision and recall a lot, though the slightly lower performance in recall and freshness gives the agents without option 4 a small advantage. This difference might be considered neglectable, but since a choice in parameter has to be made, those with option 4 will be discarded.

Choosing between option 1 and 2 is more difficult. As option 1 learns per correct prediction, it has a very small advantage regarding precision and effectiveness. From this the conclusion can be drawn that the agent with option 2 tries a bit more predictions, thus increasing its recall and freshness, but reducing its precision and effectiveness. Eventually both might be useful in a configuration with an assessor. For now option 2 will be chosen as the one to continue with.

The combination 2 and 4 was expected to perform best. Now it seems that the cap on reward is not necessary and even unwanted. From this it is concluded that the rewards are not excessive and the learning process is still able to adapt itself without it.

Q Discount Factor

The parameters that are used for the simulation are listed below. Finally table 8.3 shows the results of the simulation. These are the averages of the 5 users selected before. As all users where tracked for approximately the same amount of time, these averages are not weighted.

Fixed parameters:		*Variable (Q discount):*
Polling interval:	10 min	0
Quality threshold:	0.1	0.1
Lifetime prefetch:	30 min	0.3
Q reward:	option 2	0.5
Q learn rate:	0.2	0.8

Q discount	Precision	Recall	Effectiveness	Freshness (median)
0	21.7%	33.1%	45.1	580.4
0.1	21.4%	33.7%	44.3	574.2
0.3	20.2%	34.6%	41.7	563.1
0.5	18.9%	35.4%	39.0	554.7
0.8	17.9%	36.2%	37.2	544.2

Table 8.3: Result Q discount

The results lay again not that far from each other. A higher discount factor results in lower precision but higher recall. The higher discount factor means that future states become more important and thus the agent becomes less shortsighted. This can result in a few more mistakes as too long a foresight results in too early prefetches. On the other hand it predicts more applications as states nearby will get higher quality and thus prefetch earlier. All in all the discount value will be chosen at 0.3 as this seems to be a nice balance, precision is still in slow decline and recall as always in slow increase.

The expected value for this parameter was 0.1. As the results lay not that far off, this would still have been a good choice.

Q Learn Rate

The parameters that are used for the simulation are listed below. Finally table 8.4 shows the results of the simulation. These are the averages of the 5 users selected before. As all users where tracked for approximately the same amount of time, these averages are not weighted.

Fixed parameters:		*Variable (Q learn rate):*
Polling interval:	10 min	0.05
Quality threshold:	0.1	0.2
Lifetime prefetch:	30 min	0.5
Q reward:	option 2	0.8
Q discount:	0.3	1

Q learn rate	Precision	Recall	Effectiveness	Freshness (median)
0.05	19.4%	37.3%	39.7	542
0.2	20.2%	34.5%	41.7	564.6
0.5	21.1%	31.7%	43.9	589.8
0.8	21.1%	28.8%	44.6	621.6
1	20.3%	15.7%	45.1	742.8

Table 8.4: Result Q learn rate

In the results it is clear that a learn rate of 1 is inferior. As was expected this disregards any past results and thus only uses the last use as an indicator. For the other values it holds that a choice should be made between precision/effectiveness or recall/freshness. As a high recall and low freshness is the goal and the differences in precision and effectiveness are limited, the value with the highest recall/lowest freshness will be chosen, which is 0.05.

As an expected value, 0.2 was chosen. A low value was expected due to the fact that it makes learning more robust. That an ideal value would be this low though wasn't expected, as it is not very flexible when patterns are changing. It seems though that these patterns either do not change often, or the agent adjusts himself enough using a low learn rate.

Polling Interval and Quality Threshold

The parameters that are used for the simulation are listed below. Finally table 8.5 shows the results of the simulation. These are the averages of the 5 users selected before. As all users where tracked for approximately the same amount of time, these averages are not weighted.

Fixed parameters:

Lifetime prefetch:	30 min
Q reward:	option 2
Q discount:	0.3
Q learn rate:	0.05

Variable (Poll. interval - Qual. threshold):
30 min - 0.3
30 min - 0.2
30 min - 0.1
10 min - 0.2
10 min - 0.1
10 min - 0
5 min - 0.1
5 min - 0

Polling — Quality threshold	Precision	Recall	Effectiveness	Freshness (median)
30 min — 0.3	26.7%	24.9%	89.4	707.2
30 min — 0.2	25.5%	25.7%	85.8	692
30 min — 0.1	24.1%	26.1%	81.7	683.4
10 min — 0.2	22.2%	34.4%	45.3	592.6
10 min — 0.1	19.5%	37.5%	39.8	543.1
10 min — 0	18.6%	37.7%	38.3	539.6
5 min — 0.1	14.4%	43.5%	24.3	507.4
5 min — 0	13.2%	44.4%	22.6	499.6

Table 8.5: Result Polling interval & Quality threshold

The first thing that can be noted in the results is the relation between the polling interval and the freshness. As was to be expected, a smaller polling interval has the capability of predicting more precisely and thus reducing freshness. As a cost this comes with a lower precision and effectiveness, since more mistakes are made. For each polling interval, the goal of the prediction agent will differ, as it typically is more interesting for a certain purpose. The 30 minutes polling interval is clearly interesting if precision and effectiveness are deemed important. As values are close, a quality threshold of 0.3 would be the best choice to maximize these values. For 10 minutes polling, the recall and freshness will be more suitable goals. Since the differences between a threshold of 0 and 0.1 are not astounding, 0.1 will be chosen as a best quality threshold. Finally for a polling interval of 5 minutes, 0 is chosen as threshold to maximize recall and freshness.

The expected best combination for this variable was 10 minutes along with a quality threshold of 0.1. This was eventually deemed as one of the best options as well. As was predicted, the polling interval used is highly influential in what is favored, more accurate predictions, or more predicted application launches.

8.3 Parameters Assessment Agent

Now that the characteristics of the prediction agents have been investigated, the assessment agent has to be optimized. For this a set of prediction agents is needed, of which the assessment agent has to combine the predictions. In order for the assessment agent to best leverage its full potential, it seems best to have a diverse blend of prediction agents. The chosen agents are specified in table 8.6 and motivated below.

	Agent 0	Agent 1	Agent 2
Polling interval	30	5	10
Quality threshold	0.3	0	0.1
Q reward	1+3	2	2
Q discount	0	0.8	0.3
Q learn rate	0.8	0.05	0.05

Table 8.6: Configurations for Predictor Agents

Agent 0 is reserved to optimize precision and effectiveness. From the results gathered in the previous section, the values for the parameters are chosen. Agent 0 has all parameters that have been reported to generate the best precision/effectiveness. Agent 1 has all parameters that have been reported to generate best recall/freshness. Lastly Agent 2 represents the overall best parameters found in the previous section.

8.3.1 Parameters

A list of possible parameters for the assessment agent, along with their possible values and the expected best one, is given in table 8.7.

	Possible values/options	Values to test	Expected
Lifetime prefetch	<0 sec,1 hour]	30 min	30 min
Quality measure	1: precision of agent 2: recall of agent 3: 0.5 * precision + 0.5 * recall	1, 2, 3	3
Protocol consecutive prefetches	1: prefetch again 2: don't prefetch 3: prefetch again (only after 10 min) 3: prefetch again (only after 20 min)	1, 2, 3	3

Table 8.7: Parameters for Assessment agent

The parameters described here are explained more elaborately in section 6.2 about the implementation of a assessment agent. For each of them, an explanation for the values chosen in the table is given below.

Lifetime Prefetch

As has been discussed when determining the optimal parameters for the prediction agents, the lifetime for a prediction will be set to 30 minutes. The higher the lifetime is set, the higher the precision and recall will be evaluated for each agent, because a prefetch is deemed correct more often in this case. This is of course only an illusion, as prefetches are still done the same way with either lifetime. The freshness value is not influenced by this and thus better used for comparisons between different systems that value precision and recall differently.

Quality Measure

The quality of an agent can be measured and limited using different measures. These measures consist of the precision, recall or a combination of these values for an agent. The measure chosen determines the focus of the assessment agent and thus what is deemed more important by him. To the quality measure belongs a quality threshold. This threshold can make sure predictions of agents that are performing bad on the chosen measure are excluded.

The parameter of the quality measure will most likely determine where the focus of the agent lies. Focusing solely on precision or recall is expected to yield high results in either one of the area's, but probably a mix of these measures is preferred. As this research doesn't concern any issues regarding battery life or data usage yet, a slight preference lies on the recall/freshness.

Protocol Consecutive Prefetches

The protocol for consecutive prefetches concerns the case where a prefetch has already been done by the assessment agent and is predicted again by another agent. The assessor has 3 options to handle this, it can prefetch it anyway, confine consecutive prefetches to happen only after a certain amount of time has passed since the previous prefetch or do nothing at all.

When the protocol for consecutive prefetches is set to stop prefetching when this has already been done, the recall is expected to get lower along with higher freshness. That is because the prefetch would prevail longer if it was refreshed. The plusside of it though is that it greatly reduces the number of preformed prefetches and thus improves effectiveness. With other protocols, prefetches can be done several times within a lifetime of prefetch, reducing effectiveness. In order to keep the improved recall and freshness while not overreacting on effectiveness, the best solution expected is to only prefetch after a certain amount of time has expired since the previous prefetch.

8.3.2 Results Per Parameter

Just as in the search for optimal parameters of the prediction agent, not all combinations can be assessed here either. As the protocol for consecutive prefetches is expected to alter performance regardless of the quality measure used, this will be assessed first. This protocol is deemed more important as it has most impact on the aspects of bringing multiple predictors together.

Protocol Consecutive Prefetches

The parameters that are used for the simulation are listed below. Finally table 8.8 shows the results of the simulation. These are the averages of the 5 users selected before. As all users where tracked for approximately the same amount of time, these averages are not weighted.

Fixed parameters:	
Lifetime prefetch:	30 min
Quality measure:	option 3
- threshold:	0.2

Variable (Protocol):
option 1
option 2
option 3 (10 min)
option 3 (20 min)

Protocol	Precision	Recall	Effectiveness	Freshness (median)
option 1	17.4%	46.3%	16.7	430.8
option 2	20.3%	28.2%	55.4	693
option 3 (10 min)	15.7%	45.9%	23.7	461
option 3 (20 min)	14.1%	44.6%	30.0	517.6

Table 8.8: Result Protocol Consecutive Prefetches

Option 2 has a slight advantage in precision and quite a big one in effectiveness. This results in a very low relative recall and a very high freshness. Considering this drop, option 2 will be regarded as inferior. When looking at the other results and disregarding effectiveness, the order of performance is clear. Option 1 performs best, then option 3 (10 min) and finally option 3 (20 min). Now option 1 and 3 both prefetch again, only 3 puts some limitations on when to prefetch again. Since option 3 prefetches less applications, its effectiveness increases, since it makes less double prefetches. This effectiveness difference is expected to only increase when multiple similar agents are present, that is because these agents will prefetch more of the same applications, all of which will be granted by option 1 and none of which will be by option 2. Considering this will have most drastic results in option 1 and option 3 reduces this, option 3 will be chosen. Considering the fact that freshness is superior to effectiveness in this research, option 3 (10 min) will be chosen.

Option 3 was the expected outcome of which 10 minutes appeared to have a small advantage. Another possibility for future research would be to look at delaying prefetches, although this would not be following the advice of the predicting agents. Because of it this will likely cause more incorrect prefetches and keep freshness high.

Quality Measure

The parameters that are used for the simulation are listed below. Finally table 8.9 shows the results of the simulation. These are the averages of the 5 users selected before. As all users where tracked for approximately the same amount of time, these averages are not weighted. The threshold for the quality measure is valued along with expected values for the measures. As precision has been reported lower in previous results, the threshold to test this measure with is set lower as well.

Fixed parameters:	
Lifetime prefetch:	30 min
Protocol consecutive prefetches:	option 3 (10 min)

Variable (Quality measure - threshold):
option 1 - 0.1
option 1 - 0.2
option 2 - 0.2
option 2 - 0.3
option 3 - 0.1
option 3 - 0.2
option 3 - 0.3
option 1||2||3 - 0

Quality measure	Precision	Recall	Effectiveness	Freshness (median)
option 1 - 0.1	15.8%	44.5%	23.8	485.4
option 1 - 0.2	18.0%	39.0%	27.1	515
option 2 - 0.2	14.9%	47.9%	22.7	427.4
option 2 - 0.3	15.5%	45.9%	23.3	443.5
option 3 - 0.1	14.9%	48.4%	22.7	424.8
option 3 - 0.2	15.7%	45.9%	23.7	461
option 3 - 0.3	16.4%	41.5%	24.6	500.4
option 1||2||3 - 0	14.6%	48.9%	22.4	423.2

Table 8.9: Result Quality Measure

As expected option 1, which focuses on precision, yields the highest precision and effectiveness. Option 2 focuses on recall and thus yields a relative higher recall and lower freshness. Option 3 finally falls in between these two considering the same threshold values. Interesting enough though is that a threshold of 0, in which case it doesn't matter anymore which option is chosen as every quality measure for each agent is above 0, seems to yield most interesting results. For this research freshness and recall are deemed the most important parameters, but precision and effectiveness should be maximized closely after. As these last two parameters suffer very lightly from choosing a threshold of 0, while it does optimize freshness and recall a lot, this option will be chosen.

Option 3 was expected to yield the best results. That was because a large drop in precision and effectiveness was expected for option 2. As it turns out this drop is very small and thus acceptable for the increased performance. Finally it seems that it is best to use no filter except for the protocol of consecutive prefetches. Every prediction done by any agent of course benefits recall and freshness, but it also seems to have only a very light negative impact on precision and effectiveness.

Chapter 9

Results

With all of the system in place, the setup for testing made and the parameters investigated, the final results of the system can be gathered. Some expectations about the results will first be set. Then the system will be evaluated on the aforementioned performance measures and it will be compared to the existing literature on this topic. Appendix D refers to the raw data gathered.

9.1 Expectations

As can already be seen in chapter 8 on parameter setting, the prediction agents along with reinforcement learning are able to predict user behavior and successfully prefetch the correct applications. Using the full potential of the Multi Agent System along with the assessment agent, is expected to enhance these results. Overall the expectation is that the system is suitable for these predictions and shows promising results.

Though the benefit of the techniques used are expected to be seen. It is not expected that the results are groundbreaking. Since learning is only based on time of day, which is very limited based on related work, results will not likely be better than this work. Furthermore, the benefit of the MAS will mostly be seen when different kinds of prediction agents are used. Now these agents are very similar, with only their parameters set differently. Because of this, the improvement of the assessment agent over the separate predictors will not be high.

9.2 Performance of System

Now that the expectations are set, let's have a look at the performance of the system. First the configuration of the system is set after which the results can be gathered.

9.2.1 Configuration

In order to test the system, user data is needed. As these results are important for the evaluation of the system, more users than before are used and explained first. Then the configurations for the prediction agents along with the assessment agents are discussed.

Test Data

In section 8, where the parameters of the agents were investigated, only 5 users from the LiveLab dataset were used. The reason for this was that a lot of tests had to be run while limited time was available and for these small tests a smaller set of users would be acceptable for significance.

These results gathered here though are used to evaluate the performance of the system with. Thus significance is of higher importance here, which is why as many users as possible were tested. To be able to evaluate as long a period as possible, all users with prefixes 'A' and 'B' were chosen from the set shown in appendix A. Users with prefix 'D' where tracked for a considerably smaller timespan. As the smallest timespan determines the time that can be evaluated and a longer time allows for better evaluation, these were not included.

Prediction Agents

For the prediction agents, a large range of diverse agents was chosen, as the assessment agent is expected to perform best with a large and diverse set of predictors. Table 9.1 shows the configurations of the prediction agents. Following the table is a description of the choices.

	Agent 0	Agent 3	Agent 4	Agent 5
Polling interval	30	30	10	10
Quality threshold	0.3	0.3	0.1	0.1
Q reward	1+3	2	1+3	2
Q discount	0	0.3	0.3	0
Q learn rate	0.8	0.05	0.05	0.8
	Agent 1	**Agent 6**	**Agent 7**	**Agent 2**
Polling interval	5	5	10	10
Quality threshold	0	0	0.1	0.1
Q reward	2	2	2	2
Q discount	0.8	0.3	0.8	0.3
Q learn rate	0.05	0.05	0.05	0.05

Table 9.1: Configurations for Predictor Agents

Agent 0 till 2 are kept the same as before when determining the optimal parameters for the assessment agent in section 8.3. Agent 0, 3, 4 and 5 are reserved to optimize precision and effectiveness. From the results gathered in the previous chapter, the values for the parameters are chosen. Agent 0 has all parameters that have been reported to generate the best precision/effectiveness. Agent 1 till 3 have the determined optimal parameters with respectively the polling interval/quality threshold, Q reward and Q discount/learn rate optimized. Agent 1 has all parameters that have been reported to generate best recall/freshness. Then Agent 6 and 7 each have the before determined best parameters with respectively the polling interval/quality threshold and Q discount optimized (the other parameters were already optimal). Lastly Agent 2 represents the best parameters found in the determination of parameters for the prediction agents.

Assessment Agent

Finally the assessment agent has to be configured appropriately as well. As reported in section 8.3, option 3 of the protocol for consecutive prefetches with 10 minutes is argued to perform best. This protocol, that doesn't prefetch the same application twice within 10 minutes, is used for the assessor. Furthermore the quality measure was chosen with a threshold of 0, essentially meaning that no measure will be used and all prefetches of all agents will be done. This was reported to yield best results.

9.2.2 Results

In order to discuss the characteristics of the system, several experiments can be done. To determine the primary performance measures of the overall system and prepare the data, R scripts were used to transform the data for all users and gather it in a single file. These scripts can be found in appendix B.2. Finally Microsoft Excel 2013 was used to plot the actual graphs, except for the freshness graph. This one was made using R since the files needed for it where too large to be loaded into Excel.

Two types of tests will be done. First the data will be shown in an accumulated fashion, thus always the average of all statistics done before. The other type of test will consider the measures per day. This way more characteristics of the learning process can be extracted.

Measures Accumulated

The values that were obtained with the configuration run are displayed in table 9.2. Here one can see that each agent reduces freshness as opposed to the benchmark. The benchmark is the freshness of each application when no prefetches are done. It can also be noted already that the assessment agent performs best on recall and freshness, worst on effectiveness and nearly worst at precision.

	Precision	Recall	Effectiveness	Freshness (median)
Benchmark				854
Assessor	8.6%	45.6%	10.3	520
Agent 0	21.6%	8.4%	54.6	578
Agent 1	9.0%	39.0%	13.3	548
Agent 2	12.9%	29.0%	22.6	546
Agent 3	18.6%	16.5%	49.2	565
Agent 4	9.7%	8.8%	18.1	582
Agent 5	13.0%	15.7%	24.3	590
Agent 6	8.3%	36.0%	12.5	576
Agent 7	12.4%	31.0%	21.7	571

Table 9.2: Per Agent Results

Besides these tabular values though, graphs often tell a lot more about data and make sure data is easier compared. Figure 9.1, 9.2 and 9.3 respectively show the average precision, recall and effectiveness for each agent as a function of time. Each of these graphs has the average percentage on its y-axis and the time in days on its x-axis.

In the graph for precision it is clear that agent 0 and 3 yield the highest scores. This goes hand in hand with a relative lower score in recall for these agents. These agents also perform exceptionally well in effectiveness. In recall, agent 1, 2, 6 and 7 yield best results. Along with the remaining agents, 4 and 5, they perform lower in precision and effectiveness. Actually agent 4 and 5 manage to score below average on all 3 performance measures.

The assessment agent can be seen to outperform every single agent in recall. This feature comes at a cost though as its precision is consistently lower and its effectiveness even stays lowest of all agents.

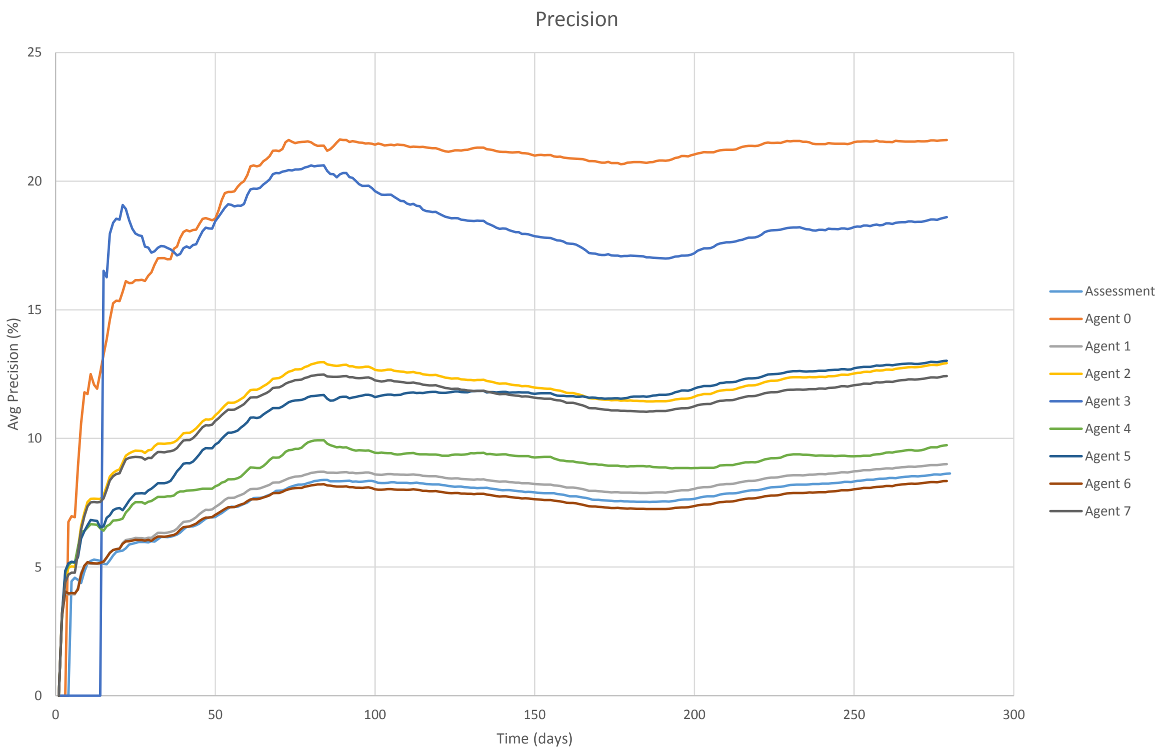


Figure 9.1: Average Precision

Figure 9.4 shows the Cumulative Probability Distribution of the freshness values for each agent. This graph can easily be used to read the difference in freshness obtained by the program. The higher the cumulutative probability for a certain time, the lower freshness was in the applications. As can be seen, all agents improve freshness regarded to the benchmark (black line). The benchmark represents the case where no prefetches are done and thus only the previous launch of an application determines its freshness. All agents perform similar based on freshness, they are all drawn with grey lines since the difference is too small to notice. The red line finally represents the assessment agent and can be seen to result in the best freshness.

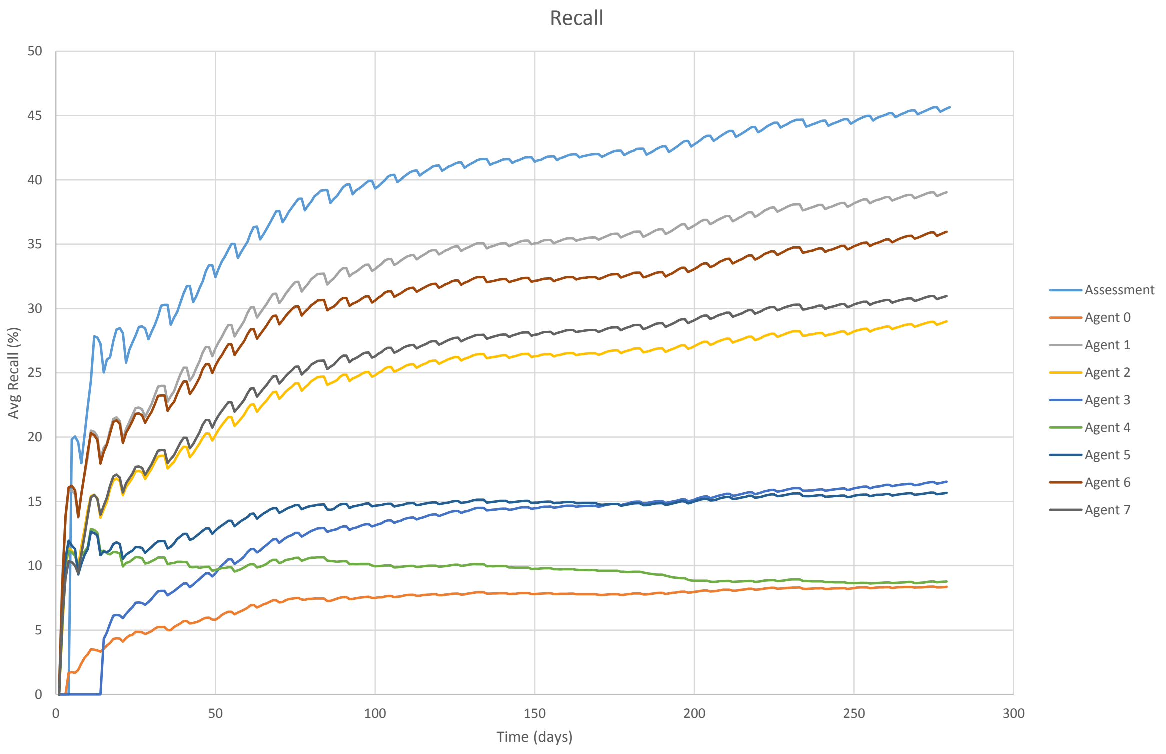


Figure 9.2: Average Recall

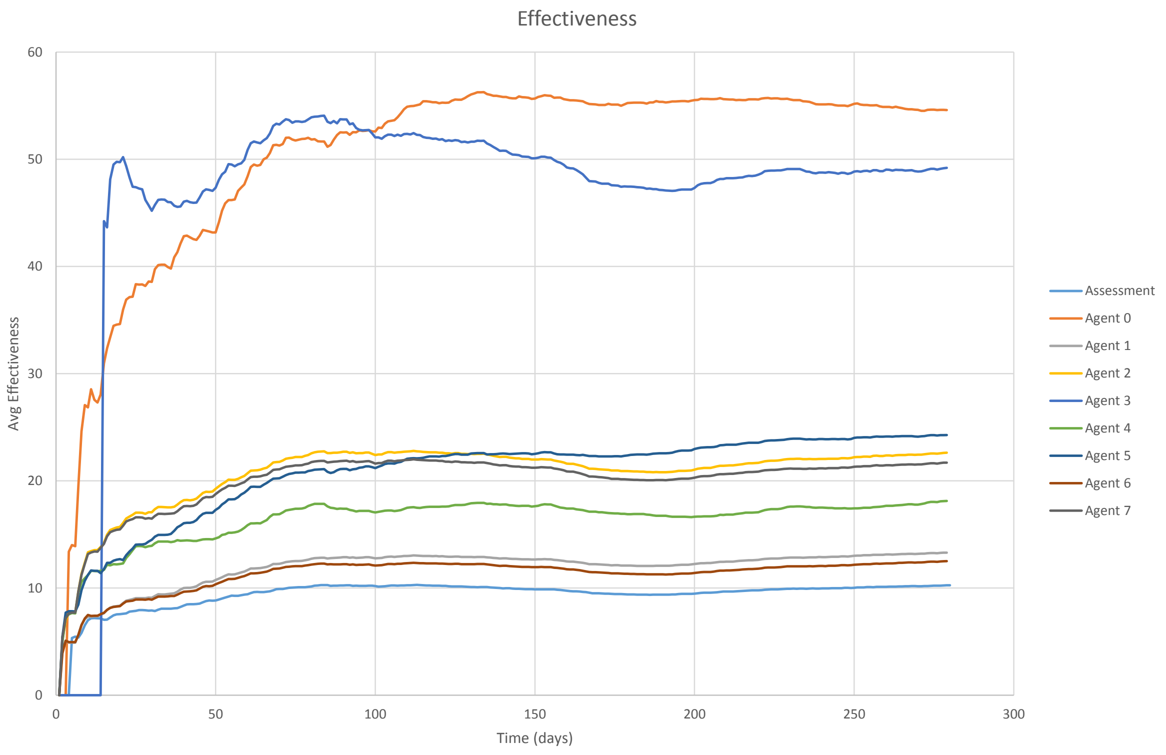


Figure 9.3: Average Effectiveness

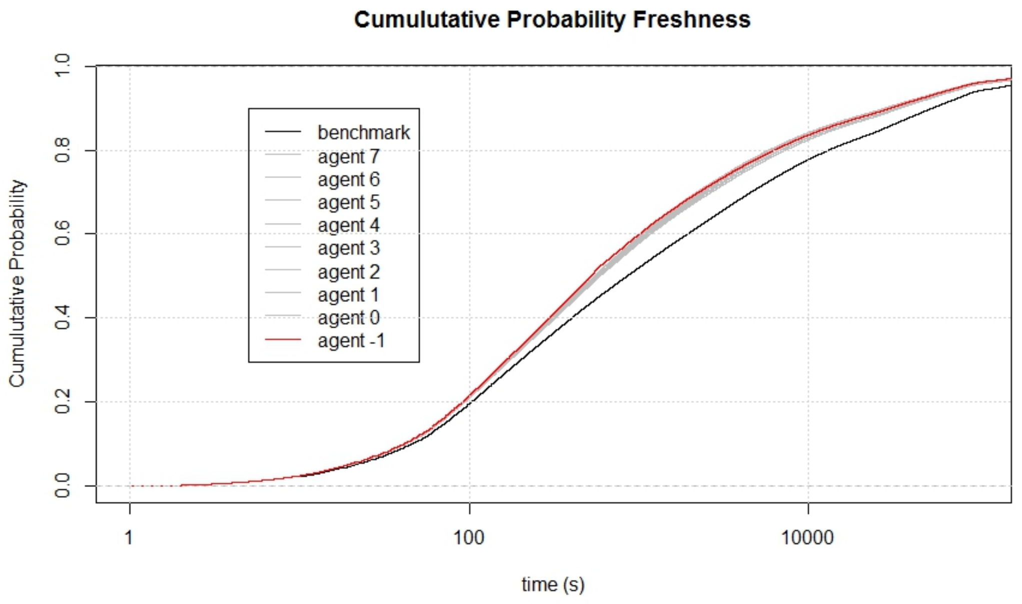


Figure 9.4: CPD Freshness

Results per Day

In order to further investigate the results obtained from the system, the performance measures will be calculated per day as well. The previous section showed the averages of the performance measures of all the history, now the performance measures are calculated per day and displayed in the following graphs. This is done only for the assessment agent. The patterns that can be seen here are similar for all other agents. Figure 9.5 shows the precision, figure 9.6 the recall and figure 9.7 the effectiveness.

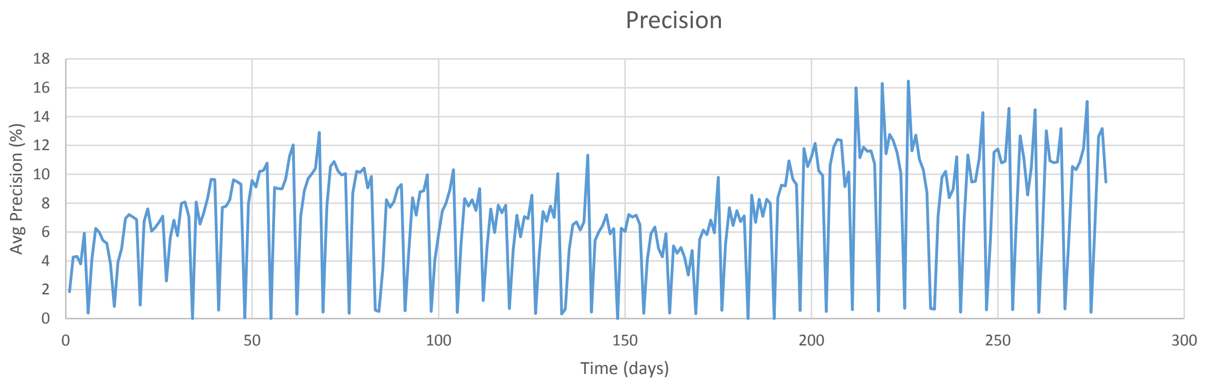


Figure 9.5: Average Precision Assessor (Daily)

All graphs first start with an increase in performance as the agent learns the user patterns. The short bursts to zero can be explained by the fact that the agents do not learn during weekends, and thus achieve zero for all performance measures during these times. Furthermore, a small decline can be noticed between 100 and 150 days. Since data gathering of users started in February, this period falls approximately around June and July, which is the most common season for holidays.

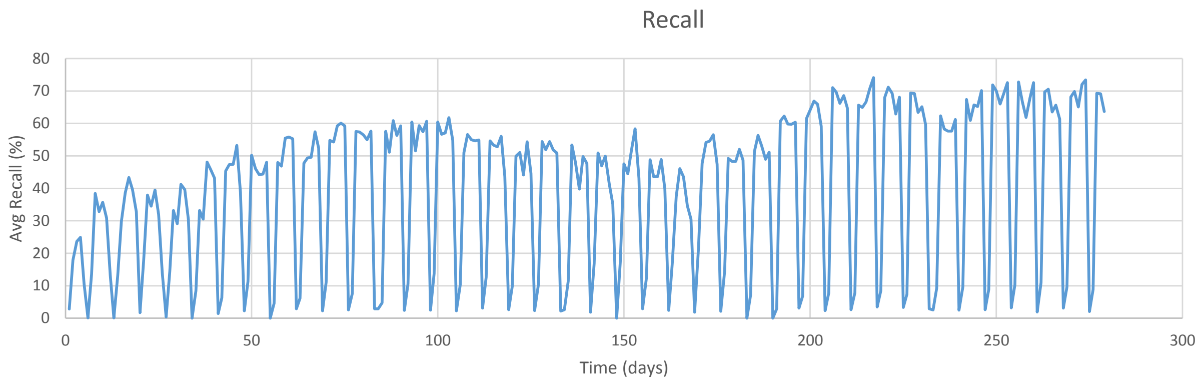


Figure 9.6: Average Recall Assessor (Daily)

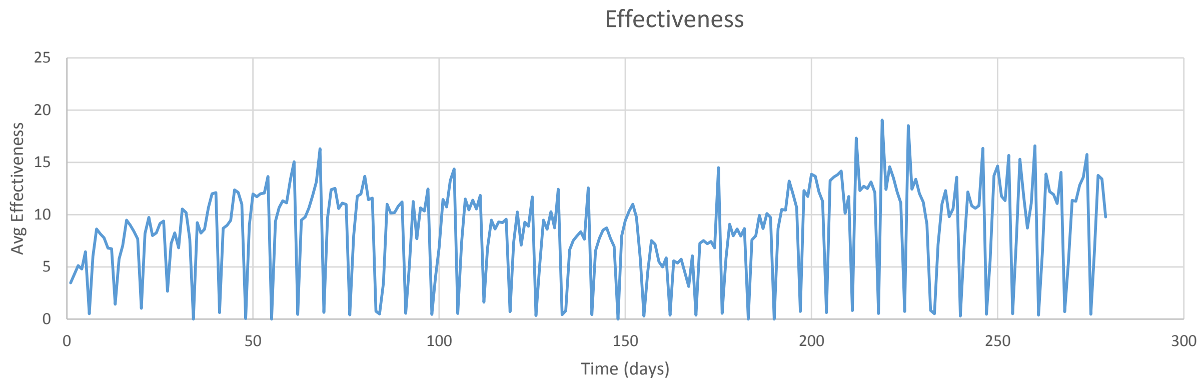


Figure 9.7: Average Effectiveness Assessor (Daily)

9.3 Compared to Related Work

In section 2.3 the algorithms FALCON and PREPP were mentioned as two predictors of application launches. In order to verify the results obtained in this study it is important to compare them to the ones obtained by related work. This section will discuss both FALCON and PREPP although there are a number of difficulties encountered for both.

9.3.1 FALCON

Let's first consider FALCON. This algorithm only predicts application launches that occur after a first application has been opened already. Right after unlocking a phone, nothing is predicted yet, only when a first launch has been detected the algorithm starts working. This cannot be compared to predicting launches at every given moment, as the latter will always be performing worse. Thus this algorithm will not be compared to the one discussed in this report.

9.3.2 PREPP

The other algorithm discussed in section 2.3 was PREPP. This algorithm does try to prefetch applications at any given time. It determines the order in which app launches occur, from this predicts which one will be next and finally learns a model to decide the time before this app will be launched. They use a couple of ways to measure performance. One considers the accuracy of the top 5 ranking for the next app to be used. Besides this the freshness of the Facebook and E-mail app is evaluated. Finally the overall system is evaluated using a controlled user study of a maximum of 2 weeks. For this study 1 week was used to let the algorithm learn and only 1 week was used to obtain performance measures.

The first measure is again not useful to compare with this study. The top 5 ranking says nothing about the time it will be used. This could very well be hours ahead. The proposed system doesn't work this way since time is inherent in its predictions. The ranking for next apps cannot be easily extracted from this, it would require the algorithm to look ahead for an unknown amount of time and extract the most likely apps to be launched. This unknown time forms the problem though.

The second measure, freshness, could be used to compare the results. The freshness represents the time between a prefetch and a launch. When no prefetch was done, it represents the time between two launches of a certain app. In the proposed system, this can be extracted quite easily. Figure 9.8 and 9.9 show the freshness as reported by PREPP for Facebook and E-mail respectively. The same graph is created using the system of this research and results are shown in Figure 9.10. The R scripts used to extract them are shown in appendix B.3. The graphs show that the freshness of the benchmarks in this study and PREPP's coincide, which shows this measure is likely to be calculated in the same way. The algorithms of FALCON and PREPP both acquire substantially better freshness values.

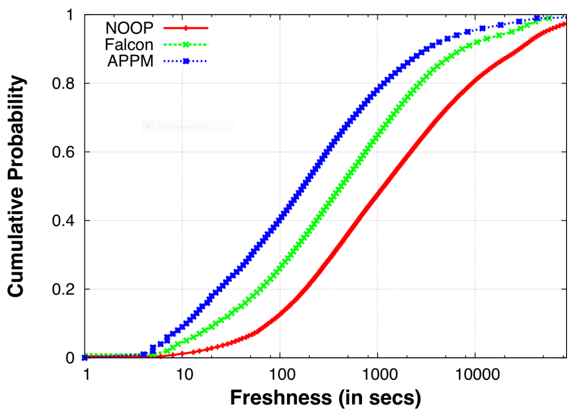


Figure 9.8: PREPP: Freshness Facebook

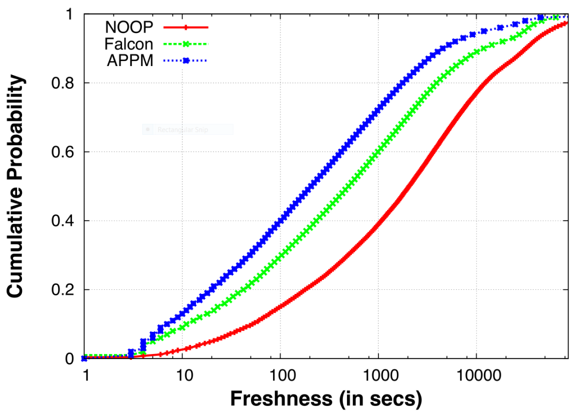


Figure 9.9: PREPP: Freshness E-mail

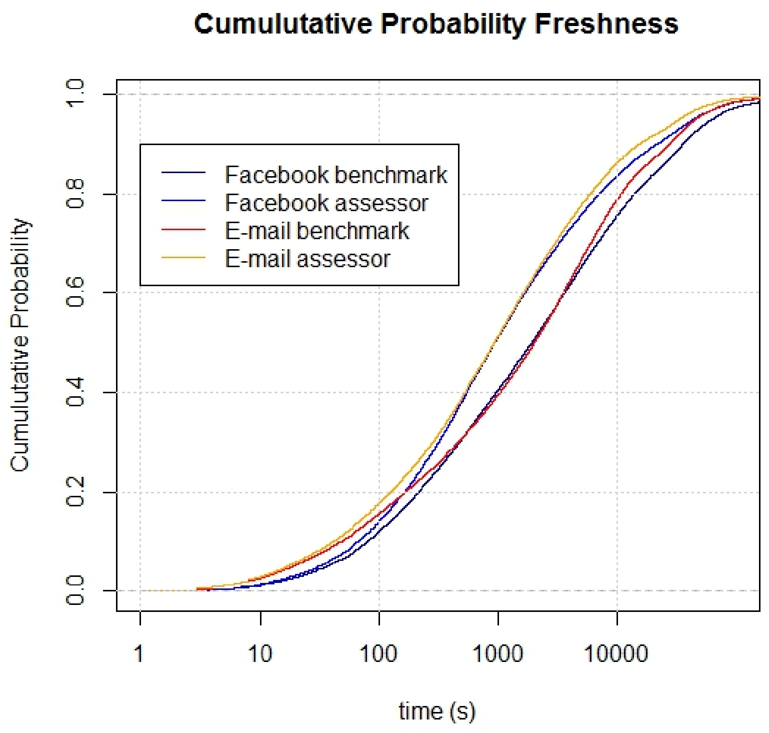


Figure 9.10: CPD Freshness of Facebook and E-mail

Finally the controlled user study of PREPP resulted in a median freshness of 2.7 minutes along with a precision of 22.12% and a recall of 46.87% [92]. Polling each 15 minutes resulted in a median freshness of 247 seconds while this resulted in 3.58 times the number of prefetches done compared to PREPP.

In order to compare to PREPP, the results obtained with the assessment agent can be used. These results can be found in table 9.2. The median freshness for the system is reported to be 520 seconds, or roughly 8.7 minutes. The average precision was 8.6% and average recall 45.6%. PREPP is not clear as to which applications are polled and thus it is not clear what 3.58 times means. Assuming all applications on the phone are used in this calculation, this means that PREPP does a prediction every $3.58 \cdot 15 = 53.7$ minutes per application. Using the R script listed in appendix B.3, the average time between prefetches for the system of this research is calculated to be 670.3 minutes, or 11.2 hours.

Chapter 10

Discussion

Now that the results have been gathered, a discussion as to how to interpret these results is up next. Section 10.1 will cover this. Besides only discussing the results though, during implementation all sorts of challenges were encountered. These hiccups and how they were resolved are discussed in section 10.2.

10.1 Results

The system that has been brought about in this research showed some interesting results which were presented in chapter 8 and 9. These chapters evaluated the parameters of the framework and the overall gathered results respectively. Here they will be discussed in the same order.

10.1.1 Parameter Setting

Before being able to assess the system, the parameters of all agents had to be evaluated. This was done in chapter 8. Here a division was made in determining parameters for the predictors and for the assessor, as will be done now.

Parameters Prediction Agent

The predictors first of all have a vast amount of parameters that can be altered. This changes their behavior and determines the performance measures that are deemed most important by them. As all different combinations are unfeasible to investigate, some choices had to be made. Because of this, some parameters were kept static, others were evaluated in selected ranges and all parameters were assessed one by one instead of testing all combinations. These decisions somewhat limit the value of the experiments, but using good reason they proved to give very good insights in the effects of altering each different parameter.

Since a conclusion of the parameters was needed for further testing, the results here have been discussed to a certain extend in this chapter already. Nevertheless some remarks can still be made on some of the parameters.

The discount factor doesn't seem to have much influence on the outcome of the performance measures. This factor determines the weight of the next state in the quality determination of a state-action pair. This small difference seems to hint that whether the future state is considered important or not doesn't influence learning much. This means that the quality of a state one timestep later adds little information to the learning process.

Overall for all parameters, it can be noted that the effectiveness measure coincides heavily with the precision measure. The effectiveness essentially describes the percentage of total predictions that let to a successfully prefetched application. When a single prediction is valid for multiple application launches, the effectiveness is high. While the case where multiple predictions are done for a single launch, the effectiveness is low. Prediction agents will rarely find themselves in these cases as they account for the predictions they already did. This performance measure will be more useful for the assessment agent as multiple of the same predictions in the same timespan are more likely to occur there.

Parameters Assessment Agent

As of writing, the assessment agent is only a simple agent that filters the predictions of multiple predictors. Due to this it has less parameters to choose from at the moment. Again some choices were made in the order in which the parameters were set, just like in the predictors. This order was to first investigate the protocol for consecutive prefetches. Both parameters have been discussed and concluded on in section 8.3. Besides the discussion there, still some remarks on the quality measure can be noted.

This quality measure was evaluated using 3 well performing agents. Partly because of it, the way each agent was evaluated didn't matter all that much, the more predictions were let through, the better. Because of this a threshold of 0 came out on top. Something to consider though is that this threshold might be relevant when bad performing agents are present. The predictions of these agents will then be filtered out and not ruin the performance of the assessor.

10.1.2 Performance System

The overall system has been assessed in section 9.2 with agents selected using the parameters found in section 8. Looking at the graphs of the performance measures, it is clear that the assessment agent mostly coincides with agent 1 and 6. Those agents have the smallest polling interval and thus make the most predictions. Since the assessor is not that sophisticated in its filtering yet, all of these predictions are actually done, unless the protocol for consecutive prefetches states otherwise. Nevertheless it is clear that the assessor is able to obtain better recall and also better freshness as all of the other agents.

Agents 4 and 5 can be noted to perform relatively bad on all performance measures. Both the agents are designed to perform best on precision and effectiveness. For these measures, the polling interval and threshold are shown to have largest influence. Therefore focusing on precision and effectiveness, without the appropriate polling interval and threshold, turns out to be useless.

The curve of accumulated recall and all of the curves that show the daily performance measures all show periodic drops. In the accumulated recall these are short setbacks but when looking at the daily measures, the reason becomes more clear. The fact that the performance measures all have a short burst to 0 can be explained by the fact that they do not learn during weekends. Every weekend all performance measures should be 0 as nothing is done by any of the agents. The fact that they're not completely 0 can be caused by some predictions being done in the transition periods from weekend to week. These drops off course heavily reduce the average performance measures. In the daily graphs one can see the actual values reached for each performance measure by the assessment agent. Recall can be seen to reach values around 70%.

In these daily graphs one can also see a small drop of the performance measures after about 100-150 days. As noted in the results section, this period of time falls approximately around June and July, being the most popular months for a holiday. These are the hottest months in Texas, the region where the data was gathered. These drops could thus be explained by the fact that user patterns alter somewhat in the holiday season, as users tend to use their phones differently. Nevertheless the algorithm maintains quite steady results and recovers immediately after.

The fact that daily effectiveness measures for the assessment agent hover a bit above 10 roughly means that approximately 10 predictions were done per correctly predicted application. Also freshness can be seen to improve in the cumulative probability distribution made with it. These measures are difficult to talk about though, as they only indirectly and roughly represent the benefits the system has and the costs that come with it.

10.1.3 Comparing to Related Work

The system is finally compared to the paper of PREPP which comes closest to this research. This comparison is difficult since PREPP rapports some choices poorly and calculates costs for the mobile device where this research does not. This thesis focuses solely on the predictive capabilities while considering measures that indicate these costs like precision and effectiveness. Furthermore PREPP is expected to learn throughout all days in a week, instead of disregarding weekends. First a comparison based on freshness is done, where it is explained that this is the only reliable way to compare the systems. After that some other ways to compare are mentioned.

Freshness

In order to compare to PREPP, freshness was reported in the results. From the performance measures gathered this is the only one probably calculated in the same way. PREPP isn't clear on how precision and recall are determined. To explain how these differences in calculation can occur, a small example will be used.

Say Facebook is prefetched at time 0 and opened at time 30 without any applications opened in between. This could be considered a correct prefetch as Facebook was the first app to be opened after the prefetch. This system would deem the prefetch incorrect when E-mail was opened at time 15, since then E-mail would have been the next application from the time of prefetch. Another approach, the approach of this research, is to determine the lifetime for a prefetch. If an app is opened before this lifetime, the prefetch is correct. Now if the lifetime in this example is 60, the prefetch will be deemed correct regardless of the E-mail launch in between. But if the lifetime is set to 15, the prefetch will be deemed incorrect, even when E-mail was not launched in between. These things can make a large difference in the outcome of precision and recall. As can be seen in the example, a high lifetime guarantees a higher percentage of correct prefetches, even though the actual prefetches are the same.

Freshness is independent of this phenomenon and thus is concluded to be the best way to compare the systems. Let's first discuss the freshness values reported on Facebook and E-mail. Both PREPP and FALCON outperform the methods of this research. This means the prefetches of those other algorithms either are closer to the actual time of the launch, or more of the launches are predicted correctly. This last statement is not likely to be the case as PREPP reports similar recall values to the proposed system in their controlled user study. The freshness difference thus has to be caused by the fact that predictions of FALCON and PREPP are closer to the actual launch times of apps. This might be explained by the small portion of features considered for the algorithm in this research. Only time of day was expected to perform worse and thus it is no surprise that PREPP and FALCON are performing better. Furthermore the polling time of agents in this system is 5 minutes, which results in fixed times on which predictions can be done. Letting agents poll at different times and improving the assessors way of combining predictions could reduce this. Finally, the agents in this research only prefetch a single application per interval, resulting in at most one prefetch per 5 minutes per prediction agent. This could reduce freshness of a specific application if another was chosen as a better candidate for prefetch.

Other

Besides freshness, a user study in the wild has been conducted by PREPP. Here recall is reported to be about the same compared to this research. Precision and freshness are reported a lot better by PREPP though. This could be because of the same reasons as discussed above, but an important difference not mentioned by PREPP could be whether the homescreen application 'SpringBoard' is taken into account. This would improve freshness as this application is opened every time the phone is used. Also, the number of prefetches actually done could influence this. PREPP mentions such a measure very briefly in their paper. In section 9.3 this notion is used to estimate the interval between prefetches. This interval is notably higher than is calculated for the system of this research. As some presumptions had to be made in the calculations, these measures are uncertain. Provided they do hold though, it would mean that PREPP uses a lot more prefetches. The system of this research could already leverage this by prefetching more often, for instance by using option 2 of the protocol for consecutive launches in the assessment agent (section 8.3).

The fact that the number of prefetches can be of importance as suggested above, is thus important for the notion of freshness. This can be explained better using a small example. Say Facebook is opened by a user at time 60. Then a single prefetch at time 30 could result in a correct prefetch and a freshness of $60 - 30 = 30$. Now say the system would try a few more prefetches to improve freshness. It could prefetch again every 5 minutes for the next 15 minutes for example. Then the system prefetches Facebook at times 30, 35, 40 and 45. All of these prefetches are correct thus raising precision, the app Facebook was still predicted correctly thus recall remains the same and freshness will be reduced to $60 - 45 = 15$. The only way to detect this is by looking at the number of prefetches actually done, or by incorporating a measure like effectiveness as is done in this research. PREPP doesn't report on this, but only evaluates the battery and data costs. This research though isn't able to evaluate on those measures, thus making comparisons on these grounds difficult.

10.2 Implementation

Besides only discussing the results obtained, some words will be used to discuss the various techniques used in the implementation. The problems that were encountered in the design and implementation phase will be treated along with the way they were fixed.

10.2.1 Xamarin

The development environment chosen initially in the research was Xamarin. As Xamarin offers a convenient way to develop applications for multiple platforms. In order to use the existing Java libraries, Java bindings could be developed. Using these bindings, the .JAR files needed for communication by the application are wrapped in C# code. This way these files can be easily used in the .NET code used by Xamarin. Though the use of such a binding library would eventually be straightforward, the binding process itself was subject to numerous amounts of errors. As the automated binding isn't capable of converting everything successfully, many errors had to be removed manually. Because of the size of Jadex, the errors kept on coming. Eventually it was decided not to use Xamarin after all and implement using Eclipse. Since no deployments to actual devices were done this was no limitation.

Xamarin still seems like a good solution for cross platform development, but the use of external .JAR files is quite complex. Due to the fact that not everything can be bound to .NET code automatically, it seems like this solution is only feasible for simple and small classes of java code.

10.2.2 Jadex

The Jadex framework provides excellent support for the creation of BDI agents. The platform was quickly deployed on a windows machine and after the usual struggle with new techniques, the framework ran on Android as well. The agents created on either one of the machines easily work on the other as well. Using the BDI tutorial provided on the website of Jadex, an understanding of this framework and the possibilities is easily obtained.

In Jadex, BDI agents can be realized in two different ways. One is by using XML files that specify the agents and the other is by using java code in combination with annotations. In this research the combination with XML files was used as it was better documented and the java standalone files didn't work out of the box. As the main difference between the two is only ease of programming, the decision to use XML was quickly made.

Speed

When creating the agents, one downside that was encountered is the speed of Jadex. Considering predictions will normally be done throughout days or even weeks, this is not a problem. But when simulating these scenarios with timesteps of a single second, the agents were too slow. Each prediction agent has to perform a plan and a prediction every timestep. This went well with one prediction agent, but as more were added, the execution of all the plans took longer than a second, resulting in prediction agents that were running behind. Upon further inspection the reason for this time was not caused by heavy tasks of the agents, but by the fact the relay server was not properly contacted. This relay server gathers information on the use of Jadex. When it is not connected the speed of the framework is drastically reduced. Some networks had firewalls that blocked this connection and thus testing with the frameworks had to be done outside these networks.

10.2.3 Tropos

In order to design the agent configuration, some existing methodologies can be followed. This research uses Tropos. Tropos provides several design phases that provide guidance in the development of agent systems. During the use of Tropos though, it became clear that designing a system with a specific goal for a user is a little different than designing a 'silent' system that improves user experience. This resulted in very few requirements to be identified in the first phases of the methodology. Nevertheless the methodology provided a nice guideline in the design process in later stages. It encouraged to think better about the system to be developed.

10.2.4 Machine Learning

During the development of the system, the choice of machine learning was often reconsidered. Reinforcement learning was chosen initially, but raised some challenges during implementation. When learning what task is performed at what time, the environment is time. This means the environment is altered by an external process, so the state changes constantly. Since reinforcement learning is based on trial and error and works with delayed rewards, such a constant change of environment can interfere with the learning process of the agent.

For this reason supervised learning was reconsidered. In time-series predictions this type of learning is often applied. It can be used to learn which applications are often used in a certain state. A difficulty faced here though, is that the launch of a certain application isn't a fact (the next time it might not happen or at a different time), where supervised learning is used to classify data based on facts. Furthermore, as the user will change behavior over time, this data can alter a lot in the future.

As supervised learning isn't suitable, eventually reinforcement learning was successfully implemented nevertheless. This was done as follows. The agent keeps as a state the current time only. The actions it has to its availability are the act of prefetching an application. The environment then keeps track of the applications that are prefetched and receives the event of an application actually being launched. Every time the agent sends an action to the environment (what application to prefetch), the environment generates a reward based on whether this application was actually launched.

SARSA

At first SARSA was considered as a reinforcement learning algorithm. When predicting based on time though, it learned very slowly. For every state (time of day), only one action is tried at a time. Besides that, since the prefetches are on hold for several timesteps, a high reward might be received for a prefetch done a few timesteps back. This information has to propogate back for the model to learn where this high reward came from. That takes some time as well. In the end the SARSA algorithm was dismissed and replaced by the very similar Q-learning algorithm. This algorithm learns state-action values for all actions possible from a certain state at the same time. This way learning became considerably faster.

RL-Glue

The RL-Glue environment provides a framework for reinforcement learning. RL-Glue works as a static class which at first seems to allow for only a single instance. But it turns out it is possible to construct multiple interfaces for RL-Glue and swap the correct one in the static class when needed. This way multiple instances of RL-Glue could be used.

Unfortunately, this workaround proved not to be enough fairly quickly. Learning has to happen simultaneously, as many agent will be doing this at the same time. Via the workaround just discussed though, learning can only happen for one agent at a time. This eventually led to the only possibility which is adjusting the RL-Glue library to not work with a static, but a dynamic class. This way enabling it to work with multiple instances.

10.2.5 Testing

In order to simulate the user queries provided by the LiveLab dataset, the SQL statements in which they are delivered need to be processed. Microsoft's SQL Server is used to serve the data to the input agent. This input agent will query the SQL database and send the simulated data to the prediction agents.

One issue though is that the SQL files are far too big to be loaded into MS SQL server at once. Some even are over 1GB which results in the SQL Server Management Studio to give memory errors. There are multiple ways to work around this problem. The SQL files can be split and executed separately. A downside to this approach is that it takes a long time. Another way is to use SQLCMD, which is a command line approach of executing scripts on the server. This still resulted in insufficient system memory errors.

Eventually a combination of both solutions was used. The large SQL files were split using a small utility program to split large text files [26]. Then the smaller files were executed using sqlcmd. Query used: "sqlcmd -d livelab -U livelabuser -P livelabuser -i [sqlfilepath].sql".

SQL JDBC

In order to connect to the MS SQL Server from the Java environment, the Microsoft JDBC Driver was used [72]. This driver consists of an API that can be used to connect to the server and execute queries on it.

At first a new database connection was used every time a query was performed to the SQL server. This reconnecting eventually proved to cost a lot of time. Either retrieving all data before simulation or keeping the connection to the database open proved to be successful solutions.

Chapter 11

Conclusion

The goal of this thesis is to predict application launches on mobile devices. In order to reach this goal though, the problem was decomposed in some smaller goals in the introduction. Each of these smaller goals tried to answer a sub question of the overall research question. Now the conclusions on each question will be presented.

In order to use the flexible and expandable intelligent agents, a framework was needed and so the first question was posed:

SQ1: Which framework for intelligent agents can best be used?

To answer this question, multiple types of agents were considered. From these the most sophisticated type, namely the BDI agent, was chosen. They can be used as reactive as well as intelligent and complex agents, thus providing the flexibility for expansions. With the BDI property as one of the constraints, various agent frameworks were evaluated. Out of the two frameworks that provided everything required, Jadex was chosen. Now that the framework is determined, a way to learn user patterns was needed. The second question concerned this topic:

SQ2: What kind of machine learning should be used in the agents?

First the choice for a type of machine learning was settled at reinforcement learning. From the many algorithms considered, Q-learning was chosen mostly because of its popularity, proven effectiveness and qualities required for this research. This concludes the questions that make up the core of the system. This system though, will eventually be ported to a mobile device to work real time. For this purpose a third question was raised:

SQ3: Which mobile platform should be used?

The 3 most popular mobile platforms (Android, iOS and Windows Phone) were considered in this evaluation. Android came out on top because of its support for Java and the open nature that more easily permits some needed functions. Lastly the system has to be tested in order to evaluate it. The last question treats this aspect:

SQ4: Which way can the system be tested?

There are multiple ways to test the system. Existing datasets or real life testing were considered for this purpose. Eventually the LiveLab dataset was chosen to work with. Real life testing would take too much time and using existing datasets would allow for quick testing and adjusting. Furthermore this dataset was used by some related work, making comparisons more valuable.

Using these subquestions and the conclusions drawn from them, the research question was successfully investigated:

RQ: Can intelligent agents combined with machine learning be used to predict launches of applications for mobile devices?

The answer to this question is yes, the combination of both techniques was proven to successfully predict application launches of mobile phone users. With these predictions the freshness of the data and thus the recency of the data loaded for an application was reduced. This implies that applications should have a reduced load time, although this very fact should be verified using a deployment on a mobile device in future work.

Firstly reinforcement learning proved to be a good way to predict user behavior. When looking at a single prediction agent, it was clearly capable of predicting what a user would launch next. Besides machine learning, also the multi agent system has proven to work well in this domain. It is shown that the assessment agent is able to improve the results from each individual prediction agent. Considering all agents are similar in how they work, the framework promises even better results when more diverse predictors are used.

The comparison with related work was more difficult. Mostly because some decisions were not entirely documented in the papers and some evaluations on the cost of the system were done differently, more elaborately. As of now the systems PREPP and FALCON perform a bit better than the proposed system. This research simply hasn't had the chance yet to implement enough suitable predictors to match or exceed these results. Future work should clear this up.

For now, the conclusion remains that intelligent agents and machine learning form an excellent combination and are very capable of predicting user behavior on a mobile device. As there are numerous expansion possibilities possible, also in various other areas, such a system shows great potential for future work. These future possibilities are discussed in chapter 12.

Chapter 12

Future Work

The system as proposed in this thesis has various different expansion possibilities. The use of a multi agent system allows for many simple alterations and additions of agents to the system. Many expansions of the system have been discussed in section 6.3 already. These cover most improvements per agent. Besides these improvements, other types of machine learning could be used as well. Chapter 4 describes an elaborate research to machine learning and covers a range of different techniques that can be considered.

Besides trying different machine learning techniques, the BDI nature of the intelligent agents as discussed in chapter 3 can be leveraged more. Goals could be added regarding battery life constraints or data usage constraints. The intelligent agents using BDI are designed to work well with conflicting goals and thus can be used with these conflicts.

To test the system better, a deployment would be essential in future work. Right now the only evaluation that could be done was the performance in predicting applications, not the actual costs of the system. Besides implementing everything on the mobile device, an attempt can be made to implement only the input and launcher agent on the mobile device and offload the others to the cloud. The prediction and assessment agents use more system resources as they perform the actual machine learning. This way battery drain could be prevented in the mobile devices. Also, when deploying the application to a phone, a user study can be done to how users perceive the improvements.

Finally, as was treated in the beginning of this thesis, the techniques discussed here can be applied to other domains. One of these could be the caching of websites or databases. As these systems are often used by a lot of users, patterns could be extracted in what is requested from them. This could result in reducing the load times of these systems drastically as well. These caching techniques are one of the practical ways Avanade could use the devised system. The usability of this thesis for Avanade is discussed in Appendix C.

Appendix A

LiveLab Data

The LiveLab data consists of data from 34 users of an iPhone 3G from Texas, USA. The data was gathered from February 19th 2010 to April 25th 2011, varying per user. A summary of this dataset can be found in table A.1. The timestamps used in this table are UNIX timestamps. Furthermore, for the statistics in this table, the application 'SpringBoard' was excluded. That is because this is the homescreen of iPhone and therefore not considered a true application.

The query that was used to retrieve this table, is the following:

```
SELECT uid,COUNT(id),COUNT(DISTINCT(name)),MIN(time),MAX(time)
  FROM [livelab].[dbo].[appusage]
  WHERE name NOT LIKE 'SpringBoard'
  GROUP BY uid
  ORDER BY uid
```

User ID	App Launch Count	Distinct Apps	Start Timestamp	End Timestamp
A00	24275	89	1266247858	1298621859
A01	13335	161	1266250192	1297743157
A02	30698	149	1266250247	1296627649
A03	26162	318	1266261792	1298014999
A04	30421	102	1266249817	1298019952
A05	16618	184	1266250280	1299316704
A06	31784	80	1266259781	1298187902
A07	25947	216	1266250360	1301891155
A08	20057	46	1266259564	1303775269
A09	43230	341	1266250036	1290422489
A10	38382	96	1266250398	1297839233
A11	22306	91	1266250426	1298070169
A12	28270	184	1267226692	1296795933
B00	2798	40	1266250484	1291625624
B02	27768	75	1266250581	1297840148
B03	23015	57	1266250572	1298352938
B04	36366	129	1266250608	1301636623
B05	23680	316	1266249209	1297755014
B06	18519	65	1266250665	1297841129
B07	13768	86	1266250689	1297836173
B08	18777	535	1266250731	1297841777
B09	17165	59	1266250760	1300901448
B10	38178	85	1266250841	1293440978
B11	9255	58	1266241808	1298537849
D00	19666	131	1283903660	1298653610
D01	6746	50	1283892079	1287558610
D02	15467	247	1283900234	1298098265
D03	24613	69	1283892318	1298005996
D04	26245	158	1283896014	1297846359
D05	17040	134	1283900114	1296261516
D06	10696	134	1283892499	1297999149
D07	3992	85	1283892712	1297991678
D08	18724	136	1283892715	1297840797
D09	16709	51	1283891070	1296798471

Table A.1: Livelab: Summary of data

Appendix B

R Scripts

In order to transform and reorder all data gathered, R was used. As many scripts had to be used multiple times, this way data could be evaluated faster. This appendix lists the R scripts that were used in this research. First those used in chapter 8 on parameter setting are listed in section B.1. Then those for the evaluation of system performance as used in section 9.2 are shown in section B.2 and lastly those to compare to related work used in section 9.3 are listed in B.3.

B.1 Scripts for Parameter Setting

Chapter 8 discusses parameter setting. In it the performance measures for precision, recall and effectiveness are calculated as well as the average median freshness. The scripts used for it are shown in this section.

B.1.1 Precision, Recall and Effectiveness

The R script that is used to extract precision, recall and effectiveness. The values used to determine these performance measures, are retrieved from the last time an application was done by that user. The simulation actually went on till a fixed last timestamp and thus longer than most users were tracked. In the void the performance measures might drop because of it.

```
#Point to the folder containing relevant data for this experiment
last_folder <- "%current evaluation folder%/"
folder <- paste0("C:/%folder containing results%",last_folder)

#First read the data from CSV file for each user (uid)
rew_A08_assess <- read.csv(paste0(folder,"A08/assessment.csv"))
rew_A09_assess <- read.csv(paste0(folder,"A09/assessment.csv"))
rew_A10_assess <- read.csv(paste0(folder,"A10/assessment.csv"))
rew_B08_assess <- read.csv(paste0(folder,"B08/assessment.csv"))
rew_B10_assess <- read.csv(paste0(folder,"B10/assessment.csv"))

#Filter each dataset until the last timestamp an application was opened for it
rew_A08_assess <- rew_A08_assess[rew_A08_assess$time<1303775269,]
rew_A09_assess <- rew_A09_assess[rew_A09_assess$time<1290422489,]
rew_A10_assess <- rew_A10_assess[rew_A10_assess$time<1297839233,]
```

```
rew_B08_assess <- rew_B08_assess[rew_B08_assess$time<1297841777,]
rew_B10_assess <- rew_B10_assess[rew_B10_assess$time<1293440978,]

#Create a list for each performance measure
precision <- c("agent_id","avg_precision")
recall <- c("agent_id","avg_recall")
effectiveness <- c("agent_id","avg_effectiveness")

#Run through each agent
for(i in -1:7){
  #Append the last row to a list, this last row contains the average of the entire run
  agent <- tail(rew_A08_assess[rew_A08_assess$id==i,],n=1)
  agent <- rbind(agent, tail(rew_A09_assess[rew_A09_assess$id==i,],n=1))
  agent <- rbind(agent, tail(rew_A10_assess[rew_A10_assess$id==i,],n=1))
  agent <- rbind(agent, tail(rew_B08_assess[rew_B08_assess$id==i,],n=1))
  agent <- rbind(agent, tail(rew_B10_assess[rew_B10_assess$id==i,],n=1))

  #Calculate the preformance values for each agent and append it to a list
  precision <- rbind(precision,c(i,sum(agent$precision)/5))
  recall <- rbind(recall,c(i,sum(agent$recall)/5))
  agent$effectiveness <- agent$noFactsPredicted / agent$noPredictions
  effectiveness <- rbind(effectiveness,c(i,sum(agent$effectiveness)/5))
}

#Print the values for each performance measure
precision
recall
effectiveness
```

Calculate Precision, Recall and Effectiveness

B.1.2 Average Median Freshness

This script concerns the freshness values needed. It calculates the average median freshness out of all freshness values for each user.

```
#Point to the folder containing relevant data for this experiment
last_folder <- "%current evaluation folder%/"
folder <- paste0("C:/%folder containing results%",last_folder)

#First create a list to save values for agents in
list_median_sum <- c("agent_id","avg_medianfreshness")

#Run through each agent
for(i in -2:7){
  #Agent -2 represents the benchmark
  if(i==-2) i = "_benchmark"

  #Read the data from CSV file for every agent, add median freshness for all the
      different users
  median_sum_i <- median(read.csv(paste0(folder,"A08/freshness",i,".csv"))$freshness)
  median_sum_i <- median_sum_i + median(read.csv(paste0(folder,"A09/freshness",i,".csv")
      )$freshness)
  median_sum_i <- median_sum_i + median(read.csv(paste0(folder,"A10/freshness",i,".csv")
      )$freshness)
  median_sum_i <- median_sum_i + median(read.csv(paste0(folder,"B08/freshness",i,".csv")
      )$freshness)
  median_sum_i <- median_sum_i + median(read.csv(paste0(folder,"B10/freshness",i,".csv")
      )$freshness)

  #append calculation to list
  list_median_sum <- rbind(list_median_sum,c(i,(median_sum_i/5)))
}

#Print the list of average median freshnesses
list_median_sum
```

Average Median Freshness

B.2 Scripts for System Performance

To transform the data used for evaluating system performance in section 9.2, the following scripts were used. All scripts in this section have to connect to the MS SQL database[1] to retrieve data about the users at some point. This can be done using the following script and will thus not be repeated in each following section.

```
#add library needed for MS SQL connection to R
library("RODBC")

#Connect to the appropriate database
odbcChannel <- odbcConnect("LiveLabDatabase", uid="livelabuser", pwd="livelabuser")

#Execute query resulting in the user info needed
userInfo <- sqlQuery(odbcChannel,
                     "SELECT uid,COUNT(id) as 'ALC'
                     ,COUNT(DISTINCT(name)) as 'DA'
                     ,MIN(time) as 'ST'
                     ,MAX(time) as 'ET'
                     FROM appusage
                     WHERE name NOT LIKE 'SpringBoard'
                     AND uid NOT LIKE '%D%'
                     GROUP BY uid
                     ORDER BY uid")

#Close the connection afterwards
odbcClose(odbcChannel)
```

Connection to MS SQL Database

B.2.1 Precision, Recall and Effectiveness

In order to plot a graph for all performance measures for every agent, a script is needed to prepare this data. Eventually everything will be written to a new file and Excel will be used to plot the data. 2 scripts were used for this purpose, one that plots the accumulated performance measures and one that plots the performance measures per day.

The averages over all users have to be gathered even though they start using their phones at different times. To synchronize them, the starting times were extracted from the timestamps to let all users start at timestamp 0. The user that was tracked the shortest amount of time determines the total time evaluated.

[1]https://andersspur.wordpress.com/2013/11/26/connect-r-to-sql-server-2012-and-14/

Accumulated

```
#Point to the folder containing relevant data for this experiment
last_folder <- "%current evaluation folder%/"
folder <- paste0("C:/%folder containing results%",last_folder)

#Point to a folder to write transformed data to
folder_transformed <- "%new folder%"

#Now find the user with the shortest timespan, this will be the timespan used
shortest_time <- 9999999999
for(i in userInfo$uid){
  userInfo_i <- userInfo[userInfo$uid==i,]
  diff <- userInfo_i$ET - userInfo_i$ST
  if(diff < shortest_time) shortest_time <- diff
}

#Transform all data gathered by extracting start time from timestamps per user
all_assessments <- data.frame()
for(user_id in userInfo$uid){
  #Extract data belonging to the user
  userInfo_i <- userInfo[userInfo$uid==user_id,]
  data_assessment <- read.csv(paste0(folder,user_id,"/assessment.csv"))

  #Transform the timestamp
  data_assessment$timestamp <- data_assessment$timestamp - userInfo_i$ST

  #Write the transformed data to a new folder
  write.csv(data_assessment, paste0(folder_transformed,user_id,"/assessment.csv"))

  #save the data in a dataframe for future reference
  #this is done because reading a dataframe is considerably faster than reading a CSV
      file
  data_assessment$uid <- user_id
  all_assessments <- rbind(all_assessments, data_assessment)
}

#Prepare a dataframe to write output to
output <- data.frame(integer(0),integer(0), double(0), double(0), double(0))
output <- rbind(output, c(0,0,0.0,0.0,0.0))

#Run trough each agent
for(agent_id in -1:7){
  current_timestamp <- 86400

  #Run through each timestamp with intervals of a day
  while(current_timestamp < shortest_time){
    added_precision <- 0
    added_recall <- 0
    added_effectiveness <- 0
    count <- 0

    #Run through each user
    for(user_id in userInfo$uid){
      #Extract data belonging to the appropriate user
      curr_data <- all_assessments[all_assessments$uid==user_id,]

      #retrieve the last row within current timestamp
      curr_data <- curr_data[curr_data$time<current_timestamp,]
      last_row <- tail(curr_data[curr_data$id==agent_id,],n=1)

      #Calculate the preformance values for each agent and append it to a list
      added_precision <- added_precision + last_row$precision
      added_recall <- added_recall + last_row$recall
      added_effectiveness <- added_effectiveness + (last_row$noFactsPredicted / last_row
          $noPredictions)
      count <- count + 1
    }

    avg_precision <- 0
    avg_recall <- 0
    avg_effectiveness <- 0

    #Calculate the average performance measure for the agent in this timestamp
    if(count != 0){
      if(length(added_precision) > 0) avg_precision <- added_precision/count
      if(length(added_precision) > 0) avg_recall <- added_recall/count
      if(length(added_precision) > 0) avg_effectiveness <- added_effectiveness/count
    }

    #Write results to the output
    output <- rbind(output, c(agent_id,current_timestamp,avg_precision,avg_recall,avg_
        effectiveness))

    current_timestamp <- current_timestamp + 86400
  }
}

#Add column names to the output
colnames(output) <- c("agent_id","timestamp","avg_precision","avg_recall","avg_
    effectiveness")

#Write the output to a file to evaluate using Excel later
write.csv(output, paste0(folder_transformed,"/agents_averaged.csv"))
```

Accumulated Average Precision, Recall and Effectiveness

Daily

```
#Point to the folder containing relevant data for this experiment
last_folder <- "%current evaluation folder%/"
folder <- paste0("C:/%folder containing results%",last_folder)

#Point to a folder to write transformed data to
folder_transformed <- "%new folder%"

#Now find the user with the shortest timespan, this will be the timespan used
shortest_time <- 9999999999
for(i in userInfo$uid){
  userInfo_i <- userInfo[userInfo$uid==i,]
  diff <- userInfo_i$ET - userInfo_i$ST
  if(diff < shortest_time) shortest_time <- diff
}

#Transform all data gathered by extracting start time from timestamps per user
all_assessments <- data.frame()
for(user_id in userInfo$uid){
  #Extract data belonging to the user
  data_assessment <- read.csv(paste0(folder_transformed,user_id,"/assessment.csv"))

  #save the data in a dataframe for future reference
  #this is done because reading a dataframe is considerably faster than reading a CSV
      file
  data_assessment$uid <- user_id
  all_assessments <- rbind(all_assessments, data_assessment)
}
```

```

#Prepare a dataframe to write output to
output <- data.frame(integer(0),integer(0), double(0), double(0), double(0))
output <- rbind(output, c(0,0,0.0,0.0,0.0))

#Prepare datafames to save old values of start of week in
old_no_correctpredictions <- data.frame(0,0)
for(user_id in userInfo$uid){
  old_no_correctpredictions <- rbind(old_no_correctpredictions, c(user_id,0))
}
old_no_correctpredictions <- old_no_correctpredictions[-c(1), ]
old_no_predictions <- old_no_correctpredictions
old_no_correctlypredictedfacts <- old_no_correctpredictions
old_no_facts <- old_no_correctpredictions

#Run trough each agent
for(agent_id in -1:7){
  current_timestamp <- 86400

  #Run through each timestamp with intervals of a day
  while(current_timestamp < shortest_time){
    added_precision <- 0
    added_recall <- 0
    added_effectiveness <- 0
    count <- 0

    #Run through each user
    for(user_id in userInfo$uid){
      #Extract data belonging to the appropriate user
      curr_data <- all_assessments[all_assessments$uid==user_id,]

      #retrieve the last row within current timestamp
      curr_data <- curr_data[curr_data$time<current_timestamp,]
      last_row <- tail(curr_data[curr_data$id==agent_id,],n=1)

      if(nrow(last_row) > 0){
        #Add column names to dataframes to filter on them
        colnames(old_no_correctpredictions) <- c("uid","oldvalue")
        colnames(old_no_predictions) <- c("uid","oldvalue")
        colnames(old_no_correctlypredictedfacts) <- c("uid","oldvalue")
        colnames(old_no_facts) <- c("uid","oldvalue")

        #Calculate differences of each measure
        diff_correctpredictions <- last_row$noPredictionsCorrect - as.numeric(old_no_correctpredictions[old_no_correctpredictions$uid==user_id,]$oldvalue)
        diff_predictions <- last_row$noPredictions - as.numeric(old_no_predictions[old_no_predictions$uid==user_id,]$oldvalue)
        diff_correctlypredictedfacts <- last_row$noFactsPredicted - as.numeric(old_no_correctlypredictedfacts[old_no_correctlypredictedfacts$uid==user_id,]$oldvalue)
        diff_facts <- last_row$noFacts - as.numeric(old_no_facts[old_no_facts$uid==user_id,]$oldvalue)

        #Calculated the sum of the performance measures for this user
        if(diff_predictions > 0) added_precision <- added_precision + (diff_correctpredictions/diff_predictions)
        if(diff_facts > 0) added_recall <- added_recall + (diff_correctlypredictedfacts/diff_facts)
        if(diff_predictions > 0) added_effectiveness <- added_effectiveness + (diff_correctlypredictedfacts/diff_predictions)

        #Update the old values for the measures
        old_no_correctpredictions[old_no_correctpredictions$uid==user_id,]$oldvalue <- last_row$noPredictionsCorrect
        old_no_predictions[old_no_predictions$uid==user_id,]$oldvalue <- last_row$
            noPredictions
        old_no_correctlypredictedfacts[old_no_correctlypredictedfacts$uid==user_id,]$
            oldvalue <- last_row$noFactsPredicted
        old_no_facts[old_no_facts$uid==user_id,]$oldvalue <- last_row$noFacts
      }
      count <- count + 1
    }

    avg_precision <- 0
    avg_recall <- 0
    avg_effectiveness <- 0

    #Calculate the average daily performance measure for the agent in this timestamp
    if(count != 0){
      if(length(added_precision) > 0) avg_precision <- added_precision/count
      if(length(added_recall) > 0) avg_recall <- added_recall/count
      if(length(added_effectiveness) > 0) avg_effectiveness <- added_effectiveness/count
    }

    #Write results to the output
    output <- rbind(output, c(agent_id,current_timestamp,avg_precision,avg_recall,avg_
        effectiveness))

    current_timestamp <- current_timestamp + 86400
  }
}

#Add column names to the output
colnames(output) <- c("agent_id","timestamp","avg_precision","avg_recall","avg_
    effectiveness")

#Write the output to a file to evaluate using Excel later
write.csv(output, paste0(folder_transformed,"/agents_averaged(weekly).csv"))
```

Daily Average Precision, Recall and Effectiveness

B.2.2 Freshness

Data also needs to be prepared in order to plot the cumulative probability distribution of the freshness with. The transformation and plotting of this data is described by the following script.

```
#Point to the folder containing relevant data for this experiment
last_folder <- "%current evaluation folder%/"
folder <- paste0("C:/%folder containing results%",last_folder)

#write for every agents the freshness values added to a file
for(i in -2:7){
  #Agent -2 represents the benchmark
  if(i==-2) i = "_benchmark"

  #First read the data from CSV file for every agent, combine all the different users
  for(user_id in userInfo$uid){
    if(exists("freshness")) freshness <- rbind(freshness, read.csv(paste0(folder,user_id
        ,"/freshness",i,".csv")))
    else freshness <- read.csv(paste0(folder,user_id,"/freshness",i,".csv"))
  }

  #Now write the appended list of freshnesses to a file
  write.csv(freshness, file=paste0(folder,"addedfreshness",i,".csv"))
}

```

```
#For each agent, read the appriopriate data and plot the CPD
#Meanwhile names, colors and linetypes are used and saved in dataframes
added_freshness <- read.csv(paste0(folder,"addedfreshness_benchmark.csv"))
plot(ecdf(added_freshness$freshness), xlim=c(1,1e5),log='x',ylab="Cumulutative
    Probability", xlab="time (s)",main="Cumulutative Probability Freshness")
names <- c("benchmark")
colors <- c("black")
linetypes <- c(1)
for(i in 7:-1){
  if(i == -1) color <- "red"
  else color <- "grey"
  added_freshness <- read.csv(paste0(folder,"addedfreshness",i,".csv"))
  lines(ecdf(added_freshness$freshness), xlim=c(1,1e5),col.h=color, col.v=color)
  names <- c(names,paste0("agent ",i))
  colors <- c(colors,color)
  linetypes <- c(linetypes,1)
}

#Print a grid to read data easier
grid(nx = NULL, ny = NULL, col = "lightgray", lty = "dotted", lwd = par("lwd"), equilogs
     = TRUE)

#Print the legend of the graphs
legend(5,.9,names, col=colors, lty=linetypes)
```

Freshness CPD

B.3 Scripts for Comparison to Related Work

To compare the freshness values reported in PREPP using Facebook and Email only, a small adjustment has to be made to the aforementioned method to print the CPD of freshness. The script used to export this CPD for the applications Facebook and Email is given below.

```
#Point to the folder containing relevant data for this experiment
last_folder <- "%current evaluation folder%/"
folder <- paste0("C:/%folder containing results%",last_folder)

#retrieve the needed files on freshness
benchmark_data <- read.csv(paste0(folder,"addedfreshness_benchmark.csv"))
assessor_data <- read.csv(paste0(folder,"addedfreshness-1.csv"))

#First print values for facebook (benchmark and assessor)
facebook_bench <- benchmark_data[benchmark_data$query_name=="com.facebook.Facebook",]
plot(ecdf(facebook_bench$freshness), xlim=c(1,1e5),log='x',ylab="Cumulutative
    Probability", xlab="time (s)",main="Cumulutative Probability Freshness",col.h="
    darkblue",col.v="darkblue")

facebook_assessor <- assessor_data[assessor_data$query_name=="com.facebook.Facebook",]
lines(ecdf(facebook_assessor$freshness), xlim=c(1,1e5),col.h="blue",col.v="blue")

#Then print values for E-mail (benchmark and assessor)
email_bench <- benchmark_data[benchmark_data$query_name=="com.apple.mobilemail",]
lines(ecdf(email_bench$freshness), xlim=c(1,1e5),col.h="red",col.v="red")

email_assessor <- assessor_data[assessor_data$query_name=="com.apple.mobilemail",]
lines(ecdf(email_assessor$freshness), xlim=c(1,1e5),col.h="orange",col.v="orange")

#Finally print the gridlines and legend
axis(1,at=1000)
axis(1,at=10)
```

```
grid(nx = NULL, ny = NULL, col = "lightgray", lty = "dotted", lwd = par("lwd"), equilogs
     = TRUE)
legend(1,.9,c("Facebook benchmark","Facebook assessor","E-mail benchmark","E-mail
    assessor"), col=c("darkblue","blue","red","orange"), lty=c(1,1,1,1))
```

Freshness CPD per Application

Furthermore a script was used to extract the average times between two prefetches per application. This measure is used to guess the amount of prefetches done and compare it to PREPP. The retrieval of user information from the MS SQL database is needed as mentioned in section B.2.

```
#Point to the folder containing relevant data for this experiment
last_folder <- "%current evaluation folder%/"
folder <- paste0("C:/%folder containing results%",last_folder)

#Prepare variables to save data
sum_time_between_prefetches <- 0
count <- 0

#Transform all data gathered by extracting start time from timestamps per user
for(user_id in userInfo$uid){
  #Extract appropriate data for user
  userInfo_i <- userInfo[userInfo$uid==user_id,]
  data_assessment <- read.csv(paste0(folder,user_id,"/assessment.csv"))

  #Retrieve last row within last timestamp collected by user
  data_assessment <- data_assessment[data_assessment$time<userInfo_i$ET,]
  last_row <- tail(data_assessment[data_assessment$id==-1,],n=1)

  #Calculate the time interval this user covered
  timeinterval <- (userInfo_i$ET - userInfo_i$ST)

  #Calculate number of predictions that were done per application
  no_predictions_per_app <- (last_row$noPredictions / userInfo_i$DA)

  #Calculate sum of time elapsed on average between two predictions for an application
  sum_time_between_prefetches <- sum_time_between_prefetches + (timeinterval / no_
      predictions_per_app)
  count <- count + 1
}

#Calculate the average time between two prefetches per application
avg_time_between_prefetches <- sum_time_between_prefetches / count
```

Average Time between Prefetches per Application

Appendix C

Usability Avanade

This thesis was written at the IT consultancy company Avanade. Avanade is a company founded by Accenture and Microsoft. Because of this, the company focuses largely on Microsoft technologies and with it the .NET framework. This appendix will have a look at all techniques used in this thesis and how Avanade can possibly use these in practice. Lastly a section will be devoted to some applications where Avanade might be able to apply the system.

C.1 Agent Framework

This thesis uses a Multi Agent System as its core architecture. These agents can be seen as individual threads that take autonomous action and decide what to do themselves. They come in varying levels of intelligence, from reactive up until fully autonomous.

One of the best advantages of such a MAS is the flexibility it provides. When set up correctly, agents can easily deal with the addition or removal of other agents that provide other functionalities. Systems like yellow pages exist that provide the possibility to easily communicate between agents what functionalities they all provide. Furthermore they provide a good framework to set up concurrent, possibly distributed systems in.

In order to implement such a system, a multi agent framework has to be used. This thesis discusses a few (section 3.2), but they are all written in Java. The reason for this is that these techniques are mostly used in research and the research community often chooses Java for its platform independence. Since Avanade works mostly with .NET, this would either require porting a framework or using/expanding an existing .NET agent framework. These do exist, but they are rather limited and thus unpopular compared to the others. These limitations include the possibility to port agents to a mobile device (Windows Phone in this case) and the required BDI functionality. As BDI is not used to its full potential, this deficiency could be overcome.

A good option for Avanade would be to use existing .NET agent frameworks and work with the functions provided by it. The system as of now doesn't utilize the more profound possibilities, making this a viable option that requires a lot less work than porting code from one programming language to another. When necessary, the frameworks can be expanded as an asset of Avanade.

C.2 Reinforcement Learning

The technique used to learn the actual user patterns was Q-learning (section 4.2). This algorithm is a form of reinforcement learning which in its turn is a form of machine learning. Reinforcement learning is useful when only a performance indicator is available for the learning process, as opposed to a clear label (supervised) or nothing at all (unsupervised).

Again, Java was used to provide this functionality. The main reason for this was the choice for an agent framework in Java. The library used, RL-Glue (section 4.5), contains algorithms in C# as well though.

Recently, Microsoft released machine learning functionality within Azure, their cloud solution. Unfortunately, this library doesn't provide reinforcement learning. It is deemed interesting by Microsoft though, as it is mentioned as a natural fit for the Internet of Things applications[1]. This gives reason to believe it will be realized by Microsoft in the future, or otherwise Avanade could create an asset out of it. Reinforcement learning bridges the gap between supervised and unsupervised learning, thus being an interesting technique for domains that cannot use these. Furthermore, the creation of a reinforcement learning algorithm is rather simple and doesn't require expert knowledge of mathematics or statistics. When using a C# solution, Avanade could easily use reinforcement learning in their projects as well as the already available (un)supervised methods.

C.3 Deployment

The developed system can eventually be deployed to mobile devices in order to work in practice. Again since Java was needed by the agent framework, Android came out on top. Also when dropping the constraint of Java, Android proved to be the best option, but this time only very mild (section 5.1). When using an agent framework in .NET, the Windows Phone environment could be used.

Furthermore Xamarin, which is popular in Microsoft as well as Avanade, can be used to develop applications for all platforms in this case. The reason this failed in this research was due to the bindings needed for Java.

C.4 Evaluation

When evaluating the system, various Microsoft techniques were used that can directly be applied by Avanade as well. These include the use of R and Excel in preparing and plotting the data acquired. Also the Microsoft SQL Server was used to supply the test data consisting of mobile phone usage amongst various users.

[1] `azure.microsoft.com/en-gb/documentation/articles/machine-learning-algorithm-cheat-sheet/`

C.5 Applications

The system was developed to predict application launches on mobile devices. It was created with the goal to improve the user experience on such a device. To achieve this, predictions can be done about what users need in order to prepare the required data. This is a clear use case that can be applied to the creation of business apps within the mobile department of Avanade. It would speed up the use of data intensive applications, which are not uncommon in companies nowadays.

Furthermore the system can be applied to speed up access to databases. Following some interviews with Avanade experts on Sitecore, improvements in caching of databases can be achieved. The content of these caches nowadays is determined by the most recently or most often used data. The cache is filled completely after which the least important data is removed. This process can be made more intelligent. Based on past user queries, predictions can be made about what query the database will receive when. These queries can then be prepared in the form of a cache, creating a cache with more relevant content and thus allowing faster access to this data.

Appendix D

Code and Data Reference

The data and code of the project is publicly available on OneDrive via the following link:

`http://1drv.ms/1Mp1udq`

Via this link all the implemented code and the gathered data is accessible. This appendix will shortly discuss the contents of each of these folders.

D.1 Code

In order to run the code provided, Java SE 1.8 is needed. Besides that the frameworks that were used have to be installed. These include RL-Glue [116], Jadex, Microsoft SQL Server and the SQL JDBC driver [72] to connect to the server.

The code folder contains for each type of agent (”input”, ”prediction”, ”assessment” and ”launcher”) a folder with an XML file and several Java files. The XML files specifies the agents characteristics. Once started in Jadex, these files make calls to the appropriate Java files that provide the performed functions. Aside from these agent folders, a ”machinelearning” folder is added that contains the Java files required for the machine learning (only ”qlearning” as of now). Lastly the folder ”settings” contains Java files that provide some miscellaneous functionalities along with a file that specifies global settings of the system (Settings.java).

D.2 Data

The data folder contains three separate folders. The first is called ”Parameter Setting” and contains all data gathered to investigate the parameters in chapter 8. The folder called ”Results” contains all data gathered to investigate the results of the entire system in chapter 9. Finally the folder called ”R Scripts” contains all R scripts that were used. These will not be discussed as they are listed in appendix B. The contents of the other 2 folders will be explained in the following sections.

All folders in the lowest levels that contain data for a specific user have the same structure that will be explained first. They all contain text files (agent[i].txt) that specify for each agent the parameters used. Furthermore they contain a CSV file (assessment.csv) with information on the precision, recall and effectiveness per timestep per agent. Finally a list of CSV files (freshness[i].csv) contains for each separate agent the freshness values of the applications during runtime.

D.2.1 Parameter Setting

The data gathered for parameter settings is separated in data gathered for the prediction agent and the assessment agent. These contain folders for every separate parameter investigation. Within these folders resides the data gathered for every different user separated and one file of added freshness values of the users per agent. These last files are used to eventually create the Cumulutative Probability Distributions for the freshness performance indicator.

D.2.2 Results

The results folder firstly contains a folder with all raw data called "Raw Data Per User", where all data gathered directly from the program resides per user. This data is transformed so that all users start their usage cycles at time 0 and run for as long as the shortest user runs. The result of this transformation is stored in "Raw Transformed Data Per User".

This transformed data can be averaged among all users so that the average performance measures for each agent can be calculated. This averaged data resides in "Averaged Per Agent Data" and using the CSV files located there, the plots used in this research are made. They are present in the folder "Plots" as well.

As the freshness performance indicator is evaluated using a CPD, all freshness data has to be accumulated. This accumulation of usage data per agent is present in "Accumulated Freshness Per Prediction Agent" and is used to create the other plots that reside in "Plots" as well.

Bibliography

[1] 2APL. 2APL: A practical agent programming language. `http://apapl.sourceforge.net/`, 2015. Accessed: 10-06-2015.

[2] Hatem Abou-Zeid and Hossam S Hassanein. Toward green media delivery: location-aware opportunities and approaches. *Wireless Communications, IEEE*, 21(4):38–46, 2014.

[3] Pekka Abrahamsson. *Agile Software Development Methods: Review and Analysis (VTT publications)*. 2002.

[4] Jorge Agüero, Miguel Rebollo, Carlos Carrascosa, and Vicente Julián. Developing intelligent agents on the android platform. *Universidad Politecnica de Valencia, Spain*, 2010.

[5] Nadav Aharony, Wei Pan, Cory Ip, Inas Khayal, and Alex Pentland. Social fmri: Investigating and shaping social mechanisms in the real world. *Pervasive and Mobile Computing*, 7(6):643–659, 2011.

[6] AOS. AOS product website: JACK. `http://www.agent-software.com.au/products/jack/`, 2015. Accessed: 09-06-2015.

[7] Martin Atzmueller and Katy Hilgenberg. Towards capturing social interactions with sdcf: An extensible framework for mobile sensing and ubiquitous data collection. In *Proceedings of the 4th International Workshop on Modeling Social Media*, page 6. ACM, 2013.

[8] Avanade Inc. Avanade company website. `http://www.avanade.com/en-us/home`, 2015. Accessed: 12-06-2015.

[9] Mohammad Gheshlaghi Azar, Remi Munos, Mohammad Ghavamzadeh, Hilbert Kappen, et al. Speedy q-learning. *Advances in neural information processing systems*, 2011.

[10] Bala M Balachandran. Developing intelligent agent applications with jade and jess. In *Knowledge-based Intelligent Information and Engineering Systems*, pages 236–244. Springer, 2008.

[11] BDI4JADE. BDI4JADE project website. `http://www.inf.ufrgs.br/prosoft/bdi4jade/`, 2015. Accessed: 09-06-2015.

[12] Fabio Bellifemine, Giovanni Caire, Agostino Poggi, and Giovanni Rimassa. Jade: A software framework for developing multi-agent applications. lessons learned. *Information and Software Technology*, 50(1):10–21, 2008.

[13] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007.

[14] Hamid R Berenji and David Vengerov. Advantages of cooperation between reinforcement learning agents in difficult stochastic problems. In *Fuzzy Systems, 2000. FUZZ IEEE 2000. The Ninth IEEE International Conference on*, volume 2, pages 871–876. IEEE, 2000.

[15] Federico Bergenti, Giovanni Caire, and Danilo Gotta. Agents on the move: Jade for android devices. In *Procs. Workshop From Objects to Agents*, 2014.

[16] Ig Ibert Bittencourt, Pedro Bispo, Evandro Costa, João Pedro, Douglas Véras, Diego Dermeval, and Henrique Pacca. Modeling jade agents from gaia methodology under the perspective of semantic web. In *Enterprise Information Systems*, pages 780–789. Springer, 2009.

[17] Matthias Böhmer. Understanding and supporting mobile application usage. 2013.

[18] Matthias Böhmer, Brent Hecht, Johannes Schöning, Antonio Krüger, and Gernot Bauer. Falling asleep with angry birds, facebook and kindle: a large scale study on mobile application usage. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, pages 47–56. ACM, 2011.

[19] Rafael H Bordini, Lars Braubach, Mehdi Dastani, Amal El Fallah-Seghrouchni, Jorge J Gomez-Sanz, Joao Leite, Gregory MP O'Hare, Alexander Pokahr, and Alessandro Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica (Slovenia)*, 30(1):33–44, 2006.

[20] Rafael H Bordini, Mehdi Dastani, Jürgen Dix, and A El Fallah Seghrouchni. *Multi-Agent Programming.* Springer, 2005.

[21] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. John Wiley & Sons, 2007.

[22] Remco R Bouckaert, Eibe Frank, Mark Hall, Richard Kirkby, Peter Reutemann, Alex Seewald, and David Scuse. Weka manual for version 3-7-8, 2013.

[23] Remco R Bouckaert, Eibe Frank, Mark A Hall, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. Weka—experiences with a java open-source project. *The Journal of Machine Learning Research*, 11:2533–2541, 2010.

[24] Lars Braubach, Winfried Lamersdorf, and Alexander Pokahr. Jadex: Implementing a bdi-infrastructure for jade agents. 2003.

[25] Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. Jadex: A bdi-agent system combining middleware and reasoning. In *Software agent-based applications, platforms and development kits*, pages 143–168. Springer, 2005.

[26] Phil Campbell. A small utility app for splitting large text files. `http://www.devtxt.com/blog/large-file-splitter-console-app-utility`, 2015. Accessed: 09-06-2015.

[27] Maria Carpen Amarie, Ioannis Pefkianakis, and Henrik Lundgren. Mobile video ad caching on smartphones. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 57–61. ACM, 2014.

[28] Luca Cernuzzi, Ambra Molesini, Andrea Omicini, and Franco Zambonelli. Adaptable multi-agent systems: the case of the gaia methodology. *International Journal of Software Engineering and Knowledge Engineering*, 21(04):491–521, 2011.

[29] Supriyo Chakraborty, Kasturi Rangan Raghavan, Matthew P Johnson, and Mani B Srivastava. A framework for context-aware privacy of sensor data on mobile systems. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, page 11. ACM, 2013.

[30] David Chu, Aman Kansal, Jie Liu, and Feng Zhao. Mobile apps: its time to move up to condos. In *Proceedings of the 13th USENIX conference on Hot topics in operating systems*, pages 16–16. USENIX Association, 2011.

[31] Diane J Cook, Michael Youngblood, Edwin O Heierman III, Karthik Gopalratnam, Sira Rao, Andrey Litvin, and Farhan Khawaja. Mavhome: An agent-based smart home. In *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 521–521. IEEE Computer Society, 2003.

[32] Massimo Cossentino, Vincent Hilaire, Ambra Molesini, and Valeria Seidita. *Handbook on Agent-Oriented Design Processes.* Springer, 2014.

[33] Mehdi Dastani. 2apl: a practical agent programming language. *Autonomous agents and multi-agent systems*, 16(3):214–248, 2008.

[34] Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. In *AAAI/IAAI*, pages 761–768, 1998.

[35] Daniel C Dennett. *The intentional stance.* MIT press, 1989.

[36] Trinh Minh Tri Do, Jan Blom, and Daniel Gatica-Perez. Smartphone usage in the wild: a large-scale analysis of applications and context. In *Proceedings of the 13th international conference on multimodal interfaces*, pages 353–360. ACM, 2011.

[37] Jaschar Domann, Sindy Hartmann, Michael Burkhardt, Alexander Barge, and Sahin Albayrak. An agile method for multiagent software engineering. *Procedia Computer Science*, 32:928–934, 2014.

[38] Nathan Eagle and Alex Pentland. Reality mining: sensing complex social systems. *Personal and ubiquitous computing*, 10(4):255–268, 2006.

[39] Nathan Eagle and Alex Sandy Pentland. Eigenbehaviors: Identifying structure in routine. *Behavioral Ecology and Sociobiology*, 63(7):1057–1066, 2009.

[40] Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, and Deborah Estrin. Diversity in smartphone usage. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 179–194. ACM, 2010.

[41] Stefan Faußer and Friedhelm Schwenker. Ensemble methods for reinforcement learning with function approximation. In *Multiple Classifier Systems*, pages 56–65. Springer, 2011.

[42] Stephen Fitchett and Andy Cockburn. Accessrank: predicting what users will do next. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2239–2242. ACM, 2012.

[43] Funf. Funf in a box project website. `http://inabox.funf.org/`, 2015. Accessed: 09-06-2015.

[44] Kehan Gao, Taghi Khoshgoftaar, and Randall Wald. Combining feature selection and ensemble learning for software quality estimation. In *The Twenty-Seventh International Flairs Conference*, 2014.

[45] Mark H Goadrich and Michael P Rogers. Smart smartphone development: ios versus android. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 607–612. ACM, 2011.

[46] Jorge Gonzalez-Palacios and Michael Luck. Extending gaia with agent design and iterative development. In *Agent-Oriented Software Engineering VIII*, pages 16–30. Springer, 2008.

[47] John J Grefenstette, David E Moriarty, and Alan C Schultz. Evolutionary algorithms for reinforcement learning. *arXiv preprint arXiv:1106.0221*, 2011.

[48] Alejandro Guerra-Hernández, Amal El Fallah-Seghrouchni, and Henry Soldano. Learning in bdi multi-agent systems. In *Computational logic in multi-agent systems*, pages 218–233. Springer, 2005.

[49] Maozu Guo, Yang Liu, and Jacek Malec. A new q-learning algorithm based on the metropolis criterion. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(5):2140–2143, 2004.

[50] Jon C Hammer and Tingxin Yan. Exploiting usage statistics for energy-efficient logical status inference on mobile phones. In *Proceedings of the 2014 ACM International Symposium on Wearable Computers*, pages 35–42. ACM, 2014.

[51] Alina Hang, Alexander De Luca, Jonas Hartmann, and Heinrich Hussmann. Oh app, where art thou?: on app launching habits of smartphone users. In *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services*, pages 392–395. ACM, 2013.

[52] Nick Howden, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. Jack intelligent agents-summary of an agent infrastructure. In *5th International conference on autonomous agents*, 2001.

[53] Ke Huang, Chunhui Zhang, Xiaoxiao Ma, and Guanling Chen. Predicting mobile application usage using contextual information. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 1059–1065. ACM, 2012.

[54] IEEE. FIPA organization website. `http://www.fipa.org/`, 2015. Accessed: 09-06-2015.

[55] SM Jacob and Biju Issac. The mobile devices and its mobile learning usage analysis. *arXiv preprint arXiv:1410.4375*, 2014.

[56] Janus Core Developers. Janus project website. `http://www.janus-project.org/`, 2015. Accessed: 09-06-2015.

[57] Jason Developers. Jason, a Java-based interpreter for an extended version of AgentSpeak. `http://jason.sourceforge.net/wp/description/`, 2015. Accessed: 10-06-2015.

[58] Cheng Hao Jin, Gouchol Pok, Yongmi Lee, Hyun-Woo Park, Kwang Deuk Kim, Unil Yun, and Keun Ho Ryu. A som clustering pattern sequence-based next symbol prediction method for day-ahead direct electricity load and price forecasting. *Energy Conversion and Management*, 90:84–92, 2015.

[59] Hyojun Kim, Nitin Agrawal, and Cristian Ungureanu. Revisiting storage for smartphones. *ACM Transactions on Storage (TOS)*, 8(4):14, 2012.

[60] Emmanouil Koukoumidis, Dimitrios Lymberopoulos, Karin Strauss, Jie Liu, and Doug Burger. Pocket cloudlets. *ACM SIGPLAN Notices*, 47(4):171–184, 2012.

[61] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement Learning*, pages 45–73. Springer, 2012.

[62] Pier Luca Lanzi. Learning classifier systems: then and now. *Evolutionary Intelligence*, 1(1):63–82, 2008.

[63] Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.

[64] Sylvain Lemouzy, Valérie Camps, and Pierre Glize. Principles and properties of a mas learning algorithm: A comparison with standard learning algorithms applied to implicit feedback assessment. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on*, volume 2, pages 228–235. IEEE, 2011.

[65] Yang Li. Reflection: enabling event prediction as an on-device service for mobile interaction. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 689–698. ACM, 2014.

[66] Dimitrios Lymberopoulos, Oriana Riva, Karin Strauss, Akshay Mittal, and Alexandros Ntoulas. Pocketweb: instant web browsing for mobile devices. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 1–12. ACM, 2012.

[67] Hamid Reza Maei and Richard S Sutton. Gq (λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the Third Conference on Artificial General Intelligence*, volume 1, pages 91–96, 2010.

[68] Vukosi Ntsakisi Marivate and Michael L Littman. An ensemble of linearly combined reinforcement-learning agents. In *AAAI (Late-Breaking Developments)*, 2013.

[69] Catharine LR McGhan, Ali Nasir, and Ella Atkins. Human intent prediction using markov decision processes. In *Proc. Infotech@ Aerospace Conference*, 2012.

[70] Microsoft. Microsoft Azure machine learning website. `http://azure.microsoft.com/en-us/services/machine-learning/`, 2015. Accessed: 09-06-2015.

[71] Microsoft. Microsoft Office Excel. `https://products.office.com/nl-nl/excel`, 2015. Accessed: 09-06-2015.

[72] Microsoft. Microsoft SQL server: JDBC driver. `https://msdn.microsoft.com/en-us/sqlserver/aa937724.aspx?f=255&MSPPError=-2147217396`, 2015. Accessed: 09-06-2015.

[73] Microsoft. Microsoft windows dev center: Launching, resuming, and multitasking for Windows Phone 8. `https://msdn.microsoft.com/en-us/library/windows/apps/jj207014%28v=vs.105%29.aspx`, 2015. Accessed: 09-06-2015.

[74] Emiliano Miluzzo, Cory T Cornelius, Ashwin Ramaswamy, Tanzeem Choudhury, Zhigang Liu, and Andrew T Campbell. Darwin phones: the evolution of sensing and inference on mobile phones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 5–20. ACM, 2010.

[75] MIT. Reality mining dataset by MIT Human Dynamics Lab. `http://realitycommons.media.mit.edu/realitymining.html`, 2015. Accessed: 09-06-2015.

[76] H.B. Mitchell. Ensemble learning. In *Data Fusion: Concepts and Ideas*, pages 295–321. Springer Berlin Heidelberg, 2012.

[77] Prashanth Mohan, Suman Nath, and Oriana Riva. Prefetching mobile ads: Can advertising systems afford it? In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 267–280. ACM, 2013.

[78] Pavlos Moraïtis, Eleftheria Petraki, and Nikolaos I Spanoudakis. Engineering jade agents with the gaia methodology. In *Agent Technologies, Infrastructures, Tools, and Applications for e-Services*, pages 77–91. Springer, 2003.

[79] Pavlos Moraitis and N Spanoudakis. *Combining gaia and jade for multi-agent systems development*. 2004.

[80] Pavlos Moraitis and Nikolaos Spanoudakis. The gaia2jade process for multi-agent systems development. *Applied Artificial Intelligence*, 20(2-4):251–273, 2006.

[81] Arash Negahban and Chih-Hung Chung. Discovering determinants of users perception of mobile device functionality fit. *Computers in Human Behavior*, 35:75–84, 2014.

[82] David T Nguyen, Ge Peng, Daniel Graham, and Gang Zhou. Smartphone application launch with smarter scheduling. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 131–134. ACM, 2014.

[83] Jacob Nielsen. Nielsen Norman Group: Response times: the 3 important limits. `http://www.nngroup.com/articles/response-times-3-important-limits/`, 2015. Accessed: 12-06-2015.

[84] Ingrid Nunes. Improving the design and modularity of bdi agents with capability relationships. *EMAS 2014*, pages 58–80, 2014.

[85] Ingrid Nunes, Carlos JP De Lucena, and Michael Luck. Bdi4jade: a bdi layer on top of jade. *ProMAS 2011*, pages 88–103, 2011.

[86] Michael J OGrady and Gregory MP OHare. Mobile devices and intelligent agentstowards a new generation of applications and services. *Information Sciences*, 171(4):335–353, 2005.

[87] Antti Oulasvirta, Tye Rattenbury, Lingyi Ma, and Eeva Raita. Habits make smartphone use more pervasive. *Personal and Ubiquitous Computing*, 16(1):105–114, 2012.

[88] Lin Padgham and Dhirendra Singh. Situational preferences for bdi plans. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1013–1020. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[89] Lin Padgham and Michael Winikoff. *Developing intelligent agent systems: A practical guide*, volume 13. John Wiley & Sons, 2005.

[90] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[91] Abhinav Parate. Designing efficient and accurate behavior-aware mobile systems. 2014.

[92] Abhinav Parate, Matthias Böhmer, David Chu, Deepak Ganesan, and Benjamin M Marlin. Practical prediction and prefetch for faster access to applications on mobile phones. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 275–284. ACM, 2013.

[93] Ioannis Partalas, Grigorios Tsoumakas, and Ioannis Vlahavas. Pruning an ensemble of classifiers via reinforcement learning. *Neurocomputing*, 72(7):1900–1909, 2009.

[94] Veljko Pejovic and Mirco Musolesi. Anticipatory mobile computing: A survey of the state of the art and research challenges. *CoRR*, abs/1306.2356, 2013.

[95] Santi Phithakkitnukoon, Ram Dantu, Rob Claxton, and Nathan Eagle. Behavior-based adaptive call predictor. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 6(3):21, 2011.

[96] Alexander Pokahr, Lars Braubach, Christopher Haubeck, and Jan Ladiges. Programming bdi agents with pure java. In *Multiagent System Technologies*, pages 216–233. Springer, 2014.

[97] R Studio. R Studio: A powerful IDE for R. `http://www.rstudio.com/`, 2015. Accessed: 09-06-2015.

[98] Ahmad Rahmati. LiveLab dataset: Measuring wireless networks and smartphone users in the field. `http://livelab.recg.rice.edu/traces.html`, 2012. Accessed: 09-06-2015.

[99] CHCR Ribeiro. Reinforcement learning agents. *Artificial intelligence review*, 17(3):223–250, 2002.

[100] Sebastian Rodriguez, Nicolas Gaud, and Stephane Galland. Sarl: a general-purpose agent-oriented programming language. In *Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014 IEEE/WIC/ACM International Joint Conferences on*, volume 3, pages 103–110. IEEE, 2014.

[101] Sandia National Laboratories. Jess project website. `http://www.jessrules.com/`, 2015. Accessed: 09-06-2015.

[102] Suranga Seneviratne, Aruna Seneviratne, Prasant Mohapatra, and Anirban Mahanti. Predicting user traits from a snapshot of apps installed on a smartphone. *ACM SIGMOBILE Mobile Computing and Communications Review*, 18(2):1–8, 2014.

[103] Clayton Shepard, Ahmad Rahmati, Chad Tossell, Lin Zhong, and Phillip Kortum. Livelab: measuring wireless networks and smartphone users in the field. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):15–20, 2011.

[104] Kent Shi and Kamal Ali. Getjar mobile application recommendations with very sparse datasets. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 204–212. ACM, 2012.

[105] Choonsung Shin, Jin-Hyuk Hong, and Anind K Dey. Understanding and prediction of mobile application usage for smart phones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 173–182. ACM, 2012.

[106] Alberto Siena and Mirko Morandini. TAOM4E project website. `http://selab.fbk.eu/taom/`, 2015. Accessed: 09-06-2015.

[107] Dhirendra Singh, Sebastian Sardina, Lin Padgham, and Stéphane Airiau. Learning context conditions for bdi plan selection. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 325–332. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[108] Dhirendra Singh, Sebastian Sardina, Lin Padgham, and Geoff James. Integrating learning into a bdi agent for environments with changing dynamics. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 2525, 2011.

[109] Matthijs TJ Spaan. Partially observable markov decision processes. In *Reinforcement Learning*, pages 387–414. Springer, 2012.

[110] Vijay Srinivasan, Saeed Moghaddam, Abhishek Mukherji, Kiran K Rachuri, Chenren Xu, and Emmanuel Munguia Tapia. Mobileminer: Mining your frequent patterns on your phone. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 389–400. ACM, 2014.

[111] Janusz A Starzyk. Motivated learning for computational intelligence. *Computational Modeling and Simulation of Intellect: Current State and Future Perspectives*, pages 265–292, 2011.

[112] Janusz A Starzyk, James Graham, and Leszek Puzio. Simulation of a motivated learning agent. In *Artificial Intelligence Applications and Innovations*, pages 205–214. Springer, 2013.

[113] Jan Sudeikat, Lars Braubach, Alexander Pokahr, Winfried Lamersdorf, and Wolfgang Renz. Validation of bdi agents. In *Programming Multi-Agent Systems*, pages 185–200. Springer, 2007.

[114] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[115] Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, 2010.

[116] Brian Tanner and Adam White. RL-Glue : Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10:2133–2136, September 2009.

[117] Telecom Italia SpA. JADE project website. `http://jade.tilab.com/`, 2015. Accessed: 09-06-2015.

[118] The R Foundation. R project for statistical computing. `http://www.r-project.org/`, 2015. Accessed: 09-06-2015.

[119] Narseo Vallina-Rodriguez and Jon Crowcroft. Erdos: achieving energy savings in mobile os. In *Proceedings of the sixth international workshop on MobiArch*, pages 37–42. ACM, 2011.

[120] Hado Van Hasselt. Reinforcement learning in continuous state and action spaces. In *Reinforcement Learning*, pages 207–251. Springer, 2012.

[121] Nikos Vlassis, Mohammad Ghavamzadeh, Shie Mannor, and Pascal Poupart. Bayesian reinforcement learning. In *Reinforcement Learning*, pages 359–386. Springer, 2012.

[122] Pavel Vrba. Java-based agent platform evaluation. In *Holonic and Multi-Agent Systems for Manufacturing*, pages 47–58. Springer, 2003.

[123] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards.* PhD thesis, University of Cambridge, 1989.

[124] Shimon Whiteson. Evolutionary computation for reinforcement learning. In *Reinforcement Learning*, pages 325–355. Springer, 2012.

[125] Marco Wiering and Martijn van Otterlo. Reinforcement learning. *Adaptation, Learning, and Optimization*, 12, 2012.

[126] Marco A Wiering and Hado van Hasselt. Ensemble algorithms in reinforcement learning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 38(4):930–936, 2008.

[127] David H Wolpert and Kagan Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(02n03):265–279, 2001.

[128] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[129] Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(02):115–152, 1995.

[130] Michael Wooldridge, Nicholas R Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[131] Xamarin Inc. Xamarin developer guide: Background fetch (iOS 7 and greater). `http://developer.xamarin.com/guides/cross-platform/application_fundamentals/backgrounding/part_3_ios_backgrounding_techniques/updating_an_application_in_the_background/#background_fetch`, 2015. Accessed: 09-06-2015.

[132] Xamarin Inc. Xamarin developer guide: Java integration overview. `http://developer.xamarin.com/guides/android/advanced_topics/java_integration_overview/`, 2015. Accessed: 09-06-2015.

[133] Qiang Xu, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Jeffrey Pang, and Shobha Venkataraman. Identifying diverse usage behaviors of smartphone apps. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 329–344. ACM, 2011.

[134] Ye Xu, Mu Lin, Hong Lu, Giuseppe Cardone, Nicholas Lane, Zhenyu Chen, Andrew Campbell, and Tanzeem Choudhury. Preference, context and communities: A multi-faceted approach to predicting smartphone app usage patterns. In *Proceedings of the 17th annual international symposium on International symposium on wearable computers*, pages 69–76. ACM, 2013.

[135] Bo Yan and Guanling Chen. Appjoy: personalized mobile application discovery. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 113–126. ACM, 2011.

[136] Tingxin Yan, David Chu, Deepak Ganesan, Aman Kansal, and Jie Liu. Fast app launching for mobile devices using predictive user context. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 113–126. ACM, 2012.

[137] Franco Zambonelli, Nicholas R Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.