

Department of Computer Science  
University College London

# **Personalised Service Discovery in Mobile Environments**

Lucia Del Prete



Submitted in partial fulfilment of the requirements  
for the degree of Doctor of Philosophy  
at University College London

November 2011

I, Maria Lucia Del Prete confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Copyright © 2012 by Lucia Del Prete  
All rights reserved.

## Abstract

In recent years, some trends have emerged that pertain both to mobile devices and the Web. On one side, mobile devices have transitioned from being simple wireless phones to become ubiquitous Web-enabled users' companions. On the other side, the Web has evolved from an online one-size-fits-all collection of interlinked documents to become an open platform of personalised services and content.

It will not be long before these trends will converge and create a Seamless Web: an integrated environment where, besides traditional services delivered by powerful server machines accessible via wide area networks, new services and content will be offered by users to users via their portable devices. As a result, mobile users will soon be exposed – in addition to traditional “on-line” Web services/content – to a parallel universe of pervasive “off-line” services provided by devices in their surroundings.

Such circumstances will raise new challenges when it comes to selecting the services to rely on, that will require solutions grounded on the characteristics of mobile environments. Two aspects will require particular attention: first, users will have access to a countless multitude of services impossible to explore; they will need assistance to identify, among this multitude, those services they are most likely to enjoy. Secondly, if today's services (and their providers) are always-on, ‘static’ and aiming at Five 9s availability, tomorrow's pervasive services will be mobile (as devices move), fine-grained, increasingly composite (to provide richer functionalities) and so more unreliable by nature.

Our research tackles the problem of service discovery in pervasive environments in two ways: on one hand, we support personalised discovery by means of a mobile recommender system, easing the discovery of pervasive services appealing to end-users. On the other hand, we enable reliable discovery, by reasoning on the composite nature of pervasive services and the physical availability of their component providers. Overall, we provide a discovery method that enables ‘better’ pervasive services, where by ‘better’ we mean both ‘more interesting’ to the user and ‘more reliable’.

## Acknowledgements

First and foremost, I would like to thank Licia Capra, for caring so much about her students, for her patient guidance, encouragement and advice, and for constantly going the extra mile. Without her, I could not have completed this work.

I have been extremely lucky to have Licia as a supervisor. Her enthusiasm has been contagious and motivational, while her insight, knowledge, and humour has made my Ph.D. experience worthwhile, stimulating and, at the same time, fun.

I would also like to thank people at Deutsche Telekom (T-Mobile) and Tesco Dotcom, for their support. In particular, I am grateful to Mike Corrigall, who gave me the opportunity to start the Ph.D., and to JJ Van Oosten, for the visionary discussions and invaluable help.

Pursuing a Ph.D. is a quite a commitment; pursuing a part time Ph.D. – at the early stages of a career, and while in a long distance relationship – is a great *collective* commitment.

It requires support, understanding and endless patience from everyone someone really cares about: partner, family, and closest friends.

I am thankful to them all, for the opportunity to pursue this path, never feeling I had to compromise their love or friendship. Thanks for encouraging me when I lacked drive and passion; most of all, thanks for pretending interest in my research.

Above all, I am indebted to Claudio, my witty and patient partner throughout. Thanks for helping me finding the humour in things and playing them down. Thanks for always being by my side, closely but never intrusive, for the unconditional freedom and the unlimited support. Most of all, thanks for never asking me more than I could offer.

Last but not least, I would like to thank Raffaella, without whom I would have never started this journey. She persuaded me to pursue a Ph.D., as many years ago, she convinced me to start in-line skating. In both cases, I fell passionate about it. Thanks! Once again, I discovered something new that brings me joy and excitement.

*a Lorenzo,  
che ancor oggi è lo specchio a cui non so sfuggire*

*a Viola,  
perché da quando si è stretta a me viaggio a cuore aperto*

*a mia madre,  
che è la certezza d'amore con cui sono nata addosso*

# Contents

<b>I</b>	<b>Overview</b>	<b>14</b>
<b>1</b>	<b>Introduction</b>	<b>16</b>
1.1	Current Trends . . . . .	16
1.1.1	Mobile Devices . . . . .	16
1.1.2	The Web . . . . .	18
1.1.3	Seamless Web - A Vision . . . . .	21
1.2	Scenario . . . . .	23
1.2.1	Running Example . . . . .	23
1.2.2	Research Challenges . . . . .	27
1.3	Hypothesis and Contributions . . . . .	30
1.3.1	‘More Appealing’ Services . . . . .	31
1.3.2	‘More Reliable’ Services . . . . .	32
1.4	Assumptions . . . . .	33
1.5	Thesis Outline . . . . .	36
<b>II</b>	<b>Personalised Discovery</b>	<b>37</b>
<b>2</b>	<b>Personalised Discovery of Pervasive Services</b>	<b>39</b>
2.1	Background . . . . .	41

2.1.1	Content-based Filtering . . . . .	43
2.1.2	Collaborative Filtering . . . . .	44
2.1.3	Hybrid methods . . . . .	50
2.2	Challenges . . . . .	52
2.3	Conceptual Model . . . . .	54
2.3.1	Core Recommender Service: diffeRS . . . . .	54
2.3.2	Compositional Recommender Service: CReSy . . . . .	62
2.4	Concrete model . . . . .	66
2.4.1	diffeRS . . . . .	67
2.4.2	CReSy . . . . .	71
2.5	Related Work . . . . .	74
2.6	Summary . . . . .	79
<b>3</b>	<b>Evaluation of Personalised Discovery</b>	<b>80</b>
3.1	Background. Evaluating Recommender Systems . . . . .	80
3.1.1	Accuracy Metrics . . . . .	81
3.1.2	Beyond Accuracy . . . . .	83
3.2	diffeRS Evaluation . . . . .	84
3.2.1	Performance . . . . .	84
3.2.2	Overhead . . . . .	96
3.3	CReSy Evaluation . . . . .	100
3.3.1	Performance . . . . .	100
3.3.2	Overhead . . . . .	104
3.4	Summary . . . . .	105

<b>III</b>	<b>Reliable Discovery</b>	<b>107</b>
<b>4</b>	<b>Reliable Discovery of Pervasive Services</b>	<b>109</b>
4.1	Background . . . . .	110
4.1.1	Service Discovery . . . . .	110
4.1.2	Context-Aware Services and Adaptation . . . . .	113
4.1.3	Service Composition and Orchestration . . . . .	115
4.1.4	Seamless Session Management . . . . .	117
4.2	Challenges . . . . .	118
4.3	Conceptual Model . . . . .	122
4.3.1	The Service Entity . . . . .	124
4.3.2	The Binding Entity . . . . .	127
4.4	Concrete model . . . . .	127
4.4.1	Mobility Predictor . . . . .	128
4.4.2	Semantic Reasoner . . . . .	129
4.4.3	Adaptive Binder . . . . .	133
4.5	Summary . . . . .	135
<b>5</b>	<b>Evaluation of Reliable Discovery</b>	<b>136</b>
5.1	Mobility Predictor . . . . .	136
5.1.1	Performance . . . . .	136
5.1.2	Overhead . . . . .	140
5.2	Semantic Reasoner . . . . .	144
5.2.1	Performance . . . . .	144
5.2.2	Overhead . . . . .	151
5.3	Adaptive Binder . . . . .	152

5.3.1	Performance . . . . .	152
5.3.2	Overhead . . . . .	160
5.4	Summary . . . . .	161
<b>IV</b>	<b>Framework and Future Work</b>	<b>163</b>
<b>6</b>	<b>Middleware Realisation</b>	<b>165</b>
6.1	Architecture . . . . .	165
6.1.1	The Service Manager . . . . .	168
6.1.2	The Service Analyser . . . . .	169
6.1.3	The Service Discoverer . . . . .	170
6.1.4	The Service Coordinator . . . . .	171
6.2	The Service Discoverer and the Discovery Pipeline . . . . .	171
6.2.1	Service Filters and Provider Filters . . . . .	172
6.2.2	Service Discovery . . . . .	172
6.3	Implementation . . . . .	174
6.4	Deployment . . . . .	175
6.5	Programmatic Complexity . . . . .	177
6.5.1	Target Service Request . . . . .	177
6.5.2	Discovery Service Request . . . . .	179
6.5.3	Advertising a service . . . . .	180
6.6	Summary . . . . .	181
<b>7</b>	<b>Conclusions and Future Work</b>	<b>182</b>
7.1	Contributions . . . . .	182
7.2	Critical Evaluation and Future Work . . . . .	185



# List of Figures

1.1	Paradigm shift in Service Delivery . . . . .	23
1.2	LG ePaper Display . . . . .	24
1.3	Liquavista ePaper Tablet . . . . .	24
1.4	Filter Chain . . . . .	31
1.5	Approach . . . . .	31
1.6	Assumptions . . . . .	34
2.1	Recommenders - Taxonomy . . . . .	42
2.2	moRe Overview . . . . .	54
2.3	diffeRS Rating Matrix Decomposition . . . . .	57
2.4	CReSy Conceptual Framework . . . . .	66
2.5	Rating Hash Tree . . . . .	68
2.6	Execution of the Recommending Algorithm. . . . .	72
2.7	Preference Prediction for Composite Items . . . . .	73
2.8	Preference Prediction for Elementary Items . . . . .	74
3.1	Accuracy for All Users (MovieLens) . . . . .	90
3.2	Accuracy by Cluster (MovieLens) . . . . .	91
3.3	Accuracy for All Users (Yelp) . . . . .	91
3.4	Accuracy by Cluster (Yelp ) . . . . .	92

3.5	Most Dynamic Months - Accuracy (MovieLens+MIT Reality) . . . . .	93
3.6	Most Dynamic Months - Coverage (MovieLens+MIT Reality) . . . . .	93
3.7	Representative Months - Accuracy (MovieLens+MIT Reality) . . . . .	94
3.8	Representative Months - Coverage (MovieLens+MIT Reality) . . . . .	94
3.9	Accuracy (Yelp + Cabspotting) . . . . .	95
3.10	Coverage (Yelp + Cabspotting) . . . . .	95
3.11	Album (Compositions) Predictions . . . . .	103
3.12	Track (Component) Predictions . . . . .	104
4.1	Scenario . . . . .	119
4.2	Service Entity . . . . .	124
4.3	Example of a MoSCA Composite Service . . . . .	126
4.4	Binding Entity . . . . .	127
4.5	Example of MoSCA Bindings . . . . .	128
5.1	Mobility Predictor - Correct Predictions . . . . .	140
5.2	Mobility Predictor - Underestimation . . . . .	140
5.3	Semantic Reasoner - Successfully Completed Compositions . . . . .	147
5.4	Semantic Reasoner - Missed Opportunities . . . . .	147
5.5	Semantic Reasoner - Success rates for Sequence Compositions . . . . .	149
5.6	Semantic Reasoner - Success rates for Parallel Compositions . . . . .	149
5.7	Semantic Reasoner - Success rates for Any Order Compositions . . . . .	150
5.8	Semantic Reasoner - Success rates for Choice Compositions . . . . .	150
5.9	Semantic Reasoner - Missed Compositions . . . . .	151
5.10	Gatwick Airport South Terminal, Airside . . . . .	154
5.11	Time-to-disappear . . . . .	155

5.12 Adaptive Binder - Success rates . . . . .	157
5.13 Predicting changes to the Environment - Time-to-disappear . . . . .	158
5.14 Predicting changes to the Environment - Time-to-disappear varying on alpha	159
6.1 Token Passing Examples . . . . .	166
6.2 MoSCA Overview . . . . .	167
6.3 MoSCA Components . . . . .	168
6.4 Discovery as Filtering/Ranking . . . . .	172
6.5 Bindings Conceptual Model . . . . .	174
6.6 MoSCA deployed as a Library . . . . .	175
6.7 MoSCA deployed as a Server and as a Stub . . . . .	176

# List of Tables

5.1	Mobility Prediction. Success Rates . . . . .	139
5.2	Mobility Prediction. Success Rates by minimum familiarity level . . . . .	141
5.3	Mobility Prediction. Underestimation Rates by minimum familiarity level .	141
5.4	Percentage of successfully completed compositions among the ones started .	148

## Part I

# Overview



# Chapter 1

## Introduction

In recent years, some radical trends have emerged that pertain both to mobile devices and the Web. On one side, mobile devices have transitioned from being simply wireless phones, available to just a few people, to become ubiquitous and Web-enabled users' companions. On the other side, the Web has been evolving from an online one-size-fits-all collection of interlinked documents, published exclusively by companies or organisations, to become an open platform of content and services, often personalised to its consumers, who are often shapers of what the Web has to offer.

In the following, we briefly discuss each of these trends before introducing the future we envision and the challenges it will raise.

### 1.1 Current Trends

#### 1.1.1 Mobile Devices

In the late '90s/early 2000s, mobile devices were fundamentally wireless phones owned by a very restricted group of people. They were enriched with basic messaging services (e.g. SMS) and, when provided access to the Web, they were accessing a secluded and downgraded version of it: a sort of mini-Web, collectively delivered by isolated walled-garden portals, mainly controlled by Telecom Operators.

In the last decade, mobile devices and their adoption have changed remarkably. Today, in fact:

- mobile devices have evolved to smart-devices, enriched with functionalities of any sort;
- mobile devices have emerged as Internet appliances running full-fledged Web Browsers,

able to access the same Web of desktop computers;

- mobile devices have become increasingly ubiquitous and increasingly personal.

### **Smart Devices.**

In the latest years, mobile phones (e.g., iPhone, WebOS, Android-powered mobile phones) have seen their computing capabilities (e.g., processing power and memory availability) grow according to Moore's law. Increasingly faster and increasingly rich in resources, they have been enriched with new functionalities that extend beyond their original purpose. Whilst initially they were integrating mainly new hardware functionalities like digital cameras, MP3 players and GPS receivers, more recently, they have been augmented with a flourishing number of applications specifically developed for their platforms (e.g., journey planners, dictionaries, navigation systems, mobile bloggers, etc.). Also, they have become increasingly connected: both to other devices in the surrounding, as well as to the Internet. In fact, modern mobile devices have been enriched with wireless network technologies of increasing bandwidth, both for Local Area Networks (e.g., Bluetooth 2, Wi-Fi) and Wide and metropolitan Area Networks (e.g., 3G, HSDPA, WiMax, LTE).

New mobile devices thus enable users to create content and to run a variety of applications. In a not so far future, these same content and applications could be exposed as services to services (and applications) running on other terminals in the surroundings.

### **Web Devices.**

In a not so distant past, mobile phones had their own Web of mobile web pages (WML pages [WAP Forum, 1999]) and their own protocol for accessing it: the Wireless Application Protocol (WAP) [Alliance, 2001]. The main reasons for such a constrained and limited version of the Web were quite straightforward: mobile devices had very limited network connectivity with very limited bandwidth, they had no ability to process “tag-soup” HTML and they did not have enough memory to load pages. With the great increase of network bandwidth, computing resources and memory, the need for having a separate and “lesser Web” has started to fade away. Hence, mobile phones started to have proper HTML browsers, accessing to the same Web pages the desktop PC were accessing. That was the time the Access NetFront and Opera browsers emerged. They could not support all technologies and all mark-up quirks, but they could do a good enough job for mobile users to start accessing the standard Web.

At the same time, wireless network technologies of increasing bandwidth and, sometime, of local reach emerged (e.g., Bluetooth 2, Wi-Fi and WiMax, 3G), thus allowing, on one side, the on-the-fly creation of networks that connect devices in proximity and, on the other

side, access to the full Web. So, full Web-enabled mobile browsers (e.g., WebKit based browsers) have started to appear and mobile devices have risen as a prominent channel for the Internet future growth.

We foresee mobile devices to be increasingly plugged into the Web paradigm. Whilst initially this could simply entail mobile devices to run Web applications locally, as if they were native applications, in the long term we predict these same applications to be exposed as services (or server applications) for others devices to consume. Hence, mobile devices and services will become an integral part of the Internet cloud.

### **Ubiquitous Devices.**

In recent years the number of portable terminals, devices embedded within our physical environments (including sensors and actuators) and, in general, digital appliances has dramatically increased. Also, new types of post-PC devices have emerged (e.g., eReaders, tablets, digital media players, portable consoles). On one side, advances in technology and popularity have often translated into cheaper costs for entry level devices, thus enabling to produce more at a lower cost, leveraging on economies of scale. On the other side, these advances have been well received by end-users, who have started looking at their smart-phones and tablets as essential companions to attain their daily tasks, thus fostering further innovation, improvements, popularity and ultimately feeding a virtuous cycle. The result is that, not only mobile devices have become more powerful, richer in terms of applications and functionalities, but also more personal and more ubiquitous, to the point that mobile phones alone (i.e., excluding tablets and the like) have reached a market penetration second to no other digital device.

We predict this trend to continue and mobile and embedded devices to become predominant in people's lives and on the Web.

#### **1.1.2 The Web**

In parallel to these trends, we have been witnessing fundamental changes to the Web and its very nature. In fact, in the '90s, the Web was fundamentally an online collection of documents linked to each other; few editors were controlling its content, whilst a multitude was just passively interacting with it. Web pages and applications were the same for all users, irrespective of their interests or history or demographic. Today, the Web has evolved to a richer and more sophisticated world:

- The Web has evolved to a development platform of re-usable services and content;

- Web users have become active shapers of what the Web has to offer;
- The Web has become increasingly personalised to each individual user;
- The Web is no longer just “online”.

### **Mash-up Web.**

Since its first appearance the Web has dramatically changed and it is progressively becoming an open service-oriented platform of reusable and composable services. Organisations and companies have started to expose their own data and capabilities as data feeds, self-standing services and widgets (a.k.a. UI services, web parts, portlets). The advent of standards, or de-facto standards, as XML [Bray et al., 1998], SOAP [Box et al., 2000], RSS [Libby, 1999], ATOM [Nottingham and Sayre, 2005], WSDL [Christensen et al., 2001], BPEL [Coalition, 2003] and more recently WSRP [Kropp et al., 2003], REST [Fielding, 2000], JSON [Crockford, 2006], JSONP [Ippolito, 2005], WADL [Hadley, 2009] and OpenSearch [Clinton, 2009] has allowed – over the years – to evolve what were initially just isolated initiatives to common practices.

The promise of an intelligent semantic Web and of a Global Giant Graph [Berners-Lee, 2007] (as opposed to a World Wide Web) will push the Web transformation even further, as meaningful data and relationships among them will build the core of an automated Web that “reads, writes, executes” [Berners-Lee et al., 2001].

Although the semantic Web may be a long way away, standards such as RDF [Lassila and Swick, 1999], RDFa [Birbeck and Adida, 2008], OWL [Schreiber and Dean, 2004], OWL-S [Martin et al., 2004] are paving the way for it to happen. On a more pragmatic side, standards like HTML5 [Hickson, 2011], WAI ARIA [Cooper and Craig, 2009], new content de-facto standards like micro formats [Allsopp, 2007] and, partly, collaborative initiatives, like folksonomies, are building means to make Web content more widely understandable. The Web is thus evolving to an open service-oriented platform, where data, as well as services, will become core components for everyone and everything (e.g., refrigerators, smart cupboards, eBooks, mobile devices, etc.) to use, mix and match; thus enabling richer and more sophisticated (composite) services.

We foresee the same paradigm (i.e., service-oriented architectures) to extend to any device whose content, information or functionalities could be exposed for others to re-use.

### **User generated Web.**

When we look at the content the Web has to offer, we cannot avoid noticing that, in the recent past, a raising number of blogs, micro blogs and personal content spaces have been flourishing. Internet users are no longer just consumers, but have started to become active

actors in shaping what the Web has to provide.

Today, users have surpassed the boundaries of their personal blogs to become active providers of what other sites have to offer, sometime becoming the real proposition of some online businesses (e.g., Facebook, Twitter, etc.) and/or intelligence behind them (e.g., Flickr).

In fact, User Generated Content (UGC) has become increasingly popular (and hence profitable) on the Web, to the point that users' feedback, users' ratings, users' uploaded pictures, videos, recipes, as well as users' relationships (or social networks) have become standard features on most popular websites.

In the future, we expect users continuing sharing content and experience for the benefit of the mass, and possibly do it even at a larger extent – e.g., by exposing their mobile device services and functionalities to others.

### **Personalised Web.**

Nowadays web users are exposed to an almost endless universe of content, information and applications. Some of these applications have become essential tools in users' daily routines (e.g. emails), but the majority of them remains unknown to most people.

Personalisation techniques have emerged in the recent past both (i) to position at 1-click away the applications users frequently employ, and (ii) to assist web users discovering content, information, products and services of interest to them (as individuals), out of an unknown multitude of options.

Explicit personalisation techniques – where users actively personalise applications to match their interest – have increasingly been gaining popularity on the Web, as shown by the variety of mash up applications existing today (personalized on-line and off-line portals, widgets, etc.). These personalised applications have allowed users to explicitly select and combine, in one easily accessible space, the data and the services available on the Web that they find most relevant.

It is though with implicit personalisation techniques – for which the users' activity is monitored and then exploited to guess individual interests – that personalisation has known its greater and wider success, as the proliferation of recommendations in current websites is testament of. Recommendation services have, in fact, allowed users to discover appealing content and services, among a multitude of alternatives impossible to explore.

In the future, with the increasing number of services and information users will be exposed to, we expect recommendation services to become more and more popular and to be employed at a larger extent in any possible domain. For instance they could be used (beside explicit personalisation) to populate the personal portals in use today.

## Off-line Web.

Arguably the most interesting Web change we have witnessed in the last decade relates to its transformation from “on-line” to “off-line” nature. With the advent of client scripting, browser local cache and more recently HTML5 [Hickson, 2011], Web Applications have become increasingly rich, responsive and capable of running off-line, to the point that today modern Web Applications can rely on network connectivity just for their distribution or calls to cloud services.

First, the rise of ECMAScript [International, 1999] and its dialects (e.g., JavaScript, ActionScript and JScript) has enabled to delegate the execution of some of the Web Application logic to the Web Browser itself. Later, the advent of HTTP 1.1 [Fielding et al., 1999] and Cache Control has allowed to cache web resources within the browser, thus allowing to re-access them even when disconnected. More recently, the introduction of XMLHttpRequest, and then of AJAX and JSON [Crockford, 2006], has enabled to create rich and interactive web applications, thus fostering much of the research and innovation being consolidated within the HTML5 standard. It is though just with the emerging of HTML5 technologies, such as offline detection, cache application manifest, local web storage, web workers, etc., that “off-line” Web applications have started to appear. Web applications have so become increasingly richer, sophisticated and better integrated within the devices they run on.

We foresee that in the future, little – if any – difference will exist between Native and Web Apps in terms of performance, efficiency and integration to the client device. Web applications will be embedded within the device OS and, possibly, even become an integral part of the same OS. Devices will become more and more capable of “running” Web Apps and technologies, not just consuming them. Hence, it is not impossible to think that “client” devices (e.g., laptops, PCs, mobile phones, etc.) will become themselves de-facto “web servers”. The current classification of “client” and “server” will become less obvious and, to some extent, obsolete. The Web will start “running” on consumer devices, beside traditional servers.

### 1.1.3 Seamless Web - A Vision

We believe it will not be long before the two main trends above (evolution of mobile devices and the Web) will converge to create a Seamless Web: an integrated environment where, besides traditional services delivered by powerful server machines accessible via wide area networks, new services and content will be offered by users to users via their portable devices. The advent of an Internet of Things [Gershenfeld et al., 2004], and of an intelligent and automated Semantic Web will foster this convergence. Mobile applications and content will be exposed as services for other services and applications, running on the devices in the surroundings, to use and combine. Services could include, for example,

podcast channels, route finders, virtual city guides, traffic updates, and the like; data would range from information about the surroundings (e.g., local amenities, shops, gigs, events) to information that co-located users generate themselves and are willing to share (e.g., microblogs, pictures, videos, etc.), as well as their combinations. As a result - in addition to 'on-line' services/content available on the online global Web - mobile users will soon be exposed to a universe of pervasive 'off-line' services, provided by devices in the surroundings. These services will be combined with traditional (on-line and global) services to craft richer and more sophisticated applications. The concept of an 'on-line' Web will be forever changed.

The Seamless Web will bring an ever-increasing number of *meaningful* self-descriptive services and information. These will be composed and aggregated, in an automated way, to form sophisticated services, content and applications. We predict mobile services and data served by mobile devices to become part of this ever-growing semantic Web and to be exposed and composed through the same means.

If today mobile platforms, like Android, allow creating and exposing data services to applications running on the same mobile phone, we expect in the future this same paradigm to evolve to make services and data available outside the boundaries of the handset. Devices may, in fact, have core or added functionalities exposed as services.

Similarly, if currently mobile platforms like Nokia Ovi or WebOS or JIL/WAC [ASL, 2011] environments, and any other HTML5 compliant framework, run web applications locally (on the device web runtime) for the benefit of their users, in future, we envisage the same web applications or their underpinning services to be exposed to other devices.

Also, as nowadays people are generous in creating content for everyone to use, and in sharing their experience (e.g., on movies, albums, songs, hotels, restaurants and products or services of any sort), so we believe they will be generous in sharing functionalities and content hosted in their devices. For instance, a user may make available her navigation system to other users; or she may let other people exploit her device fast connection.

In this kind of context, it is not difficult to imagine an integrated environment where both services accessible within the surrounding local networks and wide area networks will be available to the consumer, transparently, through the same mechanisms and will be composed to deliver complex functionalities. Personal assistants, mobile phones, tablets, eBook readers, multimedia players, hand-held devices and embedded machines will not just be service consumers, but service providers too and will make usage of the machines in their surroundings. A mobile phone may, for instance, delegate part of its processing to a PC in the proximity; a 3G hand-held device may exploit LTE (or later generation) mobile terminals connected to it by WiFi to improve the bandwidth it accesses the Internet with.

In this scenario, beside just monolithic complex services delivered by highly powerful server machines, a universe of new types of services will be openly available to application designers and developers: fine grained services, supplied by very constraint devices, that will need to be composed to enable more sophisticated functionalities. These services will

be often mobile in relation to the user (i.e., the service host is mobile, or the host is fix in a location, but it is accessible just within a local area network and the consumer moves in and out of such area).

Web services, as we know them today, will be enriched with pervasive services of *local* nature. They will be *local* as they will be useful only to people in the surroundings and they will hence be made available only to devices in their proximity.

In summary, beside the ever growing Web *global macro-cosmos*, we envision an **ubiquitous** multitude of *local micro-cosmoses*, bringing to users an endless universe of pervasive content and services of **local** nature. These services will be offered by devices embedded in the surroundings or by users to users while on the go, via their portable devices, and they will need to be composed to obtain richer and compelling services.

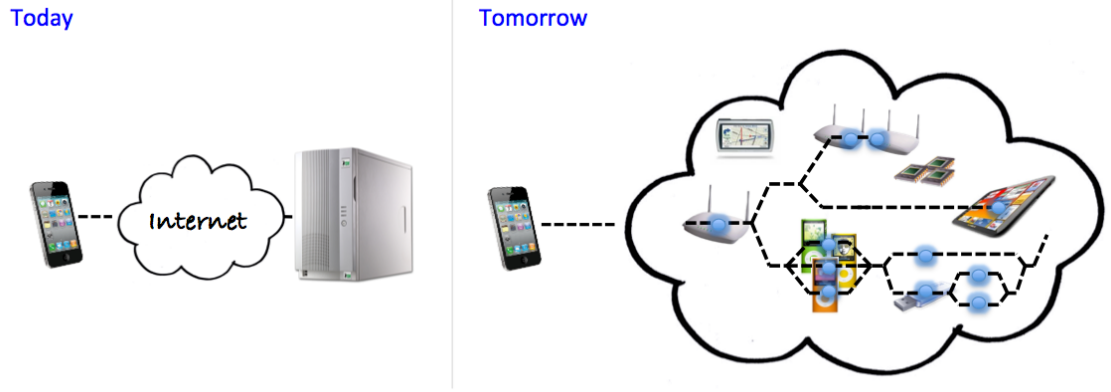


Figure 1.1: Paradigm shift in Service Delivery

## 1.2 Scenario

### 1.2.1 Running Example

In order to highlight our research challenges, we adopt a scenario-based approach that will have research questions both explicit and contextual (that is, grounded in future real life situations). The scenario describes a future commuter's journey to the office from an end-user's perspective and it is structured in different scenes.

**Travel Life Scenario:** on the way to the office.

**Actors:** Alice.

**Abstract:** the scenario depicts Alice's morning before work; it starts in the morning at home and follows her through her underground journey. Alice is a young professional living in London, commuting to the office and often travelling for work. She is sociable,

she likes to be kept informed and to make the most of her time. She is currently looking for a plumber, but she doesn't know any local one.

Alice owns a **MyGazine**, a next generation Internet tablet with Flexible Colour Touch Screen, GPS and wireless connectivity, which is very similar in concept to products researched and produced in 2011 by companies such as LG Display (Figure 1.2) and Samsung LiguaVista (Figure 1.3). The MyGazine runs the core embedded HTML5 application it



Figure 1.2: LG ePaper Display



Figure 1.3: Liguavista ePaper Tablet

takes its name from: the MyGazine App. Although specifically designed for the MyGazine, the MyGazine App runs on a variety of smart devices (e.g., smart phones and Internet tablets). The MyGazine App creates a virtual magazine whose content is – at any time – personalised to Alice's taste and context. To do so, the MyGazine App aggregates content, services and applications (e.g., traditional news, local micro-blogs, video pod-casts, music channels, rich maps and widgets on any sort) available at a given instant from devices in the surroundings, as well as traditional servers. Hence, it opportunistically selects them according to Alice's taste.

The MyGazine App thus works as a recommending mash-up companion that preserves Alice's attention span, entertains her, and provides her with the most relevant content and information at any time.

The MyGazine App gathers information, applications, services and content (audio or visual) for free from other devices and – being based on HTML5 – it allows users to cache them locally and consume them even when no longer in reach.

Users can configure the MyGazine App to add specific widgets and to have them available on the MyGazine dashboard, together with other data and services automatically selected by the MyGazine App. Alice has installed two widgets on her MyGazine: the Smart Media Channel and the Grocery Shopping Planner. The first one is an application creating a personalised podcast channel for Alice, by gathering and playing audio and visual content for free from other devices, mocking the functionalities of radio and TV channels. The second widget is an application interacting with kitchen appliances that automates and assists Alice in her grocery shopping. Alice has pre-configured the MyGazine App to have the Smart Media Channel always accessible, whilst the Grocery Shopping Planner should surface just when Alice is in the kitchen (or upon request).

This **scenario** explores how people and machines will interact and make use of future services and technologies in pervasive environments. Transports, people, buildings and objects, will be closely integrated and connected; they will be endowed with pervasive devices and will be part of the digitalized information society.

### Scene One - Breakfast at Home

It is 7:30 am and Alice is having breakfast in the kitchen. While Alice is enjoying her coffee, the MyGazine downloads the latest BBC news, which will be surfaced to Alice as she switches on her MyGazine. Also, as Alice is in the kitchen, the Grocery Shopping Planner is running; it interacts with Alice's refrigerator service to gather the shopping list for the day, hence it adds them to Alice's online Tesco basket. The shopping list is very long: Alice's refrigerator is almost empty; the Grocery Shopping Planner so decides to prompt Alice. She adds a couple of other items and checkouts: few clicks and her shopping will be delivered to her door in time for dinner.

### Scene Two - On the Tube to the Office

Alice leaves home and walks to the nearest tube station to go to work. As she gets on the underground train, she takes an empty seat and switches on her MyGazine. Meanwhile, her MyGazine has discovered a number of services and content available in the surroundings and has filtered them to just select the most appealing and relevant to Alice. As Alice switches on her MyGazine, the MyGazine creates for her a virtual magazine page, which aggregates content and services according to Alice's tastes.

The services and content the MyGazine has selected for Alice are:

- *'Where is London Twitter?'<sup>1</sup>* – 'Where is London Twitter?' is a service made available by the London Underground to all its passengers. 'Where is London Twitter' gathers the content, location and publication time of all the tweets that London Underground passengers publish or share (through their mobile devices), while commuting; hence, it displays them on a map of London, prioritising the most recent ones, and highlighting emerging concepts/tags by area. In addition to passengers' tweets, 'Where is London Twitter?' surfaces (and boosts) sponsored tweets, i.e. tweets from paying advertisers;
- *eBay Local* – eBay Local is a service made available by a passenger, in Alice's surroundings, that has subscribed to eBay Local Affiliate Programme. Users that

---

<sup>1</sup>Developed on the same idea underpinning 'Where is Twitter?' <http://visualmotive.com/twittermap/>

subscribe to the eBay Local Affiliate Programme get a percentage of all transactions initiated through their eBay Local service instance; also, they get premium points for every new affiliate they recruit (i.e., for every user that stores and hosts locally on their device the eBay Local Application and makes it available for other people to use);

- the Smart Media Channel – The Smart Media Channel is an application pre-installed in Alice’s MyGazine creating personalised podcast channels for its users by gathering and playing audio and visual content for free from other devices, mocking the functionalities of radio and TV channels. Advertisements are injected from time to time, either interrupting the audio/video streaming, or by means of interactive banners. The content to be played, as well as the adverts to be shown/reproduced, are selected based on what is currently available in the environment, taking into consideration Alice’s tastes;
- The news previously downloaded whilst Alice had breakfast;
- Some blog-posts published and shared by passengers in the surroundings or by the Metro Magazine<sup>2</sup> Information System, which is embedded in all London stations and trains;
- Some adverts beacons by some of the advert banners on the train;
- The business profile of two plumbers that have made their business profiles publicly available as an hContact card [Allsopp, 2007]. In fact, as Alice is looking for a local plumber, she has configured her MyGazine App to search for plumbers.

### Scene Three - Interacting with the MyGazine

As the MyGazine shows to Alice the new virtual page, she interacts with the Smart Media Channel and starts to listen to some music from Nedry, a London local band. Whilst listening to music, Alice skims through news and blogs to find a very interesting blog post about the local events in London for the week; she is not really interested to read it all now, so she decides to save it (locally on her device) and read it later in the week when she will be wondering what to do in her spare time.

Also, she notices the Local eBay widget. She loves the idea of finding products sold through eBay by local sellers. She can talk to them, possibly even see the products they sell for real and she does not have to pay for post delivery. She decides to give a go to the Local eBay service and she downloads it locally. From now on, whenever someone will be selling products on eBay, she will be able to see them on her MyGazine (if her MyGazine judges them relevant to her). As for now, she is not sure she wants to become a Local

---

<sup>2</sup>[www.metro.co.uk/](http://www.metro.co.uk/)

eBay affiliate, so she does not subscribe to their affiliate programme; better check how it works first.

### **Scene Four - Gym Deal**

Alice is still on the train, she is now watching a video the Smart-Media Channel has gathered from the surrounding. An advert, on the top banner, suddenly distracts Alice. This is an advert about a new gym that is offering a deal for all subscriptions made within this week: 50% discount. Alice seems to recognise the place as one close to her office. She is not sure, so she makes an explicit request to her MyGazines to search for a navigation service, or at least location one, that would allow her to locate the Gym.

There is a TomTom service available in the surroundings. Alice accesses it (through her MyGazine) and looks for the address in the banner. She is right, the gym is close to her office. She saves the advert and creates an alert to remind her to go and check the gym during the lunch break.

### **Scene Five - Meet Bob**

The train has just stopped at Oxford Circus, Alice is listening to “To Build a Home” by the Cinematic Orchestra, song selected for her by her Smart Media Companion and streamed by one of the passengers in her couch. A lot of people get off the train and many get on. The music suddenly stops playing and the Smart Companion selects some new content. Some services appear and some others disappear. Among the new ones surfaced by Alice’s magazine, there is a widget of music scores for drummers. That is spot-on, Alice has just recently started playing drums. Alice starts exploring the available music scores collections; there are few music scores and exercises that can be downloaded for free. Alice starts downloading them; meanwhile she checks if the provider of that service has a chat service enabled. He does, Bob is his name. Alice contacts Bob to thank him for the scores and starts chatting with him. He gives drum lessons in his spare time and he seats on the same carriage, not far from her. So they start talking for real. Alice switches off her MyGazine and puts it back on her bag.

## **1.2.2 Research Challenges**

### **Information overload problem**

In the following, we depict the main challenges introduced by the various scenes of the scenario above.

First and foremost, we remark that users will be exposed – along the day – to a universe of

services and information. Whilst sometime users will know exactly what they are looking for and they will be able to formulate targeted service/content requests, often users will not know what is available to them and they will need assistance to discover the services most relevant to them.

In the scenario for instance, Alice is exposed, along just about an hour, to a great variety of services and information: BBC latest news, the Grocery Shopping Planner, ‘Where is London Twitter?’, ‘eBay local’, local blog posts, adverts, the Smart Media Companion, TomTom, video and audio content, personal contacts or profiles, etc. While she explicitly requests for some of them (e.g., the Smart Media Companion in Scene Two, the Grocery Shopping Planner in Scene One, a navigation service in Scene Four, a chat service in Scene Five), more often she is unable to explore all services/content available and she needs services/content to be filtered based on her personal taste. For instance, this is the case in Scene One for ‘Where is London Twitter?’, ‘eBay local’, local blog posts, adverts, the video and audio content the Smart Media Companion plays and, in Scene Five, for the music drum scores. We refer to the challenge just described as the **Information Overload Problem**.

Information Overload is not a new problem; in fact, since the uptake of the Internet and of a Global Economy, we have been exposed to an ever-increasing universe of content, services, products and information impossible to explore. Although Recommender Systems have emerged as successful tools to address this issue, in the context of ubiquitous computing, information overload poses new challenges and it will require new solutions grounded on the very nature of pervasive services.

First and foremost, ubiquitous content and services will be mobile and just of local interest. In the scenario, for instance, ‘Where is London Twitter’ is just available to people living or transiting through London, and just when using London transports. Unlike items of global appeal (e.g., products sold on Amazon or movies rented on NetFlix), ‘Where is London Twitter’ will be relevant just to people that either have, are or will be transiting through London transports. Recommendable items will be not available to any user and not at any time. So, Recommender Systems for pervasive systems will have to ponder the local nature of the recommendable items.

Also, because of their very nature, pervasive services will be often combined to provide richer functionalities to end-users. In the scenario, this is for instance the case for the Smart Media Player, ‘Where is Twitter’, the Grocery Shopping Planner, the blog-spots aggregated together, etc. Recommendable items will not always be atomic entities and Recommender Systems for pervasive services will also need to cater for composed items. Furthermore, very little will be known on new or unusual compositions and so the item cold-start problem, which affects traditional recommender systems, will be further aggravated in pervasive environments.

### Reliability problem

Secondly we observe that, beside services and content delivered by traditional online web servers, users will be consuming services/content provided by other users' devices or by devices embedded in buildings, transports or objects. In our scenario, 'eBay local', blog posts, plumber profiles, the TomTom navigator service, the music scores, audio and video content are services hosted by other users' mobile devices and shared with users in the surroundings. Other services are provided by devices embedded within the physical spaces instead, and just made available to people traversing them. The refrigerator shopping list, the underground Metro news, the adverts beacons by adverts banners within trains and 'Where is London Twitter' are all services served by objects or devices embedded in Alice's kitchen or the underground station or the train.

Both kinds of services have in common that they are available just to consumers in the *local* proximity and they are *mobile* to them. They are *local* as they are useful only to people in the surroundings and, hence, are made available to devices in their proximity. They are *mobile* as their host is mobile and/or the host is fixed in a location, but it is accessible just within local area networks and the consumer moves in and out of such area. This applies also to 'Where is London Twitter', although served by a traditional web server; in fact, access to it is restricted just to users located in any station or transport of London. A new challenge then rises as whether service providers will be co-located to the consumer for the entire duration of the service, where by co-located we mean that are within a wireless one-hop distance. We refer to this last problem as the **Co-location Reliability Problem**.

With respect to reliability of pervasive services, we observe that the main remarkable difference between services executed in usual web environments and the mobile ones is that service (and content) providers will be mobile to service consumers. Therefore they may not be available for as long as required.

In fact, unlike traditional service providers, providers of pervasive services – being mobile and local – will keep appearing and disappearing as users move around. In the scenario, for instance, when Alice's train gets to Oxford Circus station, the streaming source for the Song "To Build a Home" disappears, whilst a new music scores service appears. Similarly, as Alice walks away from her kitchen, the refrigerator services are lost.

Hence, it will become crucial to ponder providers' mobility and local proximity – in addition to the QoS they guarantee - when selecting services.

As we have already highlighted, services will be often composed. So, it will be imperative to reason on the mobility and local proximity of groups of providers, as well as single entities.

## 1.3 Hypothesis and Contributions

Building on the above observations, we look at the service discovery in mobile environments in the context of providing users with a pleasant experience.

**Main Hypothesis.** *We argue that by reasoning on people habits and tastes, ‘better’ services compositions can be formed. In the thesis, by ‘better’ we mean both ‘more reliable’ and ‘more enjoyable’.*

The goal is twofold: ensure that (i) the end-user will enjoy a composite service and (ii) the composite service is perceived by the end-user as provided by a single provider available for the entire duration of the service.

From a *discovery* perspective this translates into:

- Selecting services and content appealing to individual users among an endless choice of pervasive services and content;
- Selecting services and providers that optimise the reliability of the execution and composition of mobile services.

We thus look at *appeal* and *reliability* as two distinct qualities of a (composite) service and we approach Service Discovery as a filtering/ranking process that we implement through a sequence of distinct filters/rankers applied one after the other, as depicted in Figure 1.4. More precisely, we model the selection of ‘more reliable’ and ‘more appealing’ compositions as a two distinct filters that we chain to obtain an overall filter selecting ‘more appealing and reliable’ services.

The overall service selection process that we obtain is the following. As we initiate a service discovery, we consider the set of all the services available in the surroundings; hence, we select the services and content appealing to end-users. We so obtain a sub-set of services (and providers) that we further scrutinise to pick those services and providers that optimise the reliability of the composition and its execution.

Note that this filtering model allows us to create a frameworks that is flexible, easily extensible and configurable. In fact, to add a new aspect to the selection process (e.g. a QoS reasoner), it is enough to add the corresponding filter (e.g. a filter based on QoS values) and chaining it to funnel pipeline. More importantly, we can play with the order and the nature of the filters in the chain to dictate the overall filtering strategy.

In the following, we discuss in details the two “filters” underpinning the Discovery Method we propose, as well as the specific contributions made.

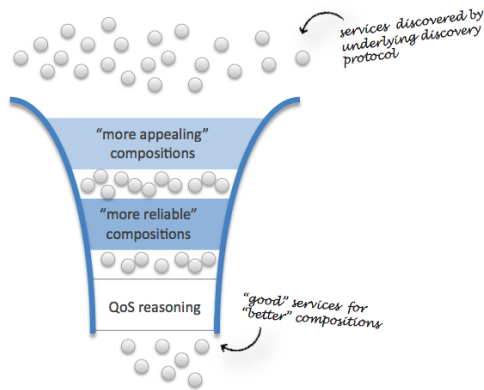


Figure 1.4: Filter Chain

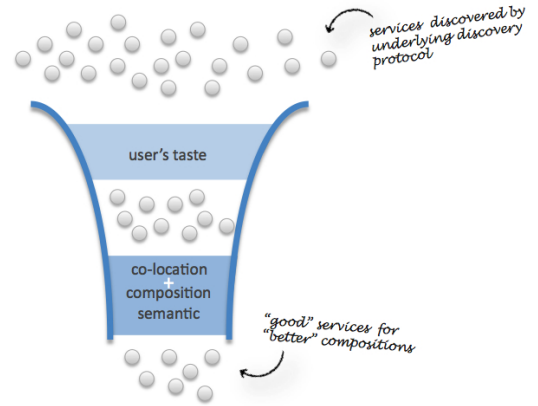


Figure 1.5: Approach

### 1.3.1 ‘More Appealing’ Services

To select services appealing to users as individuals, we adopt a method already in use, with success, on the Web: personalised recommender systems. The idea underpinning personalised recommender systems is quite simple: the appeal and appreciation for an item is subjective. In fact, different people have often very distinctive opinions on a same item, regardless of its intrinsic qualities. As a consequence, not the same items should be promoted to all users, but different items that match the personal preferences and tastes of each individual should be recommended instead.

Traditional recommender systems, though, are not appropriate to mobile environments. This is mainly because they fail to address the mobility of service providers and the local nature of the services. Also, we remark that – due to their granular nature – ubiquitous services and content will be used as part of novel composite services or aggregated content.

We make three main contributions on this area:

1. A novel algorithm and rating data structure to recommend items of local and mobile nature;
2. A novel algorithm and data structure to recommend items of composite nature;
3. A thorough evaluation on large and (when possible) real datasets of the accuracy and coverage of the methods proposed, as well as the analytical evaluation of their computing cost and resource allocation.

The results obtained have been published in the following venues

- L. Del Prete and L. Capra. “differS: a Mobile Recommender Service”. In 11th In-

ternational Conference on Mobile Data Management (MDM). Kansas City, Missouri USA. May 2010;

- L. Del Prete and L. Capra. “moRe: a Lightweight Recommender Service for Pervasive Computing”. Journal: IEEE Transactions on Mobile Computing (TMC). Pending.

### 1.3.2 ‘More Reliable’ Services

Service providers of ubiquitous services will be mobile to their customers and, possibly, they will not be available for as long as required.

In order to give users a positive experience, mobile composite services will have to be perceived by the user as supplied by a unique entity that is reachable and available for the duration of the service. Hence, we look at selecting services and service providers so that the proximity availability – here defined as the one-hop connectivity of a service provider to a service consumer – of the component services and of the overall composition is optimised. The ultimate goal is to offer a composition setting that approximates static environments and that attempts only compositions that can be carried on successfully (in terms of proximity availability).

To achieve such goal, we made the following contributions:

1. A co-location duration prediction model, that exploits human regularity of movement<sup>3</sup> to estimate device co-location durations;
2. A composition reasoner, that exploits both the predicted co-location durations for the component services and the composition semantic to attempt only compound services that are highly likely to complete successfully;
3. A thorough evaluation on large real datasets to quantify the accuracy of the co-location prediction model and the reliability of compositions.

This work has been published in:

- L. Del Prete and L. Capra. “Reliable Discovery and Selection of Composite Services in Mobile Environments”. In 12th IEEE International Enterprise Computing Conference (EDOC). Munich, Germany. September 2008;
- L. Del Prete and L. Capra. “MoSCA: Seamless Execution of Mobile Composite Services”. In 7th ACM Workshop on Adaptive and Reflective Middleware (ARM). Leuven, Belgium, December 2008;

---

<sup>3</sup>In terms of pairs co-locations

- L. Del Prete and L. Capra. “MoSCA: Service Composition in Mobile environments”. In *Middleware 2008, ACM/IFIP/USENIX 9th International Middleware*. Leuven, Belgium. December 2008.

Overall, we deal with service discovery in dynamic environments by providing a double filtering mechanism. On one side, we provide a Mobile Recommender System to ease the discovery and filtering of services, content and goods of local nature based on the end-user’s taste. On the other side, we enable the reliable composition of those services (and content) available in the surroundings in order to provide valuable functionalities. The two combined together create a discovery method that ensure “better” service compositions, where by “better” we mean both “more interesting” to the user and “more reliable” (Figure 1.5).

## 1.4 Assumptions

The main goal of our research is to provide a service discovery framework for services and content of a future Seamless Web, where pervasive services will co-exist with traditional ones. To do so, we focus on addressing the main issues that relate to ubiquitous services, whilst still ensuring common abstractions with traditional services. More precisely, we focus on selecting pervasive services based on the end-user’s tastes and their co-location reliability.

One could argue that a software layer that ponders only users preferences and co-location is not a discovery middleware, as it does not tackle fundamental issues, such as QoS. However, reasoning on QoS alone is not enough: mobility introduces new complexities that should not be ignored. We believe middleware should be structured into different layers, each focusing on a specific issue (e.g., standard discovery, QoS, etc.). In our research, we investigate one of these layers, while making the following assumptions as far as other layers are concerned (see Figure 1.6).

- First, we assume the existence of a communication (sub)layer that enables components to coordinate via some wireless data link. We are not interested in what particular communication paradigm is actually provided; we only require that it fits mobile settings, and therefore that it tackles issues such as frequent disconnections.
- Second, we assume the existence of a traditional discovery middleware to advertise, discover and binds hosts, services and resources in ad-hoc networks, as well as traditional ones; this may be a simple broadcasting mechanism implemented on each mobile device, or a more sophisticated approach (e.g., Jini [Waldo, 1999], UPnP [Thomas et al., 2004], DPWS [Driscoll and Mensch, 2009]).

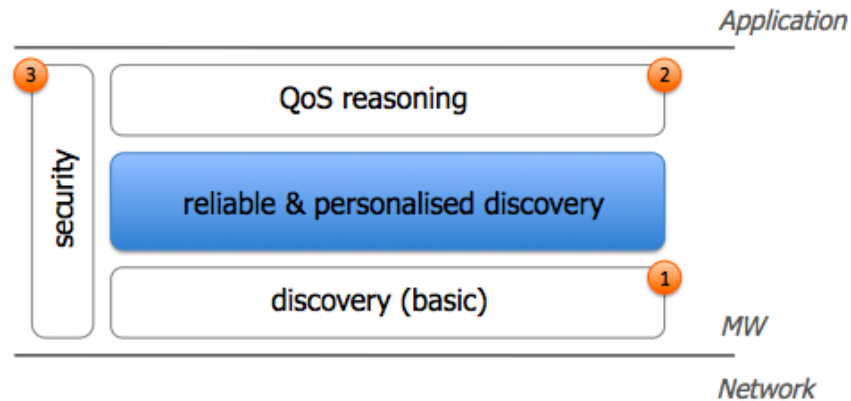


Figure 1.6: Assumptions

- On top of the “basic” discovery layer, we provide a framework that preliminary filters services and providers so that only reliable providers of services appealing to the end-user are considered.
- In addition, as we appreciate the importance of reasoning on QoS (Quality of Service), we take for granted a QoS discovery layer that applies QoS reasoning to decide which services to promote and/or filter out. It is worth noticing that, although in Figure 1.6 we depict the QoS as building on the augmented discovery layer we propose, we make no real assumption on this layer being developed on top our augmented discovery or vice-versa. In fact, QoS reasoning (as any other filtering reasoning) could be executed at any stage, as previously discussed.
- Service provider trust and, in general, the security aspects are outside the scope of our research and orthogonal to the framework we propose. Hence, we envisage a security layer that is traversal to all layers.

With respect to pervasive services and the context in which they will be available, we also take the followings for granted:

- We assume that the services and content being composed are connected with a single-hop distance.
- We assume that a taxonomy exists to map the intent of a requested service to a class of services and to enable to decompose a service in its components [Mokhtar et al., 2006]. This taxonomy could be either universal (and built on OWL-S) or can be specific to a domain/vertical or a set of services.
- We assume services are stateless (e.g., RESTful services [Fielding, 2000]).
- We target metropolitan environments.

- We also take for granted that a number of languages are available to describe services (e.g., WSDL [Christensen et al., 2001], OWL-S [Martin et al., 2004], WADL [Hadley, 2009], OpenSearch [Clinton, 2009]) and how component services interact (e.g., BPEL [Coalition, 2003], WSCDL [WSCDL Coalition, 2005]). In general, we are not interested in the particular language used to depict a composition, as much as we are on the possible manners services can be composed. As such, we will be composition description-language agnostic and, at the same time, provide a set of composition semantics that extend the ones supported by the various languages.
- Whilst we look at methods to dynamically re-define (self-adapt) a composition, we will rely on existing research for context sensing, awareness and reasoning.

## 1.5 Thesis Outline

The reminder of this document is divided into three parts:

- **Personalised Discovery** - this part discusses the main contributions realized in relation to the Mobile Recommender System and their evaluation.  
First, we highlight the problem of making personalised recommendations in mobile environments; we then illustrate the core of our contribution (Chapter 2), before examining the performance and resource overhead of the recommender system proposed to prove accurate recommendations can be made even at very low computational and memory cost (Chapter 3).
- **Reliable Discovery** - this part points out the problem of composing services in mobile environments. First, it presents a novel Discovery System method for reliable compositions (Chapter 4). It then demonstrates that, by reasoning on device mobility patterns and on the constraints imposed by the actual composition semantics, the reliability of composed services (i.e., the number of started and successfully completed compositions) can be significantly improved (Chapter 5).
- **Framework and Future Work** - This part presents a novel mobile service composition framework – including the discovery methods proposed above.  
More precisely, it describes a Middleware for pervasive services, its architecture (with particular focus on the discovery component), its implementation, and possible deployments (Chapter 6). Finally, we summarise the contributions made and explore directions for future work (Chapter 7).

## Part II

# Personalised Discovery



## Chapter 2

# Personalised Discovery of Pervasive Services

The future we envision brings an ever-increasing universe of pervasive services that will be deployed within buildings, transport systems, open spaces, people portable devices and objects of any sorts (e.g., banners, magazines, etc.). People will come across them constantly and will be overwhelmed by nearly limitless options. They will be unable to know all the services available to them and they will be incapable of finding appealing ones.

Enabling the discovery of compelling services will be crucial for pervasive services to be successful. To solve this problem, Recommender Systems [Resnick and Varian, 1997] have emerged and successfully used on the Web [Terveen and Hill, 2001]. These systems automatically advise individual users of items they might like without the need for them to search for specific items or browse through a catalogue. In doing so, they help users discover items they might not otherwise have known were available.

While we believe existing recommender methods could be used in principle for recommending pervasive services, we also believe current recommenders to be unsuitable (as they are) for pervasive services.

In fact, online recommender systems in use today leverage on a catalogue of items that is available virtually to anyone that uses the Web. The effort to build, host, run and maintain them is then justified by the potential and business benefit of appealing to a large crowd.

On the contrary, pervasive services and content will be available just to people that will come across them (i.e., that will be physically co-located with the providers of those services and content). Considering the scenario presented in section 1.2, this the case, for instance, of the services “Where is London Twitter?”, eBay Local, local blog posts and local profiles.

Furthermore, pervasive services (and content) have the potential – across the globe – to become even more numerous than the books sold on Amazon, or the songs bought on

## Chapter 2

iTunes, or the movies rented on Netflix. This would be the case, for instance, if all mobile phones in use today ran and exposed a pervasive service. Hence, it may not be feasible, nor scalable, nor economically plausible to use existing centralised recommenders for pervasive services and content.

It could be argued that to address this issue we could use several centralized recommender systems, each covering the services (and content) available in a specific geographic area; alternatively, the services in the users' proximity could be retrieved at the time recommendations are computed. Still, we would have to deal with two main problems:

- Many service providers will be mobile, so how would an online system know when providers leave and/or enter in new zones or users' neighbourhoods?
- How would centralized online systems know when two devices would be connected to each other?

To solve the first issue, online centralized recommenders could constantly monitor service providers to capture their movements, but that would be practically infeasible.

For the second concern, each terminal could send regularly its GPS coordinates, but this would provide information about physical proximity, not connectivity. In fact, appliances can be physically co-located but unable to communicate (e.g., as metallic walls may be separating them).

Even if these challenges could be solved in convenient ways, there is another issue worth considering. Users will consume pervasive services continuously along the day and in any sort of context. Hence, a recommender for pervasive services will end up knowing a great deal about them. It would not be familiar just with a user's obsession with contemporary art books, or her strong dislike for horror movies, or seasonal music tastes, or her shopping habits: grocery shopping on Tuesday nights, book shopping at the weekend. It could also possibly get acquainted with her habits: every weekday she takes a train at 8:15 AM to travel to central London; she is at the British Library most Tuesdays; she flies home almost every weekend, etc. Hence, there would be a much greater opportunity for users to have, or to feel, their privacy violated. Consequently, users may decide to opt out from this kind of recommenders, causing pervasive services, as a paradigm, to fail.

To overcome these weaknesses we propose *moRe*, a fully distributed mobile recommender service that specifically targets pervasive computing settings.

The core idea underpinning *moRe* is for users to maintain their own profile (i.e., ratings about previously consumed pervasive items) locally on their mobile device. Using radio technology (e.g., Bluetooth 2, Wi-fi), devices exchange their users' profiles during periods of colocation. So, each user gradually builds a virtual view of the local community's preferences. In order to optimise storage, *moRe* builds on a new data structure, called *Rating Hash Tree*, that locally persists profiles in a very compact format. When a recommendation has to be computed, *moRe* dynamically assesses whether the user is *mass-like minded* or *individual*: in the former case, no sophisticated recommendation algorithm is required,

and the mean rating for any given item is returned; in the latter case, a novel algorithm is used instead, that traverses the rating hash tree to efficiently compute personalised recommendations.

Following the trends we have witnessed on the Web, we anticipate a number of pervasive services to be provided as *compositions* of finer-grained services and/or aggregated content; as a result, a mobile recommender service must be able to predict the user's enjoyment of a compound service, starting from (a subset of) its components. moRe exploits team performance theory to accomplish this goal: it models fine-grain services as team members, and then it derives a prediction about the composite service based on the individual members and how well they fit together. We have conducted an extensive evaluation of moRe, both analytically and experimentally, using a variety large and (when possible) real datasets. Our results demonstrate that moRe achieves an accuracy and coverage that are comparable to those of centralized recommender systems in use today, while considerably reducing both memory allocation and processing overhead, thus being suitable to run on current smart-phones

The reminder of the chapter is structured as follow: we begin by providing background information on recommender systems (Section 2.1); we then highlight open challenges (Section 2.2). We introduce the conceptual model for *moRe* (Section 2.3) and detail how it has been designed and engineered, bearing in mind mobile device constraints (Section 2.4). We then compare our work to existing research on mobile recommender systems (Section 2.5). An extensive evaluation of moRe performance will be presented in the following chapter.

## 2.1 Background

With the uptake of the Internet, we have been exposed to an increasing universe of content, services and information impossible to explore. In order to cope with this information abundance, recommender systems have emerged. New systems like Tapestry [Goldberg et al., 1992], GroupLens [Resnick et al., 1994], Fab [Balabanovic and Shoham, 1997], Ringo and HOMR [Shardanand and Maes, 1995] started to assist users discovering emails, movies, web pages, albums or artists suited to their interests. These systems, known as Recommenders or Recommender Systems, have since spread to virtually any item driven domain. We get recommendations on products, songs, games, news, restaurants, web sites and content of any sort.

Though a variety of algorithms have been proposed in the last twenty years, with more or less promising results and with various scenarios in mind, just two main paradigms for computing recommendations have emerged, namely content-based filtering (a.k.a., cognitive filtering) and collaborative filtering (a.k.a., social filtering). The first one computes

similarities between items based on their features to then recommend to users items similar to the ones they have expressed interest in. The second one computes similarities between users, based upon their rating profile (i.e., tastes as understood from their past activities, purchase history or explicit reviews). Users similar to an active user  $u_a$  then serve as ‘advisers’ suggesting the most relevant products to her.

In recent years the two paradigms have been often combined to mitigate the limitations of either approaches and exploiting synergetic effects, thus originating a third paradigm: hybrid systems.

Methods within both content-based filtering and collaborative filtering can be further classified on the basis of how they gather information about either the content itself or social relationships (see Figure 2.1). We refer to methods using a heuristic approach as Memory-based methods, whilst we refer to methods that learn and use a cognitive model as Model-based methods.

<b>recommenders</b>	<b>content-based filtering</b>	
	<b>collaborative filtering</b>	user-based CF
		item-based CF
		matrix reduction
	<b>hybrid</b>	weighted
		switching
		mixed
		feature combination
		cascade
		feature augmentation
		meta-level

Figure 2.1: Recommenders - Taxonomy

In the following, we introduce the main methods and discuss their suitability to recommend pervasive services or content. In doing so, we focus mainly on Collaborative Filtering methods. This is because Collaborative Filtering has gained greater popularity thanks to its intrinsic ‘simplicity’ and agnosticism to content modelling. Thus it suits (almost) any kind of recommendable content that cannot be (significantly) described, or whose meta-data does not seem to explain alone user preferences, like pervasive services and content.

### 2.1.1 Content-based Filtering

Content-based filters are by far the more mature technology. They model users as isolated entities, thus allowing to only filter or make recommendations based on the items themselves. For this very reason, content-based filters have been mostly applied for recommending text documents. Initially, systems in this category were essentially *memory-based* personalised search engines, with users, not only items, being represented as vectors of keywords. Because a user profile was essentially a tuple of keywords inferred from the frequent terms in documents that user had saved, marked as interesting, or spent a lot of time reading, recommending was essentially a matter of filtering those documents where the user’s keyword were prominent [Bennett, 2006].

To this category also belong those methods that are based on the clustering of items. In this case, items (most often documents) are clustered according to some description or meta-description either inferred from the analysis of the content or explicitly fed to the system. Then, users’ profiles simply correspond to a set of interest flags (either positive or negative) to each of the item clusters. In its simplest form item clustering is still largely employed nowadays, for instance for email subscriptions and rudimentary e-mail campaigns.

Most of the document recommenders using item clustering use a *model-based* technique very common in search engines: TF-IDF (Term Frequency - Inverted Document Frequency). The idea behind TF-IDF is pretty simple: the more a term is repeated in a document the higher is the probability that a document relates to that term. This holds true as far as the term has a discriminant quality (i.e. is not very frequent in all document within a catalogue or domain). For instance, although the word ‘the’ may be very frequent in this document, it is also very popular in any document (i.e., it is not discriminant); as such, it is not representative of this document. On the other hand, the term ‘composition’ is largely used within this document, whilst is less employed within the English language; as such, it is a representative word for the concepts here introduced.

Bayesian Classifiers are also part of *model-based* methods within this category. Bayesian Classifiers build on top of an initial set of correctly classified items. Items are described by a set of attributes. As a first step, a Bayesian Classifier computes, for the initial set of correctly classified items, the probability of item attributes to appear in each cluster. Then, it uses these probabilities to estimate the probability of an item to belong to a cluster

given its attributes. This is done as per Bayes' theorem of conditional independence. It is worth noticing that Bayesian classifiers make the 'naive' assumption that item features are independent, which is usually not the case.

There are two major limitations to the use of content-based filtering. Firstly, some item types have no intrinsic content or meta-data adequate to describe them or understand users' opinions in their regards. Secondly, content-based filtering always returns content within clusters already known to the user and so are unable to generalize, causing users to miss out on interesting items outside the clusters already experienced.

Compared to other methods, content-based filtering uses very little information to predict a user's preferences (i.e., the attributes of the items used by the users and possibly, for each of them, a preference score). Hence, from a resources perspective, content-based filtering is ideal for mobile devices. On the other hand, in our research, we predict pervasive services and content to be of various types; also, we do not expect them to come with meta-data apt to explain users' preferences. As such, because of the limitations we just pointed out, content-based filtering may be inadequate for mobile environments.

### 2.1.2 Collaborative Filtering

Unlike content-based filtering, collaborative filtering provides a model that is agnostic of the type of items recommended and the actual characteristics of users. Rather than depicting items by their descriptive attributes, and users by their interests or segment features, in collaborative filtering an item is described by the users' ratings related to it. Similarly, users are represented by the ratings they have provided for each of the items consumed. As such, collaborative filtering builds upon a *rating matrix* structure that organizes user profiles in *user-rows* and items in *item-columns*. The value  $r_{i,j}$ , stored at position row  $i$  and column  $j$ , simply corresponds to the rating of user  $i$  for item  $j$ .

In general terms, the main task of collaborative filtering is to estimate the values for the  $r_{i,j}$  missing within the rating matrix – i.e., to predict the preference of a particular user (the active user) for an item – based on the preferences expressed by other users.

In the following, we first introduce memory-based collaborative filtering methods to then briefly touch on the less popular model-based collaborative filtering methods, for completeness. More precisely, in discussing memory-based collaborative filtering methods, we highlight the core logic of (i) user-based collaborative filtering, (ii) item-based collaborative filtering, and (iii) matrix reduction methods.

#### Memory-based Methods

**User based Collaborative Filtering.** Instead of attempting to explain the reasons for a user to like/dislike an item, user-based collaborative filtering builds on the simple observation that users who agreed in their evaluation of items in the past are likely to do

so again in the future. It mimics the same dynamics occurring in real-life for users seeking advice and then forming an opinion (i.e., people seek for advice from their peers and then ponder the genuine advice more or less heavily depending on the affinity they recognise with the ‘advisor’). Similarly, in user-based collaborative filtering the prediction of a user rating on an item is based on the ratings of the most similar users.

To do so, user based collaborative filtering applies a three-step approach:

- User Similarity - The users’ similarity to the active user are computed;
- Neighbours - The  $k$  users that are the most similar to the active user are selected;
- Rating Prediction - The preference expressed by the nearest neighbours on the item to recommend are weighted by the user similarity with the active user and then averaged.

**User Similarity.** A variety of similarity measures have been proposed in the literature. These measures mainly build upon the fact that users’ profiles are modelled as rating vectors. Hence, the distance between their vectors can serve to calculate users’ similarities. The most popular of the measures used is the Pearson correlation [Resnick et al., 1994], which is nothing but the cosine correlation of rating variations. Being  $S_a$  and  $S_u$  the set of items rated by the active user  $a$  and the user  $u$  respectively,  $\bar{r}_a$  and  $\bar{r}_u$  the mean preference expressed by user  $a$  and  $u$  across all items rated, the Pearson similarity measure is defined as follow:

$$pearson(a, u) = \frac{\sum_{i \in S_a \cap S_u} (r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i \in S_a \cap S_u} (r_{a,i} - \bar{r}_a)^2 \sum_{i \in S_a \cap S_u} (r_{u,i} - \bar{r}_u)^2}}$$

Alternatively the Cosine correlation [Sarwar et al., 2001] can be used

$$cosine(a, u) = \frac{\sum_{i \in S_a \cap S_u} r_{a,i} r_{u,i}}{\sqrt{\sum_{i \in S_a \cap S_u} r_{a,i}^2 \sum_{i \in S_a \cap S_u} r_{u,i}^2}}$$

Simpler measures exist, like the Manhattan correlation [Candillier et al., 2008a] and the Jaccard correlation [Candillier et al., 2008a]. The first one measures the sum of the (absolute) differences of the ratings expressed by two users on the same items; whilst the second computes the overlap of the items rated. Being  $max_r$  and  $min_r$  the maximum and minimum possible rating values, then:

$$manhattan(a, u) = 1 - \frac{1}{max_r - min_r} * \frac{\sum_{i \in S_a \cap S_u} |r_{a,i} - r_{u,i}|}{|i \in S_a \cap S_u|}$$

$$jaccard(a, u) = \frac{|S_a \cap S_u|}{|S_a \cup S_u|}$$

**Neighbours.** Neighbours for the active user  $a$  are selected on the basis of the similarity score obtained. Two main methods exist for electing the neighbours on an active user: threshold filtering and  $k$ -NN (k-Nearest Neighbours). If the first method is used, then all users whose similarity with the active user is higher or equal to a predefined threshold value  $th$  are considered neighbours of the active user; otherwise, just the  $k$  users with the highest similarity measure are elected as the active user's neighbours.

**Rating Prediction.** The rating of user  $a$  for an item  $i$  is predicted by combining the ratings of neighbour users on the same item. This rating can be estimated by the weighted sum of the ratings of the neighbours (set  $N$ ), where the weight is the user similarity between users  $a$  and  $u$  ( $sim(a, u)$ )

$$p(a, i) = \frac{\sum_{u \in N} sim(a, u) r_{u, i}}{\sum_{u \in N} sim(a, u)}$$

As users may be more or less conservative when expressing their feedback, a method to compute rating predictions based on deviations from the mean ratings has been proposed. In that case, the sum of the user's mean rating and the weighted sum of rating deviations from the mean rating are used

$$p(a, i) = \bar{r}_a + \frac{\sum_{u \in N} sim(a, u) (r_{u, i} - \bar{r}_u)}{\sum_{u \in N} sim(a, u)}$$

**Performance Improvements** Several modifications have been implemented to the core user-based collaborative filtering with the goal of making better predictions. The most relevant improvements are:

- Inverse user frequency. Inverse user frequency [Breese et al., 1998] adopts the same conceptual model of TF-IDF and rewards co-votes for less common items much more than co-votes for very popular products.
- Significance weighting. The computation of user-user correlations only considers items that both users have rated; hence, even if two users have co-rated only one single item, they may have maximum correlation if they have rated identically the same item. Clearly, correlations being based upon few data-points only are not very reliable. Significance weighting policies have hence been introduced to weight the same similarities [Ghazanfar and Prgel-Bennett, 2010]. The most popular significance weighting employs the Jaccard correlation in combination with the similarity correlation of choice.
- Filterbots. One of the main challenges of collaborative filtering is data sparsity. In fact, sparse rating matrices have small subsets of co-voted items to rate users similarities. Hence, the predictions computed are not reliable. In order to address

this issue, synthetic ratings are created. The mechanics are quite simple in principle, synthetic users are modelled with specific traits (e.g. attention to the price) to then score all possible items accordingly to the synthetic user's features. The ratings so created are fed to the system as if input by real users [Sarwar et al., 1998].

User-based collaborative filtering is affected by some non-trivial problems (beside scalability), which originates from the very nature of the method: it needs rating data to work well. In fact, user-based collaborative filtering exploits the knowledge inferred from users activities; hence, the more rating data is available, the more data to reason about, and therefore the more precise the predictions. These problems are namely the *cold start problem* and the *sparsity problem*. The cold start problem (a.k.a. start-up problem or rampup problem) is a twofold issue as it applies to both users and items. In fact, as a new user is introduced, she has no correlation with existing users and thus she may not get any personalised recommendations. Similarly, new items have no ratings and therefore may be not recommended. A similar and probably bigger problem is the rating sparsity problem. It affects mainly those portions of the recommendable catalogue with a very restricted user basis and those users whose tastes are atypical within the user basis of the catalogue. Because the amount of rated items may be limited compared to the catalogue size, the sets of co-voted items by any two users may be not large enough to enable accurate predictions.

In our research, we build upon user-based collaborative filtering due to its intrinsic qualities. In fact, Collaborative Filtering provides a model that is agnostic of the type of items recommended and the actual characteristics of users; as such, it is suitable to recommend pervasive services and content of various kinds. We acknowledge, though, that in mobile environments the problem of *sparse rating matrix* may be even more critical than in traditional scenarios. This is because pervasive services and content are not available on a global scale but just to local users. For these reasons, in our research we are particularly interested in targeting atypical users, as well as finding a method to serve new users.

**Item based Collaborative Filtering.** Item-based collaborative filtering [Sarwar et al., 2001] is symmetric to user-based collaborative filtering and shares with it a very similar approach, the same similarity measures and all of the limitations and strengths highlighted so far for user collaborative filtering. The observation underpinning item-based collaborative filtering is that users are more likely to like items that are similar to the items that they have already appreciated in the past. Intuitively, the approach mimics real user behaviour, as people often judge unknown items by comparing it to known, similar items. To achieve this, item based collaborative filtering applies a three-step approach:

- Item Similarity - The items' similarity to the active item are computed;
- Neighbours - The  $k$  items that are the most similar to the active item are selected;

- **Rating Prediction** - The preference expressed by the active user on the nearest neighbour items are weighted by their similarity and then averaged.

In our research we take a user-based rather than item-based approach, as we predict users to own and exchange their rating profiles.

**Matrix reduction methods.** Matrix reduction methods have recently been investigated in collaborative filtering to relate items to users. The idea underpinning matrix reduction methods is that there must exist some discriminant traits of items and users that explain the reasons for a user to like or dislike an item. Unlike content-based filtering algorithms, these features are not pre-known or imposed, but are rather discovered by analysing the rating data collected. They analyse the relationships between a set of users and the items they have scored by producing a set of non obvious features (latent features) that relate both to items and users, thus creating a common ground that allows to generalise the way people rate items in a data set. The problem that matrix reduction methods address is to find those latent features that explain users' ratings and, hence, can be exploited to predict missing ratings. They examine the rating matrix to find a lower rank  $k$  approximation for it; intuitively this means finding  $k$  latent features that describe users and items and reasonably reveal how users rate items. To do so, matrix reduction methods in its simplest form use a three-step approach:

- **Description** - This phase aims to find a concise description of users' ratings to then create a generalisation model for predicting missing ratings. It effectively consists in the approximation of the Rating Matrix to obtain a latent feature matrix relating users to items.
- **Generalisation** - In this phase, a generalisation model is created from the description previously obtained. The rating matrix approximation computed in the description step is factored into two matrices depicting the relevance of features in user profiles and item profiles, respectively.
- **Prediction** - the preference of an active user for an item is estimated using the generalisation model and pondering the active user's profile by the profile of the item to recommend.

**Description.** This phase aims to find a concise description of the users' ratings to create a generalisation model for predicting missing ratings. The Rating Matrix is approximated with a lower-rank matrix to obtain a latent feature matrix relating users to items. To approximate the rating matrix, Singular Vector Decomposition (SVD) is often used. SVD has, in fact, the ability to provide the best approximation of a matrix in terms of Frobenius norm (i.e., the square root of the sum of the absolute squares of the error matrix elements).

If we consider a rating matrix  $R$ , that collects users' rating so that each row vector  $u_i$  corresponds to a user profile and each column vector  $i_j$  corresponds to an item profile, then  $R$  will be formed by  $n$  rows (where  $n$  is the number of users) and  $m$  columns (where  $m$  is the number of items). The SVD approximation technique factors the matrix  $R$  into three matrices as  $R = USI$ .  $U$  and  $I$  are two orthogonal matrices of size  $n * k$  and  $k * m$  respectively,  $S$  is a diagonal matrix of size  $k * k$  and  $k$  is the rank of the original matrix  $R$  (i.e., the maximum number of independent rows or columns). Intuitively,  $k$  is the number of the latent features describing the user-item space; each row in  $U$  relates users to the latent features and each column in  $I$  relates items to those same features. The singular values in  $S$  (i.e., the diagonal elements) can be thought of as scalar 'gain controls' by which the presence in a user profile of the corresponding feature is related to the same feature in an item.

**Generalisation.** In this step the approximation  $R = USI$  is factored to obtain two distinct matrices  $U^*$  and  $I^*$  so that  $R = USI = U^*I^*$ . These two matrices effectively convey the relevance of each feature in describing each user and each item respectively. They are computed as follow. First  $S^{1/2}$  is calculated, then  $U^*$  and  $I^*$  are defined as  $U^* = US^{1/2}$  and  $I^* = IS^{1/2}$ . Intuitively  $U^*$  and  $I^*$  are the  $k$  dimensional representation of the  $n$  users and  $k$  dimensional representation of  $m$  items.

**Prediction.** The matrices obtained as the generalisation model are then used to compute the recommendation score of the active user  $a$  for an item  $i$ . To predict this rating the dot product of the  $a^{th}$  row of  $U^*$  and the  $i^{th}$  column of  $I^*$  is calculated. Hence

$$p(a, i) = U_a^* \cdot I_i^*$$

SVD is more efficient in memory and secondary storage allocation than user-based and item-based collaborative filtering as it stores just two reduced users and items matrices of size  $n * k$  and  $m * k$  respectively. As such, if used for a mobile recommender it would enable to save memory space compared to Collaborative Filtering. On the other hand SVD poses issues in processing resources, as the decomposition process requires a processing effort remarkably greater than for user-based and item-based collaborative filtering. Also, preliminary studies show how SVD provides in average better recommendation than user-based and item-based collaborative filtering for dense rating matrices, whilst performs consistently worse than those in the case of sparse matrices [Huttner, 2009]. As already discussed, we anticipate the problem of sparse rating matrices to be even more significant for mobile recommender systems than in typical recommenders. Also, mobile recommenders will be constrained in terms of computing resources and energy supply; as

such, methods that minimise data processing should be favoured. For these reasons, in our research, we have disregarded SVD as a possible method to build a mobile recommender system.

### Model-based methods

Model-based collaborative filtering methods model the collaborative filtering task as calculating the expected value for the preference of a user on an item, given the user ratings on other items. They provide item recommendations by first developing a model of users' ratings, which is built upon different machine learning algorithms such as clustering and Bayesian networks.

Methods that exploit Bayesian networks create a probabilistic map of the likelihood that a user might like a certain item given her feedback on other items. The mathematical basis for this is Bayes theorem of conditional probabilities. A Bayes network is a set of such conditional probabilities – each corresponding to a node – mapped together in a directed acyclic graph. In Collaborative Filtering recommenders exploiting Bayesian networks, each node corresponds to an item, whilst a link from an item  $A$  to an item  $B$  signifies that the likelihood a user liking item  $B$  depends on her appreciation of item  $A$ . The underlying Bayesian network is produced looking at the statistical occurrences of item preferences in users' profiles and it is updated regularly to include the data associated with new ratings. Recommender systems built on Bayesian networks model the active users' ratings as known events. They subsequently infer the expected value of the item whose preference needs to be predicted by traversing the network. Bayesian networks are quite expensive to build, more importantly they lead to large super-exponential models that need to be traversed when making a prediction. Also, they typically have learning phases longer than memory-based methods, this resulting in a lag before changes in users' behaviour are reflected in recommendations [Breese et al., 1998]. For these reasons we have discarded their use in a mobile recommender system, which we predict will be limited in processing power and will constantly gather users data to learn from.

#### 2.1.3 Hybrid methods

Hybrid approaches are often used to mitigate the limitations inherent in each of recommending algorithm individually. For instance pure collaborative filtering overcomes the constraints of content-based methods by enabling to deal with any type of items and to recommend items, which are – content-wise – completely different from those previously rated. In turn, content-based methods address most of the issues inherent in collaborative systems, with the exception of the 'new user' problem, as also content-based algorithms require a user profile.

In the literature, seven approaches for integrating different recommending algorithms are

known [Burke, 2002]:

- Weighted
- Switching
- Mixed
- Feature combination
- Cascade
- Feature augmentation
- Meta-level

A weighted hybrid recommender is one in which all the recommending techniques available are used concurrently and their results are all pondered to obtain a final recommendation. On the contrary, in switching hybrid recommender, just one recommending algorithm is used at any time and is selected upon the current context. The mixed approach forms a list of recommendations by aggregating the top recommendations provided by various recommending algorithms implemented in the system. A radically different approach is used for feature augmentation: in this case, collective information and content data are combined in a augmented data set. To do so, the collaborative information (e.g., rating) is treated as any other content attribute and then content-based techniques are employed over this augmented data set. The cascade hybrid approach entails a staged process for which each recommender effectively acts as a filter to the subsequent one. In this technique, one recommendation technique is employed first to produce a coarse ranking of candidates and a second technique refines the recommendation from among this candidate set. In the meta-level approach the model learned by one recommender is used as input to another. For instance, very often to create dense rating matrices, items and/or users are clustered and collaborative filtering is used on the classes identified rather than the single entities.

Hybrid recommenders require implementing several approaches at the same time, thus further increasing the allocation of processing and memory resources.

Our research aims to explore the possibility of creating a lightweight recommender system for mobile environments whose precision and coverage are comparable to the ones of traditional recommenders in use today. We do not aspire at improving recommender systems for traditional environments. For these reasons, although we understand that hybrid recommenders may improve the coverage or precision of the recommendation made, we have overlooked these methods in the context of our research.

In the next section, we present *moRe*, a fully decentralised mobile recommender system, specifically designed, from the start, to run efficiently on mobile devices and be deployed in human pervasive networks. As our subsequent evaluation will demonstrate, *moRe* is capable of achieving this goal, whilst providing an accuracy and coverage comparable to those of centralised recommender systems in use today.

## 2.2 Challenges

In order to point out the main research challenges for recommending pervasive services and content, we analyse, from a recommendation perspective, one of the scenes of the scenario introduced in Chapter 1: *Scene Two - On the Tube to the Office* (Section 1.2.1). In the following we repeat the scene for convenience.

*Alice leaves home and walks to the nearest tube station to go to work. As she gets on the underground train, she takes an empty seat and switches on her MyGazine. Meanwhile her MyGazine has discovered a number of services and content available in the surroundings and has filtered them to just select the most appealing and relevant to Alice. As Alice switches on her MyGazine, the MyGazine creates for her a virtual magazine page that aggregates content and services according to Alice's tastes.*

*The services and content the MyGazine has selected for Alice are: 'Where is London Twitter?', eBay Local, the Smart Media Channel, the news previously downloaded whilst Alice had breakfast, some blog-posts published and shared by passengers in the surrounding, as well as the same Metro Magazine Information System embedded to all London stations and trains, some adverts beaconed by some of the advert banners on the train and the local business profiles of two plumbers in the surroundings*

This simple scene introduces a variety of problems that Mobile Recommender Systems will have to address.

First, it highlights the main issue considered in this chapter: Alice will come across a multitude of pervasive and mobile services, impossible to explore. She will hence need assistance. As discussed in the introduction, in recent years recommender systems have emerged as an affective tool to help users with a similar problem: information abundance. Current recommender systems, though, cannot be used as they are. In fact - as pointed out already - they fail to address the mobility of service providers and local connectivity. Mobile recommenders will have to overcome these weaknesses.

Whilst nowadays recommender systems leverage on one single global centralized view of a rating population (its opinions and habits), mobile recommender systems will lack such unique view. People we meet along the day through our personal devices will be part of the rating community that scores the services in our surroundings. In fact, when Alice

jumps on the train, she does not only discover new services (and/or providers), but also users that have experienced services she has already used or she may come across in the near future. On one side, this will require mobile recommenders to craft a dynamic, virtual view of users' ratings and the rating matrix. On another side, this could cause review information to be scarce, thus possibly aggravating the sparse rating matrix problem already affecting traditional recommenders.

Also, if recommender systems as we know them nowadays are provided by always-on, powerful and scalable systems, mobile recommender systems will have to run on constrained devices. This is because mobile devices are, by their very nature limited in terms of run time memory, processing power and especially battery. Hence, in order to be successful, mobile recommenders will have to optimise the use of resources, still providing recommendations whose precision and coverage is comparable with the ones of powerful recommender systems in use today.

Existing recommenders often consist of an online system and an offline engine. The offline engine does the heavy-lifting work and is used to analyse and process the information the recommender algorithm employed relies upon. The information so processed is pushed to the online engine, which calculates the actual recommendations. This allows optimizing the computations made at run time and the response times. In mobile devices there are no distinctions between offline and online environments. Everything happens on the same device and the same resources (CPU, heap memory, battery) are used to pre-process data and predict preferences. Mobile recommenders will need to optimize the processing of data in order to provide response times still acceptable to their users.

Finally, we remark that services and content will be used both in isolation and composed or aggregated. Whilst rating data will be available for common compositions and individual service/content, no information will be available for novel compositions or for services/content rarely used alone. This will aggravate the data rating matrix sparsity problem and the item cold start problem.

To address the concerns just outlined, we propose *moRe*, a *mobile Recommender* that isolates the problem of recommending items of local and mobile nature from the complexity of advising users on items of composite nature. To do so, *moRe* builds on a core mobile recommender system – *diffeRS* – that it then augments with a compositional one – *CReSy*.

In the following we discuss *moRe* looking at *diffeRS* and *CReSy* separately. First, we discuss the main Conceptual Model; in Section 2.3.1 we introduce *diffeRS* abstract data model and recommending strategy; while in Section 2.3.2 we focus on *CReSy* and describe how it enriches *diffeRS* for composite items.

Then in Section 2.4 we describe a concrete realisation in terms of data structures and algorithms for both *diffeRS* (Section 2.4.1) and *CReSy* (Section 2.4.2).

## 2.3 Conceptual Model

*moRe* is a fully decentralised recommender service that runs on users' mobile devices and that targets, as recommendable items, those services and content that are local and mobile with respect to their users. *moRe* consists of two main components: a core recommender service called *diffeRS* (*differential Recommender System*), that tackles the problem of recommending items of *local* and *mobile* nature; and a compositional recommender service called *CReSy* (*Compositional Recommender System*), built on top of the previous one, with the aim of specifically targeting items of *composite* nature (i.e., to predict users' preferences on novel compositions, based on the information collected on their components, and vice versa). In the following, we first present *diffeRS* (Section 2.3.1), the abstract data model underpinning it and its recommending algorithm; we then introduce *CReSy* (Section 2.3.2) and highlight how it enriches *diffeRS* to explicitly handle compositions.



Figure 2.2: *moRe* Overview

### 2.3.1 Core Recommender Service: *diffeRS*

*diffeRS* (*differential Recommender System*) is a fully decentralised mobile recommender, specifically designed for mobile scenarios. It runs on users' devices and optimizes their resources still achieving accuracy comparable to the centralized recommender systems in use today. On one side, it targets, as recommendable items, services and content that are pervasive, local and mobile. On the other side, it is intended to run on constrained devices such as mobile devices. To do so, *diffeRS* builds upon the following three observations.

**Observation 1** – *Rating information for pervasive services and content is meaningful only to people that are (or will be) in the same surroundings. It does make sense then to distribute rating information only to local users, rather than broadcasting it multi-hop across all users.*

As highlighted in Section 2.2, on one side recommenders for pervasive services will lack a centralized view of the rating population and associated rating matrix; on the other side, because of their local nature, that information will be relevant to just the users that will ever be in the proximity of those services. We therefore let users maintain their personal rating profile on their personal device, and exchange their rating profiles upon ‘encounters’,

where by ‘encounter’ of two users we mean the event connecting users’ personal devices locally with a one-hop distance. By exchanging profiles, users end up collecting the ratings of people they have been co-located with, at least once. So they create a virtual view of the rating matrix for the services they either have experienced or may experience.

**Observation 2** – *Any rating community can be modelled as formed by two and fundamentally different groups of users: mass-like minded users and individual users (i.e., users that have rather atypical tastes). As Mark Penn remarked [Penn and Zalesne, 2007], the tastes of mass-like minded users (macro-trends) are easy to notice and do not necessarily require clever engines to highlight and exploit them. On the contrary, patterns within the preferences of individual users (micro-trends) are counter-intuitive, difficult to discover and have the potential to elicit behaviours otherwise extremely difficult to understand and guess. Hence, recommender systems should focus on comprehending and predicting the tastes of atypical users.*

The second idea develops from Mark Penn’s remark that ‘the biggest trends are the micro-trends’ [Penn and Zalesne, 2007]. Often macro-trends are evident and do not necessitate sophisticated systems to learn and exploit them. On the other hand, micro-trends are counter-intuitive, very difficult to spot and therefore to use. More importantly, micro-trends have the ability to explain and anticipate events and patterns otherwise inexplicable or unpredictable.

We use the distinction between micro-trends and macro-trends for inspiration to classify users as either mass-like minded or atypical. In fact, similarly to macro-trends, in the recommender systems world the tastes of mass-like minded users are easy to notice and do not necessarily require clever engines to highlight and exploit them; on the contrary, patterns within the preferences of individual users are difficult to discover. Also, as micro-trends, they have the potential to make clear and foresee behaviours otherwise extremely difficult to understand and predict. Hence, the intuition is that core to recommenders is the ability to understand and predict the tastes of atypical users.

**Observation 3** – *The more frequently a user’s preferences deviate from those of the rating community she belongs to, the less accurately such community represents her, and vice-versa. Paradoxically, in traditional recommender systems, popular items and trend followers risk driving predictions for atypical users too. This is because items of universal appeal are those for which more users’ feedback exists (i.e., they appear in the profiles of many users, and thus drive the quantification of users’ similarity as the only items in overlap).*

We observe that most of user collaborative filtering methods reason about users similarities and they do so for all users within a rating community. Although this may work well

in terms of overall performance, this fails to deal with users whose tastes are singular and different to the rating community they belong to. In fact, we observe that in almost any catalogue a non-negligible number of items exists for which users, regardless of their actual tastes, share a similar appreciations. For instance, in the music domain, both fans of Britney Spear, Green Day, The Muse and The Cinematic Orchestra would agree in liking the Beatles, nevertheless they have very distinctive tastes. Similarly in the grocery world, both junk food lovers, vegetarians, meat fanatics and sophisticated food lovers would like bananas.

Users with peculiar tastes – as anyone else – consume and rate items of universal appeal (e.g., Beatles’ ‘Help’). However they also use peculiar items, which have been experienced and hence rated by very few other users in the rating community (e.g., Cinematic Orchestra’s ‘To build a home’ [Orchestra, 2007]). Because items of universal appeal are the items for which more users’ feedback exists, they also are the most frequent in users’ overlapping sets of co-rated items. The consequence is that, paradoxically, popular items and trend followers risk driving predictions for atypical users.

We therefore propose a new collaborative filtering approach to estimate users’ preferences that distinguishes atypical users from trends followers.

diffeRS exploits these observations as follow: first, rating profiles are only exchanged between colocated people (i.e., at one hop wireless distance) during encounters (observation 1); a virtual view of the local community is then built on each device running diffeRS.

Second, users are dynamically distinguished between mass-like minded and individual users, simply by looking at the average deviation of their profile from the preferences expressed by the community as a whole (observation 2).

Finally, for mass-like minded users, predictions are simply computed as the average of the preferences expressed by the rating community they belong to. For atypical users, instead, a user-based Collaborative Filtering approach is used; however, recommenders are not searched within the whole rating community but only among those other atypical users who most similarly deviate from the rating community, that is, those users who are *similarly different* (observation 3). The details of how diffeRS leverages these observations, in terms of abstract data model and algorithm, are presented next.

### Abstract Data Model

The data model used within *diffeRS* builds upon the observation, just made, that ratings on pervasive services are meaningful just to users that share the same environments. In fact, *diffeRS* collects just rating information about people that, at any time, have been co-located. To do so, in *diffeRS* a user profile is a vector of ratings expressed that users own and store in their personal devices. Whenever two terminals encounter, their users’ vectors

are exchanged, so that each user builds, over time, a local rating matrix  $P$  containing the ratings of other users in the surroundings, which approximates the global rating matrix used in traditional recommenders.

Unlike conventional recommenders, *diffeRS* does not exploit the local rating matrix  $P$  for computing recommendations, but uses it solely as a repository for the rating information collected. This is because  $P$  stores data about users that are very different in nature: mass-like-minded users (or trend followers)  $U^M$  that assent with the rating community and individuals (or atypical users)  $U^I$  who do not.

As they are in agreement with the rating community, mass-like minded users are fairly represented by it. Consequently, the community mean ratings should predict reasonably well their preferences and the advantage of using for them any sort of collaborative filtering should be limited. We can then avoid to use  $P$  to predict recommendations for mass-like minded users. Just a vector collecting the item average ratings should suffice. We call this vector Community Profile  $u_C$ .

Instead, individual users are by definition in disagreement with the rating community. As mass-like minded users agree with the community, then for transitive property, individual users dissent with them too. It hence makes very little sense to seek advisers for individual users among trend followers. They are radically different. When predicting ratings for individual users, we thus exclusively seek the advice from other atypical customers; we do so by exploiting a sub-matrix  $P^I$  of the local rating matrix  $P$  that contains the preferences expressed by users in  $U^I$  only. As a result, *diffeRS* conceptually decomposes a rating matrix  $P$  in a vector, the Community Profile  $u_C$  (used to predict preferences for  $U^M$ ), and a significantly smaller rating matrix  $P^I$  (used to predict preferences for  $U^I$ ), as shown in Fig. 2.3.

This results in much less data being loaded in memory to compute recommendations.

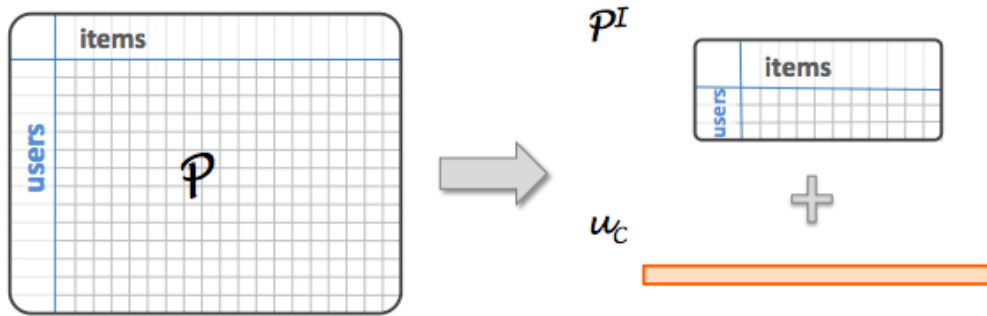


Figure 2.3: *diffeRS* Rating Matrix Decomposition

### Recommending Algorithm

The mechanics behind Recommender Systems are quite simple in principle. Whenever a user is looking for a recommendation, the interrogated Recommender System computes

for each eligible item a score that reflects how likely the user would enjoy it; items are then ranked based on this score, and the top (or the top- $k$ ) items are recommended. Hence, the real key challenge of a Recommender System is how to compute similarity scores.

Various methods have emerged, with Collaborative Filtering becoming increasingly popular and becoming an industry de-facto standard. The intent of our research is not to create a radically new recommender method, but rather to investigate how the same uptake we witness nowadays on the Web for recommender systems could be replicated in mobile environments; as such, *diffeRS* is based and built upon the same technology that has driven the popularity and success of recommenders: Collaborative Filtering, more precisely user-based Collaborative Filtering (CF). Being  $u_a$  the profile of active user  $a$ , for whom a prediction for item  $i$  needs to be computed, traditional user-based CF performs three steps: compute users' similarity to  $a$  for all known users (step 1), determine  $a$ 's neighbours (i.e., recommenders) as those with higher similarity (step 2), and compute a predicted rating for item  $i$  based on the preferences expressed by  $a$ 's neighbours for  $i$  (step 3).

*diffeRS* proposes two key changes to this approach: as a pre-step (step 0), it determines whether  $a$  is a mass-like minded user or an individual one. It does so by calculating  $u_a$ 's average deviation  $d_a$  from the community profile vector  $u_C$ ; if the deviation is lower than a constant parameter  $\alpha$ ,  $a$  is deemed to be mass-like minded, so that the predicted rating for item  $i$  is simply the community average opinion  $u_C[i]$ . If the user is deemed individual, the three-step user based CF is performed instead, but with a major difference: rather than using the matrix  $P$  (i.e., all users and all ratings), it uses the smaller rating matrix  $P^I$ . In so doing, it computes the similarity between  $a$  and atypical users only, and it derives a recommendation based on their opinions alone. The computational complexity of *diffeRS* is thus considerably reduced, as heavy calculations (i.e., users' similarity) are performed exclusively for 'difficult to predict' people, and using considerably smaller amounts of data.

Algorithm 1 provides an overview of *diffeRS* rating prediction algorithm. A more detailed description of its steps follows.

**Step 0 - User Classification.** The idea underpinning the method to discriminate mass-like minded users from individual ones develops from the observation that the fact users share or not a similar appreciation (or dislike) with their rating community is indicative of their agreement or disagreement with it (see Observation 3). So, each preference conveys some information on the degree at which a user conforms to the rating community she is part of.

Intuitively, the more and the more frequently users discord with the rating community, the more they would differ from it. Hence, if we indicate with  $u_a$  the profile of user  $a$ ,  $u_C$  the community profile, and  $I_a$  the set of items for which  $a$  has expressed a preference, then we can classify a user as being mass-like minded or atypical by quantifying its *deviation* from the community. More precisely, we compute the average of the absolute difference

---

**Algorithm 1** Rating Prediction Algorithm

---

**Require:**  $u_C$  and  $P^I$  have already been calculated**Input Parameters**

- $u_a$  the profile of the active user;
- $i$  item to estimate a preference for;
- $\Delta_r$  maximum rating difference in a given rating scale;
- $I$  set of all rated items;
- $\bar{r}_x$  average rating for user  $x$ .

**Returns:** rating  $r_{a,i}$ 

{Step (0) - User Classification}

$$d_a = \frac{1}{|I_a|} \sum_j |u_a[j] - u_C[j]|$$

**if**  $d_a \leq \alpha$  **then**    return  $u_C[i]$ **end if**

{Step (1) - Computing User Similarities}

**for all**  $k \in U^I, k \neq a$  **do**

$$w_{a,k} = \frac{1}{|I_a \cup I_k|} \sum_{j \in I_a \cap I_k} \left( 1 - \frac{|u_k[j] - u_a[j]|}{\Delta_r} \right)$$

**end for**

{Step (2) - Selections of Neighbours}

Neighbourhood =  $U^I \setminus \{a\}$ 

{Step (3) - Weighted Average }

$$r_{a,i} = \bar{r}_a + \frac{\sum_{k \in \text{Neighbourhood}} w_{a,k} * (u_k[i] - \bar{r}_k)}{\sum_{k \in \text{Neighbourhood}} w_{a,k}}$$

return  $r_{a,i}$ 

---

between the ratings that  $a$  has expressed ( $u_a[j]$ ) and the community feedback for the same items ( $u_C[j]$ ). We so obtain a measure of users' individuality that is independent on the number of items rated by the user:

$$d_a = \frac{1}{|I_a|} \sum_j |u_a[j] - u_C[j]| \quad (2.1)$$

If the deviation  $d_a$  for a given user  $a$  is lower or equal to a constant parameter  $\alpha \in [0, \Delta_r]$ , differS classifies the user as mass-like minded, otherwise as individual (with  $\Delta_r$  being the maximum difference between two rating values in a given rating scale). In the experiments presented in Chapter 3 we use  $\alpha = 1$ .

**Step 1 - Computing User Similarities.** If a user is deemed individual, the next step is to find suitable recommenders. Unlike traditional correlation models, that do so by finding similar rating behaviours within the community as a whole, here we ponder the *differences* that users manifest against only other atypical users instead. In other words, we look at how other individual users *similarly differ* from the rating community.

This requires us to quantify the similarity  $w_{a,k}$  between the active user  $a$  and other *individual* users  $k \in U^I$  as follow: first, we compute the difference in their ratings for each item  $j$  they have co-rated  $|u_k[j] - u_a[j]|$ ; we then sum these differences (normalised by the maximum differences between two rating values in the rating scale  $\Delta_r$ ) to infer a measure of users' correlation that boosts 'similarly different' tastes. Finally, we use the Jaccard index  $\frac{|I_a \cap I_k|}{|I_a \cup I_k|}$  to weight the reliability (or confidence) of the computed similarities. Being  $I_a$  and  $I_k$  the set of items rated by  $a$  and  $k$  respectively, and  $\Delta_r$  the maximum rating difference, we thus compute the similarity between users  $a$  and  $k$  as:

$$w_{a,k} = \frac{1}{|I_a \cup I_k|} \sum_{j \in I_a \cap I_k} \left( 1 - \frac{|u_a[j] - u_k[j]|}{\Delta_r} \right) \quad (2.2)$$

The idea underpinning this users' similarity measure is simple. Individuals have peculiar tastes and strike from the mass as distinct personalities. Their ranking behaviours may be very personal and different from anyone in the community. Therefore, when comparing unusual users, it may make little sense to look for comparable ranking patterns. Instead, we seek out similarities in the appreciation of each item co-rated and we do so by looking at how they differ from the rating community. For these reasons, we do not use correlation models that aim to highlight similar rating behaviours, like Cosine or Pearson correlations, but we ponder the actual differences users manifest against the same rating community in ranking the same items. More precisely, for each item two users have graded, we look how similarly or differently to the community they have rated the same items. If they both have liked it more (or less) than the rating community, we compute their disagreement simply as the absolute difference in appreciating it; otherwise, if they have opposite opinions compared to the rating community (i.e., one user enjoyed more than the average, whilst the other graded it less), we sum the differences of judgement in relation to the mass. In other words we calculate the difference between user  $u_a$  and user  $u_u$  in rating an item  $i$  as follow:

$$w_{a,k}(i) = \begin{cases} ||u_a[i] - u_C[i]| - |u_k[i] - u_C[i]| | \\ \text{if } (u_a[i] - u_C[i])(u_k[i] - u_C[i]) \geq 0 \\ |u_a[i] - u_C[i]| + |u_k[i] - u_C[i]| \\ \text{otherwise} \end{cases}$$

This is equivalent to:

$$w_{a,k}(i) = |u_a[i] - u_k[i]|$$

Intuitively the less frequently two users disagree in ranking the same items, the more similar they are. Hence, we use the difference between users in ranking the same items – normalised by the maximum differences between two rating values in the ranking scale – to infer a measure of users' correlation that boosts similarly different users.

We remark that the computation of user-user correlations only considers products that both users have rated; hence, even if two users have co-rated only one single product they may have maximum correlation if they have rated identically the same product. Clearly, correlations being based upon few data-points only, are not very reliable. Therefore we use Jaccard correlation as a significance weighting to weight the same similarities. The Jaccard correlation measures the overlap that two vectors share in regards to the item rated. So, it provides both a measure of the reliability of similarity coefficient as well as measure of similarity. In fact, the fact itself that users – regardless of their appreciation – have been consuming the same items indicate that they are exposed to the same set of services and/or share a similar knowledge of the item catalogue.

As a result, being  $I_a$  and  $I_k$  the set of items rated by  $u_a$  and  $u_k$  respectively,  $\Delta_r$  the maximum rating difference, we compute the similarity between users as:

$$w_{a,k} = jaccard(a, k) \frac{\sum_{i \in I_a \cap I_k} (\Delta_r - |u_k[i] - u_a[i]|)}{\Delta_r |I_a \cap I_k|} \quad (2.3)$$

This can be also expressed as:

$$w_{a,k} = jaccard(a, k) \frac{\sum_{i \in I_a \cap I_k} \left(1 - \frac{|u_k[i] - u_a[i]|}{\Delta_r}\right)}{|I_a \cap I_k|}$$

If we expand the Jaccard correlation to:

$$jaccard(a, k) = \frac{|I_a \cap I_k|}{|I_a \cup I_k|}$$

Then:

$$w_{a,k} = \frac{1}{|I_a \cup I_k|} \sum_{i \in I_a \cap I_k} \left(1 - \frac{|u_k[i] - u_a[i]|}{\Delta_r}\right)$$

We so obtain a correlation formula that is lightweight and consists of sums rather than products.

**Step 2 - Selections of Neighbours.** In traditional user-based CF systems, neighbours for an active user  $a$  are selected by either considering the  $k$  nearest neighbours (i.e.,  $k$

most similar users) or by considering all users whose similarity to  $a$  is greater than a given threshold. We take a slightly different approach and consider as neighbours *all individual users* who have rated at least one item in common with the active user  $a$ . This is because, as introduced in Section 2.2, one of the key challenges that mobile recommenders will have to address is the potential scarcity of review information, thus possibly aggravating the sparse rating matrix problem already affecting traditional recommenders. Therefore, when selecting users' neighbours, *diffeRS* has to address and compensate for these challenges: (i) the sparse rating matrix problem and (ii) the constrained memory, processing and power energy of mobile devices. On one side, the larger is the information available to make a prediction, the better; on the other side, the amount of data processing should be limited. The approach we take is to use *just* key information, but to use it *all*. Therefore *diffeRS* targets these problems (i) by using *just individual users'* ratings, but (ii) by considering *all individual users* that have expressed a preference for an item rated by  $u_a$  as  $u_a$ 's neighbours, and by (iii) applying a lightweight correlation formula as discussed in Step 1.

**Step 3 - Weighted Average.** Finally, the predicted rating of the active user  $a$  for item  $i$  is estimated using a weighted average:

$$r_{a,i} = \bar{r}_a + \frac{\sum_{k \in \text{Neighbourhood}} w_{a,k} * (u_k[i] - \bar{r}_k)}{\sum_{k \in \text{Neighbourhood}} w_{a,k}} \quad (2.4)$$

where  $\bar{r}_k$  is the average rating for user  $k$ .

Note that *diffeRS* is agnostic with respect to the nature of the item it recommends: it could be a simple service  $i$ , or indeed be the composition of several others  $i = \{i_1, i_2, \dots, i_n\}$ . In either case, *diffeRS* solely relies on the feedback available about  $i$  as a whole, to predict how much user  $a$  will enjoy it. If feedback about its individual services  $i_1, i_2, \dots, i_n$  is available within the community, but no rating about the composite service  $i$  has ever been given (i.e., it is the first time such mash-up is consumed), *diffeRS* is not capable of predicting how much user  $a$  will enjoy the composition. To cater for these scenarios, we propose a compositional recommender service on top of *diffeRS*.

### 2.3.2 Compositional Recommender Service: CReSy

CReSy is a compositional recommender service, built on top of *diffeRS*, specifically designed to compute predictions about novel compositions of services and mash-ups of any sorts. CReSy runs on users' devices and optimizes their resources still achieving a good accuracy. On one side, it targets, as recommendable items, services and content that are the result of unusual combinations or that are seldom used in isolation. On the other

side, it is intended to run on constrained devices such as mobile devices. To do so, CReSy builds upon the following observations.

**Observation 1** – *The appreciation of a composite item is intrinsically bound to the appreciation for its components.*

Composite items – i.e., composite services and mash-ups – combine together a set of services and/or content to offer rich and diverse functionalities or information. Supposedly, each of the component service or content adds to the composite item; so, the performance of each component plays a role in the perception of the overall service or mash-up. Hence, we argue that – to some extent – (i) the preference for a composition derives from the ones of its components and (ii) the feedback expressed on a composition is also feedback to its components.

We refine this first intuition and observe that:

**Observation 2** *Composite items can be modelled as teams.*

We remark that a composite item can be very simply thought of as a collection of varied items united to provide a greater functionality. In real life, this resembles the concept of a team. In fact, a team is a group of diverse individuals working together toward a common goal: winning a tournament, delivering a project, etc. Also, teams and composite items are similar in that the reputation of a group is often linked to the reputation of its single elements and vice-versa.

CReSy builds on these observations and provides (i) a method to predict a user’s preference for a composite item if her feedback on some of its components is known (Recommending Composite Items) and (ii) a method to estimate a user’s appreciation for an individual item, starting from the user’s perception of the compositions the item has been an element of (Recommending Elementary Items).

In the following, we discuss the conceptual model underpinning these two methods.

### Recommending Composite Items

As the performance of a team does not depends solely on the ability of its members, but also on how well they work together, so a user’s perception of a composite service derives from both the component services and their actual combination.

CReSy builds on this observation and adopts a Team Performance model that allows to carter for both the quality of its elements and their synergy; this is the AM (Ability, Motivation) framework [Boxall and Purcell, 2003]. According to this model, the performance

$P$  of a team can be expressed as a function  $P = f(A, M)$ , where  $A$  depicts the quality of its members (i.e., their *ability*), while  $M$  denotes a collective harmonising drive (i.e., their *motivation*). CReSy adapts the AM model to suit a compositional recommender service as follow: it maps components to team members and compositions to teams; team performance thus becomes a measure of a user's preference for a mash-up.

More precisely, CReSy interprets *ability* as the qualities of component services and *motivation* as the value derived from their aggregation.

Let us consider, for example, the mash-up ‘Where is London Twitter?’ introduced in Section 1.2. Such composite service combines Google Maps<sup>1</sup> and Twitter<sup>2</sup> services to show the geographical Twitter penetration on a map. Traditional recommender systems, including diffeRS, would only be able to predict how much a user would enjoy the mash-up ‘Where is London Twitter?’ if feedback about it, as a whole, existed. CReSy, instead, models the ‘Where is London Twitter?’ mash-up as a team, and Google Maps and Twitter as its members; so, if feedback about (a subset of) such members is available, CReSy can predict a user's preference for their composition. This task is not as straightforward as it may appear at first, and simply returning the average of individual members' predictions results in high prediction errors overall, as our evaluation shall demonstrate (e.g., the fact that a user may not particularly appreciate Twitter does not necessarily entail a negative feedback on ‘Where is London Twitter?’).

As this simple example shows, when it comes to users' perception of composite services, the effect of ability and motivation is not obvious. To depict it, we use the model of the Diversity Prediction Theorem<sup>3</sup> [Bray et al., 2008] [Krogh and Vedelsby, 1995] [Leamer, 1978], which states that:

$$Crowd\ Error = Average\ Error - Diversity \quad (2.5)$$

Now, we can model an *Error* as the difference between a perfect performance (*Perfection*) and the actual *Performance* (i.e.,  $Error = Perfection - Performance$ ). Consequently, the Diversity Prediction Theorem can also be expressed as:

$$Perfection - Crowd\ Performance = Perfection - Average\ Performance - Diversity \quad (2.6)$$

---

<sup>1</sup><http://maps.google.com/>

<sup>2</sup><http://twitter.com/>

<sup>3</sup>The version of the theorem that we adopt follows Krogh, A. and Vedelsby, J. 1995. “Neural Network Ensembles, Cross Validation, and Active Learning.” In *Advances in Neural Information Processing Systems* 7. Edited by G. Tesauro, D. S. Touretsky, and T.K. Leen, 231-38. Cambridge, MA: MIT Press. For more general background see Leamer, E. (1978) *Specification Searches - ad hoc Inference with Nonexperimental Data*. New York: John Wiley and Sons.

If we then subtract *Performance* to both members and change sign, the same becomes:

$$\text{Crowd Performance} = \text{Average Performance} + \text{Diversity} \quad (2.7)$$

So, we paraphrase the Diversity Prediction Theorem to state that the collective ability of any crowd is equal to the average ability of its members *plus* the diversity of the group; in other words, we map *motivation* (in the AM framework) to *diversity* in Equation 2.7. The intuition behind this mapping is that human curiosity (and disinterest which arises due to monotony) tends to favour novel, heterogeneous compositions. As we shall demonstrate by means of simulation using real datasets relating to human infotainment, predictions about composite items achieve high degree of accuracy when using the Diversity Prediction Theorem rather than relying on simple item averages, thus backing up this intuition.

Based on this interpretation of the Diversity Theorem, within the scope of a compositional recommender service, the predicted preference of composite item  $c$  by user  $a$  can be expressed as follow:

$$p(a, c) = \overline{u_a[i]}_{i \in I_c} + \sigma(u_a[i])_{i \in I_c} , \quad (2.8)$$

with  $I_c$  being the set of items making up composite item  $c$ , and  $u_a[i]$  the feedback that the active user  $a$  has expressed on a component  $i \in I_c$ . If feedback  $u_a[i]$  is known about a subset only of the elementary items  $i \in I_c$ , the above equation is still applied based on this partial knowledge.

### Recommending Elementary Items

So far we have discussed a model that predicts user  $a$ ' preferences for a composition  $c$ , starting from preferences about individual items ( $u_a[i]$ ). For completeness, we also look at the opposite case (i.e., estimate the rating of an item from the score of the compositions it is an element of). It is in fact possible that, although a user has experienced a service (or a content source) in compositions (or a mash-up), she has never experienced it in isolation and a need may arise to predict her inclination for it. For instance, this could be the case of an innovative service that needs to gain popularity before being proposed to a wide audience as a self-standing service. That was, for example, the case of Shazam<sup>4</sup>. Although today Shazam is the world leading music discovery service and it brands its own service and applications, in its earlier days Shazam was powering music download applications and portals for few Telecom Operators (e.g., Vodafone, T-Mobile and AT&T, as a white-label service provider). When Shazam started launching its discovery service alone, traditional recommender systems would have failed to predict users' preferences for it, although many people already experienced it (in combination to other services).

We argue that the feedback expressed on applications empowered by Shazam is itself

---

<sup>4</sup><http://www.shazam.com/>

feedback to the Shazam service. Hence, we propose to ponder the ratings the user assigned on the composite items to infer a preference on the component item.

More precisely, if we indicate with  $C^i$  the set of compositions an item  $i$  has been part of, and with  $u_a[c]$  the feedback expressed by the active user  $a$  on the compositions  $c \in C^i$ , then we estimate the preference of an active user  $a$  for the constituent item  $i$  as:

$$p(a, i) = \frac{1}{|C^i|} \sum_{c \in C^i} u_a[c] \quad (2.9)$$

In other words, item  $i$  automatically receives a rating which is the average of the preference expressed on all the mash-ups it is an element of. From a Team modelling perspective, the intuition supporting this model is that the success of an individual is inextricably bound to the success of the teams she contributes to.

The conceptual framework that we so obtain is depicted in Figure 2.4. In there, arrow 1 and arrow 2 depict conceptually the inference processes for recommending composite items and recommending elementary items, respectively.

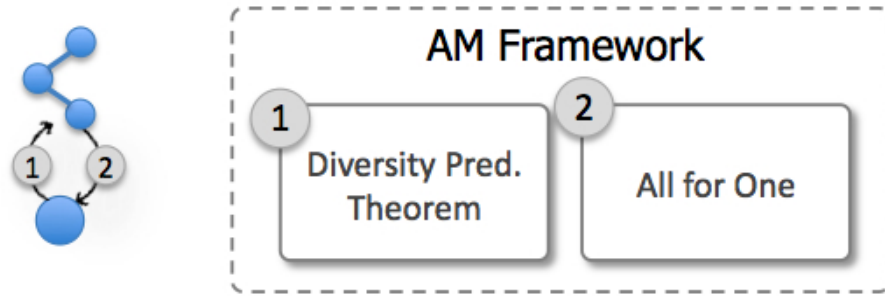


Figure 2.4: CReSy Conceptual Framework

## 2.4 Concrete model

moRe targets mobile devices that are, by nature, constrained in terms of principal memory and processing power, relative to the platforms where traditional recommender systems run. Even if technological advances will soon result in larger memory and faster processors, battery remains a bottleneck (i.e., Moore's Law will apply to the miniaturisation of battery size, rather than increasing its lifetime [Paradiso and Starner, 2005]). Therefore, it becomes crucial to optimise mobile recommender systems to reduce primary memory allocation and processing, thus reducing power consumption as a side effect. In this section, we discuss how the models introduced in Section 2.3 have been mapped to an efficient realisation.

### 2.4.1 diffeRS

#### Data Structures

In order to represent a rating matrix (be that the global one  $P$ , or the individual one  $P^I$ ) in a compact format, as well as to speed up the computation of users' similarities, we have developed an ad-hoc data structure that we call *Rating Hash Tree*. A Rating Hash Tree organizes users' ratings by items rated *and* by distinct rating values, as depicted in Figure 2.5. For example, if the set of possible rating values is  $R = [1, 5]$ , then the Rating Hash Tree associates to each item  $i \in I$ , a maximum of  $|R| = 5$  sets (which we refer to as  $P_{i,r}$ ), each collecting the IDs of those users who have rated item  $i$  with value  $r \in R$ . For example in Figure 2.5, item  $i_y$  has received one rating  $r = 1$  by user *Alice*, two ratings  $r = 2$  (one of which by user *Bob*), and two ratings  $r = 4$  (one of which by user  $u_x$ ).

In practice, the Rating Hash Tree is an hash map of hash maps, where the root hash map (Level 0 Hash Map) associates item IDs to item objects in an array<sup>5</sup>, the Level 1 Hash Maps relate items to rating maps, and the ratings maps (Level 2 Hash Maps) associate each rating value to the users who have rated the related item with such value. Due to the levelled nature of the overall structure, we have called this structure Hash Tree.

This structure is in some aspects similar to the Bitmap Index and Inverted Files structures used in Information Retrieval (IR) [Zobel and Moffat, 2006]. In fact, like a Bitmap Index, the Rating Hash Tree groups information by repetitive values (i.e. the rating values). Also, similarly to Inverted Files, the Rating Hash Tree organises information per key values (in our case rating values) that reference files (in our case users profiles). The similarities between the Rating Hash Tree and the Bitmap Index and Inverted Files though concern only Level 2 Hash Maps.

diffeRS exploits the Rating Hash Tree structure to represent both  $P$  and  $P^I$ . We call the Rating Hash Tree representing  $P$  as *Rating Hash Tree Repository*; in practice, this is persisted on the device secondary memory as a collection of small files (one for each  $P_{i,r}$ ), grouped in folders (one for each item  $i$ ). We then call the Rating Hash Tree for  $P^I$  as the *Individuals' Rating Hash Tree*; this is persisted on secondary storage if the user is (currently) deemed as mass-like minded; in this case, only the Community Profile  $u_C$  is kept in primary memory. If the user is deemed atypical instead, the Individuals' Rating Hash Tree is loaded in primary memory, where it is represented as a chain of hash maps.

This structure is much more compact than the traditional storage of *user-item-rating* triplets. More importantly, the Rating Hash Tree enables the optimisation of the number of calculations made to quantify user similarities: for any given item  $i$ , in fact, divergence with respect to the active user  $a$  is computed *once* for all individual users who have

<sup>5</sup>For clarity we do not depict the root hash map in Figure 2.5, instead we directly depict the referenced array.

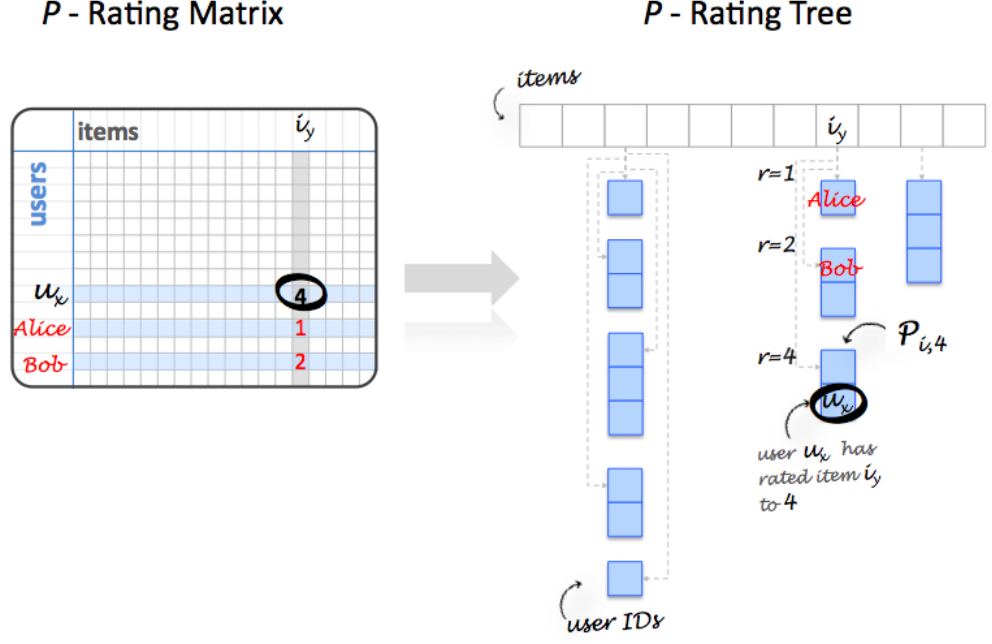


Figure 2.5: Rating Hash Tree

expressed the same rating  $r$  (i.e., once for all users in  $P_{i,r}$ ).

Whenever an encounter occurs, which results in a new (or updated) profile  $u_{new}$  becoming locally available, these data structures are processed as follow (Algorithm 2):

1. The profile  $u_{new}$  is stored in the Rating Hash Tree Repository  $P$  in secondary storage; new item averages, for each item  $i$  rated in  $u_{new}$ , are calculated, thus resulting in a new community profile  $u_C^{new}$ ;
2. Users' deviations from the new Community Profile are computed, which may result in changes in  $U^M$  (trend followers) and  $U^I$  (individuals). Note that this step does not require the re-computation of Equation 2.1 for all users. Rather, we observe that the only users whose deviation from the mass may change are those who have rated at least one item in common with  $u_{new}$ . For these users alone, their deviations from the community are incrementally re-assessed (i.e., the difference between  $|u_k[j] - u_C^{old}[j]|$  and  $|u_k[j] - u_C^{new}[j]|$  are computed and added to the old user  $k$ 's deviation). As previously highlighted, the structure of the Rating Hash Tree enables the computation of these incremental differences just once for all users that have rated item  $i$  with the same rating value  $r$ , thus further reducing the required calculations;
3. Finally, for each user whose deviation from the community has changed, a re-classification is performed, which may require some profiles to be removed from the Individuals' Rating Hash Tree (if users have now become trend followers), and

some others to be copied from the Rating Hash Tree Repository to the Individuals' Rating Hash Tree (for users who are now classified as individuals).

### Recommending Algorithm

Traditional recommender systems pre-compute users' similarities and neighbourhoods in heavy-weight batch processes. As resources (CPU, heap memory and battery) are scarce on mobile devices, *diffeRS* uses a lazy approach instead, whereby users' similarities (Equation 2.3) are computed only on-demand (i.e., when a recommendation for an atypical user is requested); these similarities are then locally cached until updates to the local Individuals' Rating Hash Tree invalidate them.

More precisely, *diffeRS* first compares the active user's profile  $u_a$  against the Community Profile  $u_C$ , to determine if  $a$  is a trend follower. If so, it estimates the preference for an item  $i$  with its average rating, otherwise it uses the preferences expressed for item  $i$  by other atypical users. If  $a$ 's neighbours (and their similarity scores) are already stored in the local cache, *diffeRS* simply uses them; otherwise it computes them while also calculating preference predictions, as depicted in Algorithm 3. As shown, the sets  $P_{i,r}$  are traversed first, to construct a super set  $U_a$  of possible neighbours to user  $a$  (step 0). Users' similarities  $\overline{w_{a,k}}$  are then *incrementally* computed (step 1), based on an equivalent form of Equation 2.3:

$$\begin{aligned}
 w_{a,k} &= \frac{1}{|I_a \cup I_k|} \sum_{j \in I_a \cap I_k} \left( 1 - \frac{|u_k[j] - u_a[j]|}{\Delta_r} \right) \\
 &= \frac{|I_a \cap I_k|}{|I_a \cup I_k|} - \frac{1}{\Delta_r |I_a \cup I_k|} * \overline{w_{a,k}}, \text{ with} \\
 \overline{w_{a,k}} &= \sum_{j \in I_a \cap I_k} |u_k[j] - u_a[j]|
 \end{aligned} \tag{2.10}$$

For each item  $j$  rated by user  $a$ , the sets  $P_{j,r}, \forall r \in R$  are traversed; while doing so, the differences between  $a$ 's rating of item  $j$  and that of any user  $k \in P_{j,r}$  (i.e.,  $|r - r_{a,j}|$ ) are computed. Once the traversal is complete, the similarity measure between  $a$  and users in  $P_{j,r}$  is also available. Finally, the ratings provided by  $a$ 's neighbours are weighted by their similarities and averaged, as per Equation 2.4 (step 3).

Figure 2.6 exemplifies the recommending process to predict the preference of an atypical user Alice on an item  $i$ . The sets  $P_{i,r}, \forall r \in R$  are traversed first, to identify those atypical users who have rated item  $i$  (e.g., Bob). These users represent Alice's neighbours (step 1 in the figure). For each such user, *diffeRS* computes their similarity to Alice as per Equation

---

**Algorithm 2** Preference Update Algorithm

---

**Require:**  $P, P^I, I, U^I, u_C, R$  and  $\alpha$  defined

- $P$  the Rating Hash Tree Repository;
- $P^I$  the Individuals' Rating Hash Tree;
- $I$  set of items in the catalogue;
- $U^I$  the set of atypical users;
- $u_C$  the Community Profile;
- $\alpha$  the threshold value of the deviation from the mass.

**Note :**  $d_k$  is the deviation of user  $k$  from the community.**Input Parameters :** new user profile  $u_{new}$ .{Step (1) - Update of Community Profile and  $P$ } $U^{delta} = \emptyset$  $u_C^{new} = u_C$ **for all** item  $i$  rated by  $u_{new}$  **do** $r = r_{u_{new},i}$  $P_{i,r} = P_{i,r} \cup u_{new}$  $U^{delta} = U^{delta} \cup \{u_k | u_k \in P_i\}$  $u_C^{new}(i) = \frac{|P_i| * u_C(i) + r}{|P_i| + 1}$ **end for**

{Step (2) - User Classification Update}

 $U^{I-}, U^{M-} = \emptyset$ **for all** items  $i$  rated by  $u_{new}$  and any distinct value  $r$  **do** $\forall u_k \in P_{i,r}, d_k = d_k + \text{variation between } |r - r_{C,j}| \text{ and } |r - r_{C,j}^{new}|$ **end for** $U^{M-} = \{u_k | u_k \in U^{delta} \wedge u_k \notin U^I \wedge d_k > \alpha\}$  $U^{I-} = \{u_k | u_k \in U^{delta} \wedge u_k \in U^I \wedge d_k \leq \alpha\}$ {Step (3) -  $P^I$  Update and Cleansing} $u_C \Leftarrow u_C^{new}$ **if**  $d^{new} > \alpha$  **then** $P_{i,r}^I = P_{i,r}^I \cup u_{new}$ **end if** $U^I = (U^I - U^{I-}) \cup U^{M-}$ remove  $u_k \in U^{I-} \forall P_{i,r}^I$  $\forall P_{i,r}^I, P_{i,r}^I = P_{i,r}^I \cup \{u_k \in P_{i,r} \wedge u_k \in U^{M-}\}$ return

---

---

**Algorithm 3** Rating Hash Tree Traversal Algorithm
 

---

**Require:**

- $P^I$  the Individuals' Rating Hash Tree;
- $u^C$  the Community Profile;
- $max_r$  and  $min_r$  the maximum and minimum rating values.

**Input Parameters**

- $u_a$ , the profile of the active user  $a$  for which a preference needs to be estimated;
- $i$  item to estimate a preference for.

{Step (0) - Computing neighbours' superset}

$$U_a = \{u_k | \exists r, u_k \in P_{i,r}^I\}$$

{Step (1) - Computing neighbours' similarities }

**for all** distinct rating  $r \in R$  of each item  $j$  rated by  $U_a$  **do**

$$w = |r - r_{a,j}|$$

**for all**  $u_k | (u_k \in P_{j,r}^I \wedge u_k \in U_a)$  **do**

$$w_{a,u_k} = w_{a,u_k} + w$$

**end for**

**end for**

$$w_{a,u_k} = \frac{|I_a \cap I_u|}{|I_a \cup I_u|} - \frac{1}{(max_r - min_r) |I_a \cup I_u|} w_{a,u_k}, \forall u_k \in U_a$$

{Step (2) - Weighted average}

$$r_{a,i} = \bar{r}_a + \frac{\sum_{u \in U_a} w_{a,u} * (r_{i,u} - \bar{r}_u)}{\sum_{u \in U_a} w_{a,u}}$$

return  $r_{a,i}$

---

3.1. In the example, Bob has rated 2 items in common with Alice, but expressing different opinions (Bob has rated them 3 and 5 respectively, whilst Alice have scored them 4 and 3), thus  $\overline{w_{alice,bob}} = |4 - 3| + |3 - 5| = 3$  (step 2). In our example, the rating scale is  $R = 1, 2, 3, 4, 5$ , thus  $\Delta_r = 4$ ; assuming  $|I_{alice} \cup I_{bob}| = 5$  and  $|I_{alice} \cap I_{bob}| = 2$ , then Equation 2.4 would yield  $w_{alice,bob} = \frac{2}{5} - \frac{1}{4*5} * (|4 - 3| + |3 - 5|) = 0.55$ .

### 2.4.2 CReSy

CReSy realisation is lightweight by nature. In fact, to predict a user  $a$ 's preference on a composite item  $c$ , CReSy only considers the feedback provided on its components by  $a$  itself; similarly, when propagating ratings from a composite item  $c$  to each of its components  $i$ , only the compositions consumed by  $a$ , of which  $i$  had been part of, are needed.

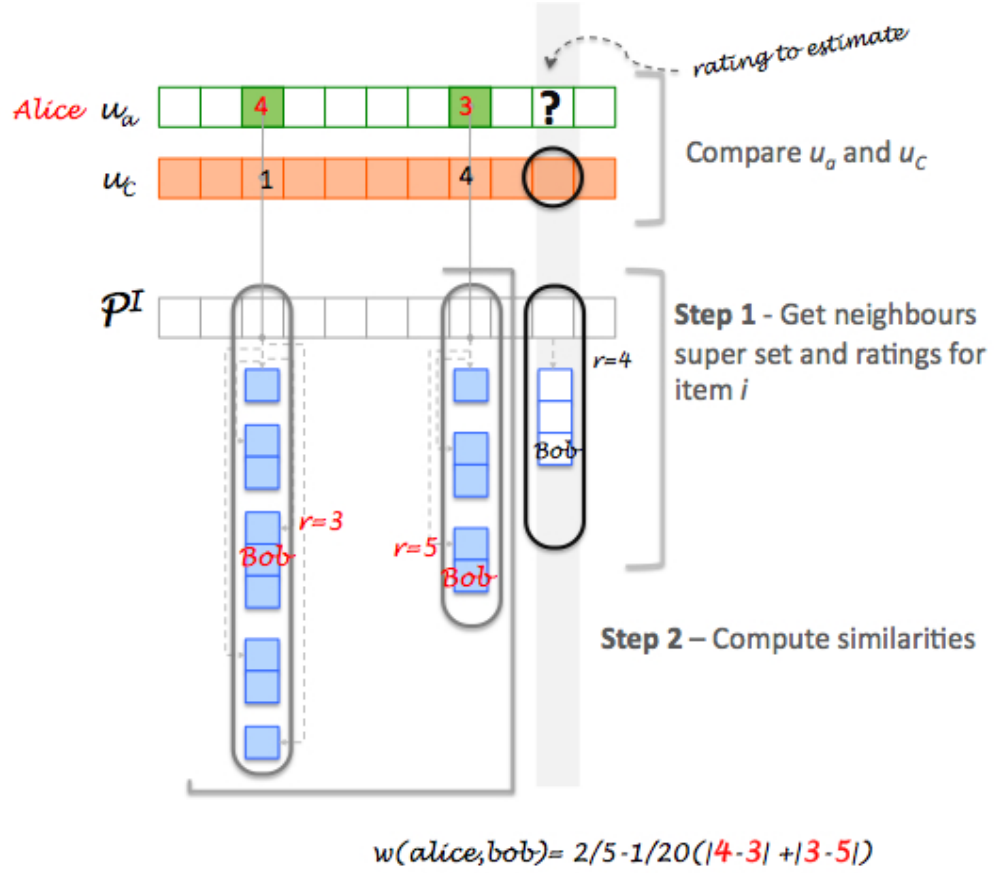


Figure 2.6: Execution of the Recommending Algorithm.

In the following, we consider each case in turn.

### Recommending Composite Items

When a rating has to be predicted for a composite item  $c$ , CReSy first extracts, from the descriptor of  $c$ , the list of its members. We make here the assumption that the anatomy of a composed service is exposed, by means of well defined service composition standards (e.g., WS-BPEL [Coalition, 2003], OWL-S [Martin et al., 2004]). CReSy then traverses the active user  $a$ 's profile to collect its ratings on such items; finally, it estimates  $a$ 's rating for  $c$  as per Equation 2.8. Figure 2.7 depicts the process just described; in this case, item  $c$  mashes-up the items  $i$ ,  $j$  and  $k$ . As the active user  $a$  has never rated item  $k$ , only the feedback on members  $i$  and  $j$  is used to derive a predicted rating on the composite item  $c$ .

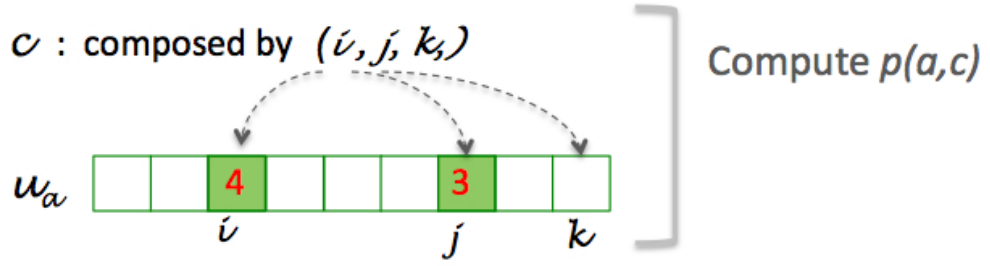


Figure 2.7: Preference Prediction for Composite Items

### Recommending Elementary Items

The computation of a rating for an element  $i$ , based on the feedback given to a composition  $c$ , requires more processing. This is because, unlike the previous case when CReSy estimates a user's preference for a component, not all the information is available at discovery time. In fact, we assume elementary items to not have any information on the composite items they are part of, due to the very spontaneous nature of service compositions and mash-ups.

So, for each item  $i$  that has never been used (and thus rated) per se, CReSy maintains an unordered list of the compositions the element  $i$  has been an element of. We call this element-to-composition mapping a *mapping bag* and the item they related to their *origin item*. All together, mapping bags are collected in a simple mapping structure called *item-to-compositions map* (see Figure 2.8).

In practice, the item-to-compositions map consists of a collection of files – one for each *origin item*. The bag for an item  $i$  is simply a file (named after origin item  $i$ 's identifier) persisting the IDs of the compositions within which  $i$  has been used. Note that mapping bags are saved just until their *origin item* becomes recommendable (i.e., a sufficient number of feedback has been gathered for it). The reason for using files instead of loading the item-to-compositions map in primary memory is quite simple: at the time CReSy needs to compute a user's preference for an origin item  $i$ , CReSy knows the item unique ID and, by consequence, the name of the file collecting its mapping bag, so it can directly access the file in question without need to search for it. Also, mapping bag files are, by nature, quite limited and easy to parse (as they just collect a comma separated list of the IDs of the items the origin item is a component of), hence CReSy can process them in no time. On the other hand, due to the popularity of mash-ups, if we were to store them in memory, we could end up loading a remarkable number of objects. Also, we predict to rarely access such structure. This is because, by nature, the item-to-compositions map persists information on items rarely used alone.

CReSy uses the data structures described as follow: when a user's preference on an *origin*

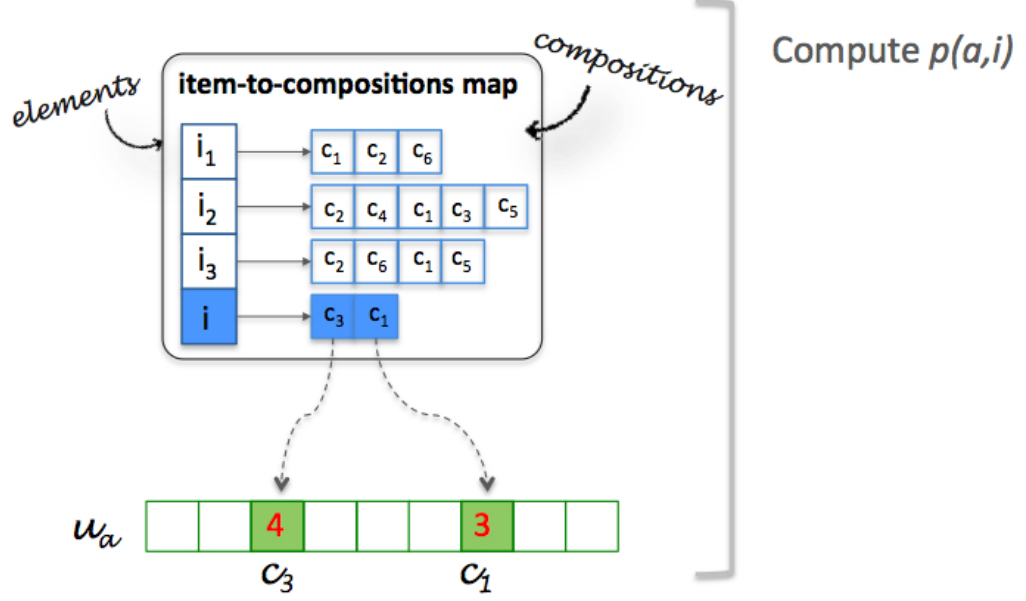


Figure 2.8: Preference Prediction for Elementary Items

item  $i$  is requested, CReSy first accesses the item-to-compositions map to obtain the mapping bag for  $i$  and so the combinations  $c_i$  it is part of; it then traverses the active user's profile (passed as part of a recommendation requests) and collects the related ratings; finally, it averages their values as per Equation 2.9.

Figure 2.8 illustrates this process in practice, when CReSy needs to predict a user's preference for an item  $i$ . In this example, CReSy accesses the  $i$ 's mapping bag which contains  $c_3$  and  $c_1$ ; it then processes the user's profile to collect the related ratings. The active user has rated  $c_3$  to 4 for  $c_1$  to 3. Hence, CReSy averages these values and estimates that the active user will possibly rate the new item  $p(a, i) = 1/2 * (4 + 3) = 3.5$

## 2.5 Related Work

Research on Mobile Recommender Systems specifically developed for *pervasive* decentralised environments is still in its early days, and very few works exist that specifically target the challenges they face. Whilst initially researchers have been looking at ways to improve the *usability* of traditional recommender systems on mobile devices [Miller et al., 2003], in the last years the focus has shifted to the core challenges i.e. *resource scarcity* and *data scarcity*. In the following, we look at the research carried on these two areas and compare our work against it. We look at the research carried out in these two areas jointly: this is because Resource Scarcity and Data Scarcity are intrinsically correlated. In fact, approaches that solve the issue of data scarcity often do so at the expense of greater

resource needs. More data means also more processing and memory allocation. Similarly, solutions that minimise the use of device processing and memory often do so by filtering out data collected, and so they accentuate the data scarcity problem.

The problem of resource allocation is not new to recommender systems, but in mobile recommender acquires new meanings and emphasis.

In traditional recommenders, the problem of resource allocation has mainly concerned scalability issues. To overcome such issues, research initiatives in the area of decentralised recommender systems – mainly in the area of fixed P2P networks – have emerged (e.g., [Coster and Svensson, 2005], [Tveit, 2001], [Miller et al., 2004], [Han et al., 2004], [Wang et al., 2006], [Wang et al., 2009]). These techniques have proved to give accurate results, despite working on a smaller subset of information (about users/items/ratings) than centralised recommenders. However, as discussed in Section 2.1, these techniques cannot be straightforwardly applied to mobile settings: the relative mobility of users and items implies that the set of recommendable services and data is continuously changing, rather than being stable over time; it is thus unlikely to be affordable (neither financially nor computationally) to setup and deploy a purely infrastructure-based recommender service to serve such a dynamic environment.

In mobile recommenders, the problem of resource allocation acquires new meanings. Whilst traditional recommender systems could rely - at least in theory - on almost unlimited processing and memory resources, mobile devices are severely constrained in this regard. Some research has addressed this specific challenge by proposing mobile recommender systems that employ a semi-decentralised approach [Coster and Svensson, 2005, Miller et al., 2004]. These systems are effectively extensions to typical server based systems and they still bank on centralised rating repositories and recommender servers. For instance, [Coster and Svensson, 2005] proposes an incremental collaborative filtering algorithm heavily relying on a central recommender. In there, users are occasionally connected to a central server from which they download and store on their devices just a subset of selected user profiles, together with a ranked list of predictions. When the user is offline, a service on the local device can still recommend items based on the predictions made the last time the user was connected. Each time the user supplies new ratings or exchanges rating profiles with new encountered users, the list of predictions is recomputed locally on the device. This is until the user gets connected to the central server again. These methods suffer from some of the drawbacks of traditional P2P recommender systems. In fact, in decentralised contexts, such as mobile and pervasive environments, it may be not possible to gather all information in centralised repositories.

Consequently some research has been recently looking at recommender systems in decentralised environments [Tveit, 2001, Miller et al., 2004, Gratz et al., 2008, de Spindler et al., 2006]. Some of the methods proposed for these environments seek advisors just among users currently physically connected with the active user. For instance, [Tveit, 2001] builds

upon user-based collaborative filtering and models the selection of users' neighbours as a search problem. To elect similar users, it broadcasts queries consisting of the rating vector of the active user to all terminals in reach. Agents running on users' devices compute the proximity with the cached previous messages: if the proximity is higher than a threshold, then the peer sends back the cached voting vector. If the proximity measure is lower, the query-voting vector is broadcasted further to other peers. This way the agent running on the active user device collects profiles of similar peers that then exploits to predict its owner's preferences. Similarly, the recommender proposed in [Gratz et al., 2008] computes recommendations based solely on the taste of other like-minded users in nearby mobile environments. To do so, whenever an active user requires an advice, the system starts a search for other like-minded users in the mobile ad-hoc network.

Although these methods remarkably minimise resource allocation, they fall back in the accuracy of the prediction made as indeed, in real scenarios, far too few users are co-located and so the active user's neighbours are selected among a very limited set of users.

The problem of data scarcity originates from the same environment recommender systems for pervasive services are targeting. On one side, the lack of device resources is such that mobile recommenders cannot persist all the information (about users and/or ratings and/or items), nor they can afford to process them all; on the other side, the local nature of pervasive services entails that they are of interest to a restricted audience (rather than a global one) and so fewer feedback data can be collected.

The issue of data scarcity has attracted attention recently.

[Gratz et al., 2008] addresses the problem of data scarcity depicting users by the features (weighted keywords) of the content consumed, rather than the preference they expressed on the item experienced. As a result, the similarity between two users is calculated via their content-based profiles instead of their commonly rated items. In this case, the accuracy of the approach strictly depends on the nature of the items being recommended, and cannot be extended to domains where the correlation between interest keywords and tastes is loose.

In [Ruffo and Schifanella, 2009], users elect who they trust as recommenders explicitly, rather than letting an algorithm to impersonally identify users' neighbours; however, trust does not imply taste similarity (e.g., I may trust your opinions as true, despite not sharing them), so the applicability of such concepts is restricted. Similarly, in [Moloney, 2005], the concept of trust is used to recommend who it is safe to interact with in pervasive settings; while safety can be assessed from trust, the same cannot be said for taste similarity.

[de Spindler et al., 2006] adopts a different approach and tackles the problem of data scarcity by computing users' similarity based on the amount of time users spend in the same places at the same time, rather than based on similarity of ratings on the same items. In there, in fact, people exchange their rating profiles each time they share the

same location. This information then feeds a user-based collaborative filtering recommender. Unlike traditional user-based collaborative filtering algorithm, users' similarities are computed via a spatio-temporal proximity measure (i.e., two users are considered similar if they consume the same items simultaneously). The underpinning assumption that the more time is spent in the same places, the more similar the users are, is however dubious, and, as the approach has never been evaluated, its accuracy cannot be assessed. Also, the solution proposed imposes significant constraints to its adoption as it requires (i) a central storage server to persist and publish information on the recommendable items and (ii) all devices to embed a GPS unit.

PocketLens [Miller et al., 2004] – when deployed in the fully decentralised Transitive Architecture – mitigates the data scarcity issue by enlarging the set of possible advisors beyond the ones currently co-located to the active user. In fact, PocketLens builds an item-based recommender on the ratings of users judged as similar neighbours among the ones encountered, or that are neighbours of users encountered. To do so, each time it encounters new users, it discovers and elects new similar neighbours, whose ratings are fed to the underlying item-based prediction algorithm. So ratings of users judged as dissimilar are lost.

Rather than re-defining users' similarity metrics, in [Schifanella et al., 2008] an epidemic protocol to propagate users' profiles and compute (traditional) users' similarity is proposed; while we support the choice of using well-established and thoroughly-assessed users' similarity metrics, their evaluation is rather limited: in fact, unrealistic mobility models are being used where an incredibly high number of interactions with randomly chosen users occur (thus facilitating information dissemination), while in real human networks encounters are much fewer and non-random [Hong et al., 2008]. Also when considering pervasive services, MobHinter [Schifanella et al., 2008] shows another limitation. Pervasive services are often local; this is either because they are embedded within buildings and transports, or because they run on the device of people that move in restricted areas. For a given user, MobHinter processes not only the ratings of users encountered, but also of their neighbours. These neighbours may live in very different geographies than the user in question. Hence, MobHinter may consume computing resources on data not particularly useful. For instance, if we consider a user that transits on regular basis both in London and Milan, then MobHinter would process, for people living in London, also the ratings of a possibly large portion of users that mainly experience services within Milan area.

Both MobHinter and PocketLens also present other drawbacks. Both recommenders get rid of the ratings of users assessed as non-similar or non-affine. This poses a non-trivial problem. In fact, a user profile evolves in time; as such, a user that at any given time appears different to a user  $u$ , may indeed emerge as similar to  $u$  as new ratings for  $u$  are captured. This results in users' neighbours to possibly contain users that are not the most similar ones among the ones encountered (and their neighbours).

All the methods investigated so far lack in taking full advantage of the mobile communities they plug into. They either heavily restrict the set of users they exchange rating information with [Tveit, 2001] [Miller et al., 2004], [Gratz et al., 2008] [de Spindler et al., 2006] or, at another extreme, they process any feedback information available [Schifanella et al., 2008, Miller et al., 2004].

We take a different approach and we argue that *all* and *only* the ratings from users that share the same environments are critical to predict users' preferences on content and services of local nature. On one side, unlike in [de Spindler et al., 2006], we look at all users that have been co-located at any time, not only at users that have experienced similar amenities at the same time. Also, unlike traditional recommenders and [Schifanella et al., 2008] [Miller et al., 2004], we do not consider all users, but just the ones that have or may be exposed to the same local services and content.

Unlike in [de Spindler et al., 2006], we argue that spatio-temporal proximity alone does not provide significant information on users' similarities and we exploit it simply as a mechanism to distinguish users that live and transit through the same locations (and hence will be or have been exposed to the same local services and content). Instead, to compute users' similarities, we build on the natural distinction between trend followers and atypical users in a community. Hence, we bank on these discrete segments to create a lightweight recommender whose accuracy and coverage are comparable with current traditional recommender systems as we shall demonstrate next.

It is also worth noting that, to date, not much attention has been paid to quantifying (and minimising) the cost of running recommender system algorithms on mobile devices, both in terms of computation, memory and battery, despite the intrinsic limitations of these devices. While centralised recommender systems and non-mobile decentralised recommenders take advantage of powerful servers for the offline processing of data, in mobile devices there is no distinction between offline and online environments: everything happens on the same device, and the same resources (CPU, heap memory, battery) are used to pre-process data and predict preferences. It is thus essential to optimise the use of available resources, while still providing recommendations whose precision and coverage are comparable to those of powerful centralised recommender systems in use today. In the next chapter we complement our evaluation of moRe accuracy and coverage with analytical discussion of the overhead it entails.

Last but not least, the problem of recommending items of composite nature has gone largely unexplored until recently; as pervasive services and content will increasingly be combined and consumed in novel mash-ups, it is critical to be able to predict users' preferences on novel aggregations/compositions from partially known preferences about component items. Note that multi-criteria recommender systems (e.g., [Manouselis and Costopoulou, 2007]) have been studied, and so have recommenders that target groups of users rather than individuals (e.g., [Jameson and Smyth, 2007]). However, they solve orthogonal problems to ours: in particular, the former looks at multiple attributes of the

same (composite) item, rather than looking at the item’s ratings only, to improve the utility of a recommendation; in our case though, we are interested in looking at the same attribute (i.e., rating) but of different items at the same time. The latter tries to identify common traits among users so to compute a single recommendation for those within the same group; in our case, though, component services can be very different from each other, and yet we must be able to predict a rating for their composition.

Lately, some research has been looking at recommending composite items, mainly within the music industry [Xie et al., 2010] [Basu Roy et al., 2010] [Khabbaz et al., 2011] [Baccigalupo and Plaza, 2006] [Hayes and Cunningham, 2000] [Tecnologica et al., 2001]. This research, though, builds on content features and it targets domains (like music) whose items meta-data allow to explain users’ preferences. Instead, in our research, we predict pervasive services and content to be of various types; also, we do not expect them to come with meta-data apt to explain users’ preferences. As such, by design, CReSy suits (almost) any kind of recommendable items that have no intrinsic content or meta-data adequate to describe them or understand users’ opinions in their regards, like pervasive services and content.

## 2.6 Summary

In this chapter, we have presented a new method to recommend pervasive services. More precisely, we have discussed moRe, a recommender system specifically designed from the start to be deployed on mobile devices in pervasive computing scenarios. The main observation underpinning the method proposed is that, for mass-like-minded users, there is no need to run expensive algorithms, and item preferences can be simply (yet accurately) predicted using rating averages. For atypical users instead, moRe uses a novel algorithm and companion data structure. Moreover, to enable the prediction of users’ preferences on novel compositions, moRe models compound items as teams, and it uses team performance management reasoning to encompass the qualities of the constituent services, as well as their actual combination, to compute accurate predictions. In the next Chapter we look at the evaluation of the proposed methods and prove that moRe provides accurate recommendations at very low computational and memory cost.

## Chapter 3

# Evaluation of Personalised Discovery

In this Chapter, we present a thorough evaluation of the Personalised Discovery methods proposed before.

First, we introduce the main evaluation metrics used to assess a recommender system; hence, we focus on the actual evaluation of our mobile recommender service: moRe. As moRe consists of two components (i.e., diffeRS and CReSy) layered on top of each other, we present their evaluation in turn: Section 3.2 presents an evaluation of diffeRS as core recommender service for pervasive environments; Section 3.3 then assesses CReSy as a recommender service for composite items. For each of them, we present the results of both an experimental and an analytical evaluation; the former quantifies their performance as recommenders, the latter quantifies the resource overhead they entail.

### 3.1 Background. Evaluating Recommender Systems

The main intent of a recommender system is to assist end users (all end users) to discover items of their taste across a universe of choices that they cannot scrutinise themselves. This implies three major aspects about recommending: the recommender system needs to be accurate when predicting users' appreciations of items; also, it needs to allow users to discover novel items; finally, it needs to do so for as many users and items as possible. Various measures exist to assess the qualities of a recommender. These measures can be broadly distinguished between those assessing the accuracy of a recommender, and those assessing other qualities (e.g. coverage, novelty, serendipity, etc.). In the following, we look briefly to the main metrics used to evaluate recommenders.

### 3.1.1 Accuracy Metrics

Accuracy metrics have been defined first and foremost for two major tasks: (i) assessing the precision of single predictions and (ii) evaluate the ability to discriminate ‘good’ items among all the available ones.

#### Predictive Accuracy

Predictive accuracy metrics measure the deviation of predicted preferences from the actual ratings. Two main metrics are used in the literature for it: the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE). The Mean Absolute Error measures the average absolute deviation between a predicted rating and the user’s true rating. It represents an effective means to measure the statistical accuracy of predictions

$$MAE = \frac{\sum_{(i,j) \in Predictions} |r_{i,j} - p(i,j)|}{|Predictions|}$$

The Root Mean Square Error sizes the square deviations between predictions and actual ratings. Hence, it pronounces the impact of large errors over small ones.

$$RMSE = \sqrt{\frac{\sum_{(i,j) \in Predictions} (r_{i,j} - p(i,j))^2}{|Predictions|}}$$

#### Recall and Relevancy

Precision and Recall come from Information Retrieval (IR). They do not assess single predictions; instead, they evaluate the ability of a recommendation system to return all relevant items to active users. Given an active user, two sets of items can be identified: (i) items that the recommender system judges as relevant to the users – in the following we refer to this set as set  $A$  – and (ii) items effectively relevant to the user – in the following we refer to this as set  $B$ . Ideally, a recommender system should be able to judge as relevant to a user **all** and **only** the items that are effectively relevant to her. In order to assess a recommender system, the following should be known:

- Number of items judged as relevant that are effectively relevant (positive hits)
- Number of items judged as relevant that are not indeed relevant (negative hits)
- Number of items judged as not relevant but that are indeed relevant (negative misses)
- Number of items judged as not relevant that are indeed not relevant (positive misses)

An ideal recommender would have both negative misses and negative hits equal to zero. All these entities are affectively related to each other, so it is enough to compute just two of the following entities: precision, recall and fallout.

Precision is the rate of effectively relevant items among the ones judged as relevant

$$precision = |A \cap B|/|A|.$$

Recall is the rate of retrieved items judged as relevant among the ones effectively relevant

$$recall = |A \cap B|/|B|.$$

Fallout is the rate of items judged as relevant among the ones indeed not relevant

$$fallout = |A \cap \overline{B}|/|\overline{B}|.$$

If any two of these three metrics are known, then the third one can be derived. In fact, if we use a parameter called generality ( $G$ ), which is a measure of the density of relevant items in a catalogue of size  $N$  (i.e.,  $G = |B|/N$ ) then the following holds true:

$$precision = \frac{G * recall}{(G * recall) + fallout(1 - G)}$$

Rather than measuring two separate entities, often any two of the three metrics here introduced are combined in one single metric such as the  $F1$ .  $F1$  gives equal weight to precision and recall:

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

### Signal vs. Noise

When it comes to recommendations, it is not so important that all relevant results are returned, but rather that the system is more often correct than wrong when providing recommendations. On this aspect, metrics in use in electronics can be used to assess a recommender system. These metrics aim to assess the ability of a filtering system to distinguish between signal and noise. In recommendation systems, the signal corresponds to the recall and the noise corresponds to the fallout. These two metrics are often plotted to create the Relative Operating Characteristics (ROC). The ROC helps to understand how effective a filtering system is in distinguishing the signal against the noise as the noise increases. Indirectly, this allows to appreciate how the signal to noise ratio varies as the number of recommendations returned in a recommendation list grows.

### 3.1.2 Beyond Accuracy

Recommenders are first and foremost tools to assist users in a universe of endless choices; as such they are about much more than just accuracy. Still, non-accuracy metrics have largely been denied major research interest so far and have only been treated as marginally important supplements for accuracy metrics, as they are difficult to measure. In the following we shortly introduce qualities and associated metrics – where existing – that are beyond accuracy.

#### Coverage

Even the most precise recommender has very little value if, in fact, it can estimate preferences just for a small subset of items and/or users. In order to assess the ability of a recommender to provide predictions for as many items as possible, the coverage metric is used. The coverage of a recommender system is a measure of the portion of items in the system over which the system can form predictions or make recommendations. Two types of coverage exist: prediction coverage and catalogue coverage. The first one assesses the percentage of items a recommender can form predictions for, the second evaluates the percentage of items that the recommender ever recommends to users. Prediction coverage is measured by selecting a random sample of user-item pairs, for which predictions are requested and by then measuring the percentage for which a prediction was provided. Catalogue coverage is usually measured on a set of recommendations provided within a delimited interval of time. For instance, it might be measured by computing, for each recommendable item, the number of times it has been recommended per unique user. Both prediction and catalogue coverage need to be measured in combination with accuracy. This is because higher coverage can be often achieved compromising on the accuracy and vice versa.

#### Learning Rate

The learning rate assesses the ability of learning from the data available and hence reaching acceptable quality. Intuitively, as the quantity of learning data increases, the quality of the predictions should increase. Three different learning rates have been considered in recommender systems: overall learning rate, per-item learning rate, and per-user learning rate. The overall, per-item and per-user learning rates plot the increase of the quality of the recommendation with the increase of the number of ratings, the number of ratings per item and the number of preferences expressed per user respectively. The issue of evaluating the learning rates in recommender systems has not been significantly covered in the literature; we predict, though, that as more mobile recommender systems will be investigated, learning rates will become a much more significant evaluation factor.

### Novelty and Serendipity

Recommender systems that provide recommendations that are highly accurate but also obvious would be useless in practice. In fact, obvious recommendations can be easily guessed without the need of intelligent systems. Hence, recommender systems should also be assessed for the ‘non-obviousness’ of the recommendations made. This entails assessing their novelty as well their serendipity. Novel recommendations refer to recommendations made on items the user was unaware of; whilst serendipitous recommendations refer to recommendations of items users are unaware of and would probably always be. The distinction between novelty and serendipity is important when evaluating collaborative filtering recommender algorithms. This is because the potential for serendipitous recommendations is one facet of collaborative filtering that traditional content-based information filtering systems do not have. We are not aware of any serendipity measure at this time. Note that serendipity should be assessed in combination with accuracy. However, current methodologies for assessing accuracy require items to be already known and rated. As such, accuracy intrinsically rewards algorithms that make the most obvious recommendations, while a good serendipity metric would look at the way the recommendations are broadening the user’s interests over time.

## 3.2 diffeRS Evaluation

### 3.2.1 Performance

In this section, we discuss the performance evaluation of diffeRS; we introduce the methodology applied, the datasets used and the metrics adopted before presenting the actual experiments setup and their results.

#### Methodology

When analysing diffeRS we are interested in two aspects: (i) the qualities of the recommending algorithm itself, and (ii) its suitability for mobile environments. To assess both of them, we first study diffeRS recommending algorithm and evaluate diffeRS as a traditional recommender system; hence, we focus on diffeRS as a mobile recommender and we analyse its performance as more and more user profiles are exchanged overtime.

In both cases we follow the de-facto standard methodology used in recommender systems research: we take a rating dataset and split it into a *training set* and a *test set*. The former represents knowledge that the recommender system can leverage; the latter provides both the test cases (i.e., the user  $u$ -item  $i$  pairs for which diffeRS will try to compute a prediction

$p(u, i)$ ) as well as the actual preferences  $r_{u,i}$  registered by the user. We feed the training set to the recommenders in study; hence, we run the use cases in the test set and compare estimated preferences against real preferences; subsequently, we analyse the results and we derive key metrics. We run such evaluation for multiple splits of the rating data set.

In principle, we use this same methodology both for assessing diffeRS as a centralised recommender system and a mobile one, but with few differences. When evaluating diffeRS against traditional (centralised) recommender systems, we assume the whole training set to be known and accessible to all users; as a consequence, predictions for all user-item pairs in the test set are computed at once, based on such training set. Things are different when assessing diffeRS as a mobile recommender. First, we have to simulate users' mobility and their profile exchanges; secondly, we need to explore the qualities of the recommenders as more and more rating profiles are exchanged.

In fact, in pervasive deployments, each user has just a limited knowledge of the preferences expressed by other people, based on the profiles exchanged thus far upon encounter. Such knowledge (i.e., the training set) grows over time, as new encounters are made and new profiles are exchanged. So, when assessing diffeRS as a mobile recommender, our evaluation aims to study the impact that mobility (and thus the partial knowledge that has been acquired so far) has on the qualities of the recommendations provided. To do so, whenever a new encounter occurs, predictions for the items in these users' test sets are re-computed, and key metrics recorded.

In practice, we use real mobility datasets collecting the traces of users' encounters and we randomly map users in the mobility dataset to users of a rating dataset (still preserving the rating distribution of the rating dataset, as later explained when describing the actual Datasets). Hence, we built on each device a training set that consists only of the users' profiles the same device has collected so far. Each time two devices exchange for the first time their rating profiles, we update their training sets, and we make new predictions for the items in the test set, thus re-calculating precision and coverage as knowledge grows.

## Metrics

We have evaluated diffeRS performance in terms of both *accuracy* and *coverage*. We assess accuracy by measuring the Mean Absolute Error (MAE), that is, the average absolute deviation between a predicted rating  $p(u, i)$  and the user's true rating  $r_{u,i}$ , over a set  $R_T$  of user-item pairs:

$$MAE = \frac{\sum_{(i,j) \in R_T} |r_{i,j} - p(i, j)|}{|R_T|}$$

We use such metric as it is the de-facto standard metric within academic research on Recommender Systems.

Being  $T$  the set of all possible user-item pairs in the test set and  $R_T \subset T$  the set of pairs for which a preference can be predicted, we compute coverage as:

$$Coverage = \frac{|R_T|}{|T|}$$

We use these two metrics rather differently depending on whether we are assessing diffeRS recommending algorithms or its qualities as a mobile recommender system. In the former case, we are interested in assessing its accuracy with respect to traditional recommender systems, whilst we do not look at the prediction coverage. This is because diffeRS can only improve it (by returning item averages in the case users' neighbours are not found). Consequently we evaluate its precision just for the cases where both diffeRS and traditional recommenders return a prediction. In the latter case, we are also interested in measuring diffeRS coverage in addition to its accuracy. This is because we predict prediction coverage to become a prominent problem in mobile recommenders as data scarcity becomes a crucial issue. So, when assessing diffeRS as a mobile recommender we analyse the impact that mobility has on both accuracy and coverage. To do so, whenever a new encounter occurs, we compute the prediction MAE and coverage and we compare them against a centralised deployment. We will thus study how MAE decreases, while coverage increases, as more knowledge becomes locally available.

## Datasets

Whilst assessing diffeRS against traditional centralised recommenders simply requires access to users' rating information, things get more complex as we analyse diffeRS in mobile environments. In fact, in order to realistically evaluate diffeRS as a mobile recommender system in pervasive environments, we need actual data about both users' ratings *and* about users' encounters. The former serves to model rating behaviours and to verify predictions (as discussed above); the latter serves to model propagation of rating information (training set), based on which predictions are computed. As no single dataset exists containing both types of information, we relied on two types of datasets, namely, rating datasets and movement datasets and we have mapped their users. More precisely, a first set of experiments was conducted mapping users in the MovieLens Light<sup>1</sup> rating dataset to users in the MIT Reality Mining<sup>2</sup> movement dataset; a second set of experiments was then conducted mapping the Yelp<sup>3</sup> rating dataset to the Cabspotting<sup>4</sup> movement one.

The main characteristics of each of these datasets are briefly summarised below.

- *MovieLens Light (rating dataset)* - MovieLens is a web-based research recommender

---

<sup>1</sup><http://www.grouplens.org/node/73>

<sup>2</sup><http://reality.media.mit.edu/>

<sup>3</sup><http://www.yelp.com/>

<sup>4</sup><http://Cabspotting.org>

system that debuted in late 1997. The MovieLens dataset was collected by the GroupLens Research Project at the University of Minnesota during the seven-month period from September 19th, 1997 through April 22nd, 1998 and has been widely used by the recommender systems research community to validate results. We have employed a subset of the whole MovieLens rating data known as MovieLens Light. This subset contains 943 randomly selected users, providing 100,000 ratings (in a scale 1 to 5) of 1682 movies. We have chosen this dataset as it has become over the years a de-facto standard dataset to validate results.

- *Yelp (rating dataset)* - The Yelp dataset collects users' ratings for restaurants in New York. More precisely, it stores the rating profiles of 1000 users that gravitate the most in the area of Tribeca, Green Village and Soho and have collectively provided 59,126 ratings (in a scale 1 to 5) on 4,707 restaurants. The data refers to a period that elapses from 29 October 2004 to 27 October 2009 (i.e., the day we have crawled Yelp.com and built this dataset). We have chosen this dataset as restaurants fit with our vision about items of local nature.
- *MIT Reality Mining (movement dataset)* - The MIT Reality Mining dataset – available from CRAWDAD [Kotz and Henderson, 2005] repository – contains the Bluetooth IDs of devices in proximity, as detected by one hundred Nokia 6600 phones, which were given to MIT staff and students, over a period of nine months over the course of the 2004-2005 academic year. Each phone had been configured to scan the environment and log co-located devices (i.e., within one hop distance). The final dataset contains in excess of 500,000 hours of data about human activity. We chose this dataset as it has been collected in what we consider a typical setting for the consumption of pervasive services (i.e., a university campus, with some services being available centrally, others from devices embedded in buildings, and others still from peer Bluetooth devices); moreover, the dataset spans a long enough period of time for users' profiles to be exchanged and for our model to learn and exploit them.
- *Cabspotting (movement dataset)* - The Cabspotting dataset is a dataset published by the Exploratorium – the museum of science, art and human perception through the Cabspotting project<sup>5</sup>. It contains GPS coordinates of about 500 taxis collected approximately every 10 seconds for a period of one month (May 2008) in the San Francisco Bay Area. We chose this dataset as it provides a mobility model very different in nature from the MIT Reality Mining dataset, while still being very pertinent to metropolitan environments. Moreover, it allowed us to analyse differRS performance when dealing with a much bigger community than the one covered by the MIT Reality Mining project.

The mapping has then been conducted as follows: 100 users from the MovieLens Light rating dataset have been chosen at random, making sure that the resulting ratings-per-

---

<sup>5</sup><http://cabspotting.org>

user distribution followed the same ratings-per-user distribution of the whole MovieLens Light; these users have then been mapped to the 100 users in the MIT Reality Mining dataset, and this mapping has been repeated 100 times at random.

More precisely, we have ordered the 943 MovieLens Light users by ascending number of rated items. Hence, we have divided the ordered list in 100 groups of about 94 users (i.e. 943 divided by 100), with the first group including the biggest user profiles in terms of items rated, and the last group consisting of the smallest user profiles. Then for each group we have selected randomly a user and we have mapped it to a user in the MIT Reality Mining dataset.

The actual mapping of MovieLens to MIT Reality Mining users can have a major impact on the overall experiments. For instance, if we map MovieLens users that have rated the greatest number of items with the MIT Reality Mining users involved in the greatest number of encounters we will have a greater number of rating profiles exchanged and hence greater virtual local rating matrices in each user's device thus leading to better prediction coverage and precision. To limit the impact of "fortunate" or "unfortunate" mappings we have repeated all experiments for 100 random mappings.

Also we remark that, by mapping the MovieLens Light and MIT Reality Mining datasets, we have created a synthetic dataset of users that have rated movies in the MovieLens dataset and work or study at the MIT. This poses non-trivial constraints on our validation experiments as effectively we are making the assumption that people social interactions (encounters) and propensity to experience new products or services (movies) and share their experience with the community (through ratings) are independent, which is probably not the case. Ideally we would have liked to use a real dataset that combines users' movements and ratings; this has not been possible as such dataset is not presently available to the research community. In the future, it will be indeed interesting to re-assess differRS on datasets that combine the two aspects (as may become possible from data being released by websites like FourSquare).

We have used the same approach for the Yelp-Capspotting mapping. In fact, 500 users have been randomly selected from the Yelp dataset in a ratings-per-user distribution-preserving manner; these users have then been mapped to the 500 users in the Capspotting set, and the mapping has been repeated 100 times at random.

Before the beginning of the simulation, the set of ratings that each user has expressed is split in two, so that 90% becomes the user's profile, and the remaining 10% represents the user's test set (this split too has been repeated 100 times at random). At the start of the simulation, a user's training set contains only its own profile (i.e., they have no knowledge about ratings from other devices). When user  $a$  encounters  $b$ , they exchange their own rating profiles; each of them then uses differRS to compute predictions over their test set, based on the profiles collected thus far.

## Benchmarks

In order to evaluate diffeRS, we have adopted as reference benchmark the  $k$ -Nearest Neighbour [Herlocker et al., 1999] user-based collaborative filtering method. Rather than using the traditional Person coefficient alone, we have weighted it by the Jaccard index, as their combination provides better accuracy than the traditional Pearson coefficient alone [Candillier et al., 2008b]. In the following we refer to this method as *jpCF*.

Whenever examining diffeRS as a mobile recommender, we still compare the results against *jpCF*, but this time with  $k$  set to be all users encountered thus far, as otherwise both accuracy and coverage were severely compromised due to data sparsity. In addition, we have also considered a small variation of this benchmark, whereby an item’s average rating is returned, whenever a prediction about such item has to be made for a user  $u$  whose neighbourhood is empty (i.e., no similar users could be found); we do so as each device has very little knowledge about the local rating community in the initial phases of the simulation, and this has remarkably negative effects on the coverage achieved by traditional techniques. We refer to this last method as *jpCF@item avg.* In all experiments, we have used diffeRS with  $\alpha = 1$  to distinguish mass-like minded users from atypical ones.

## Setup

We have run two sets of experiments, the first one exploring diffeRS as a recommender and the second one extending the study to appraise diffeRS as a mobile recommender.

**diffeRS as Recommender.** To analyse diffeRS qualities against traditional recommenders we have conducted three distinct classes of experiments. The first one aimed to assess diffeRS prediction algorithm for all users, regardless of their classification, whilst the second and the third classes focused on the evaluation of serving mass-like minded users or individuals, respectively.

For the first class of experiments we have considered all the ratings in the rating dataset analysed, and have split randomly them in two, so to obtain a training set consisting of 90% of the ratings and a test set containing the remaining 10%. Hence, we have fed the training set to both diffeRS and *jpCF* and we computed predictions for the test set. We have repeated this experiments for 100 random splits and, for each split, for different values of  $k$  nearest neighbours (for *jpCF*). The experiments to assess *diffeRS* on trend-followers and atypical users were very similar and simply differed in that the ratings from just mass-like minded users or just individuals were considered in the test.

**diffeRS as Mobile Recommender.** To assess both accuracy and coverage of diffeRS when deployed in a totally distributed mobile setting, we have randomly mapped a rating

dataset (i.e., MovieLens Light or Yelp) to the users in a movement dataset (i.e., MIT Reality Mining or Cabspotting) after splitting the rating dataset in 90% training and 10% test, we have assigned to each user its own profile as per training set (i.e., users' profiles do not include ratings in the test set); we have analysed the prediction computed over the test case by each user, at each encounter, based on the profiles collected thus far, and we have compared them to predictions made by differRS when deployed as a traditional (centralised) recommender. Note that, at the beginning of the simulation, a user's training set contains only its own profile (i.e., they have no knowledge about ratings from other devices). Then, as a user  $a$  encounters a user  $b$ , they exchange their own rating profiles and  $b$ 's profile becomes part of  $a$ 's training set and vice versa.

We have repeated the mapping of users, between movement and rating datasets, 100 times at random. This is to mitigate the effect a peculiar mapping may have on the final results. For example, a mapping that matches the richest rating profiles to the most dynamic users would correspond to more dense users' training sets and ultimately better predictions; similarly a mapping that relates less prolific reviewers to more dynamic users would result in sparser training sets and hence poorer recommendations.

## Results

The first set of experiments we have conducted aimed to assess the accuracy of differRS prediction algorithm in a centralised setting. In the following, we report the results for the cases where we assess differRS prediction accuracy across all users (Figure 3.1 and Figure 3.3), and for just mass-like minded users and individuals separately (Figure 3.2 and Figure 3.4).

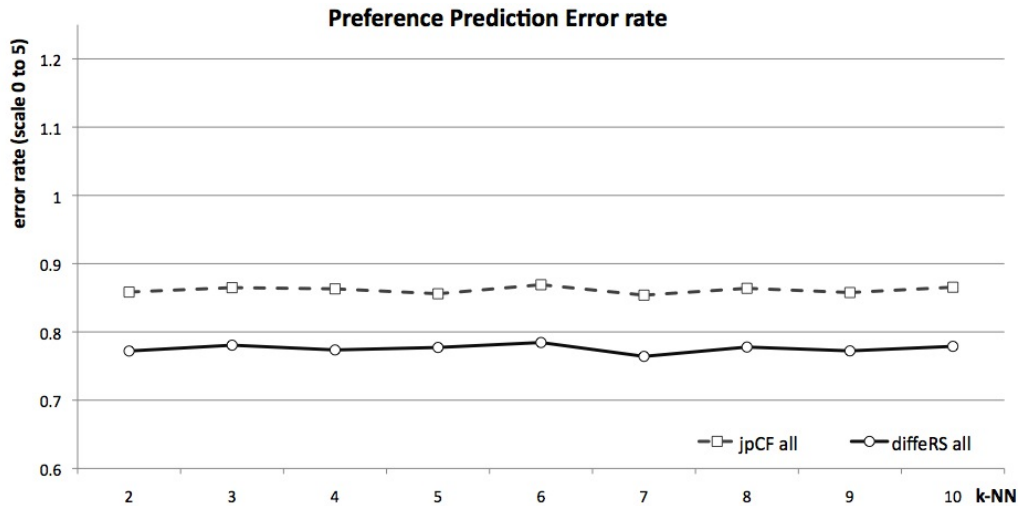


Figure 3.1: Accuracy for All Users (MovieLens)

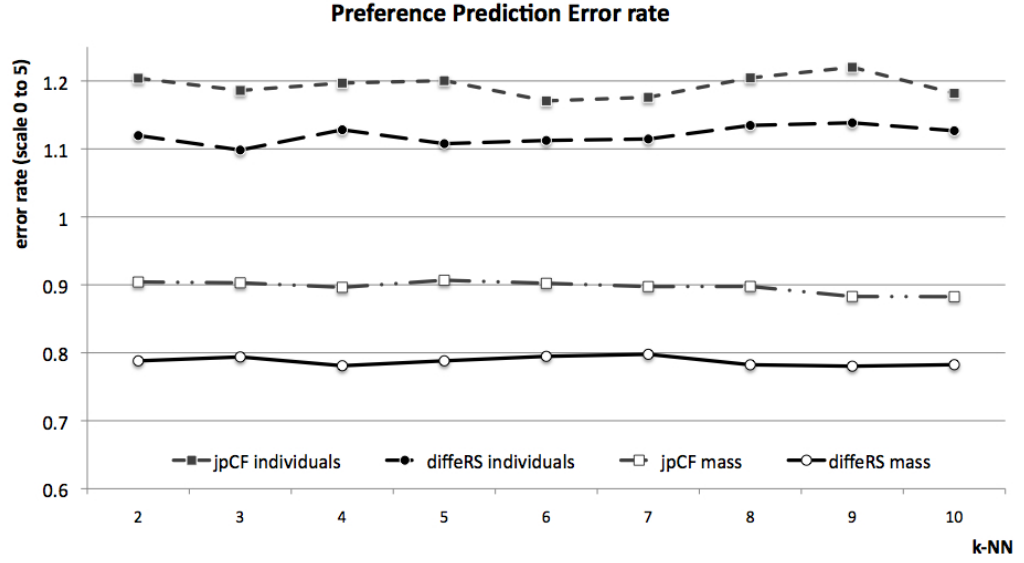


Figure 3.2: Accuracy by Cluster (MovieLens)

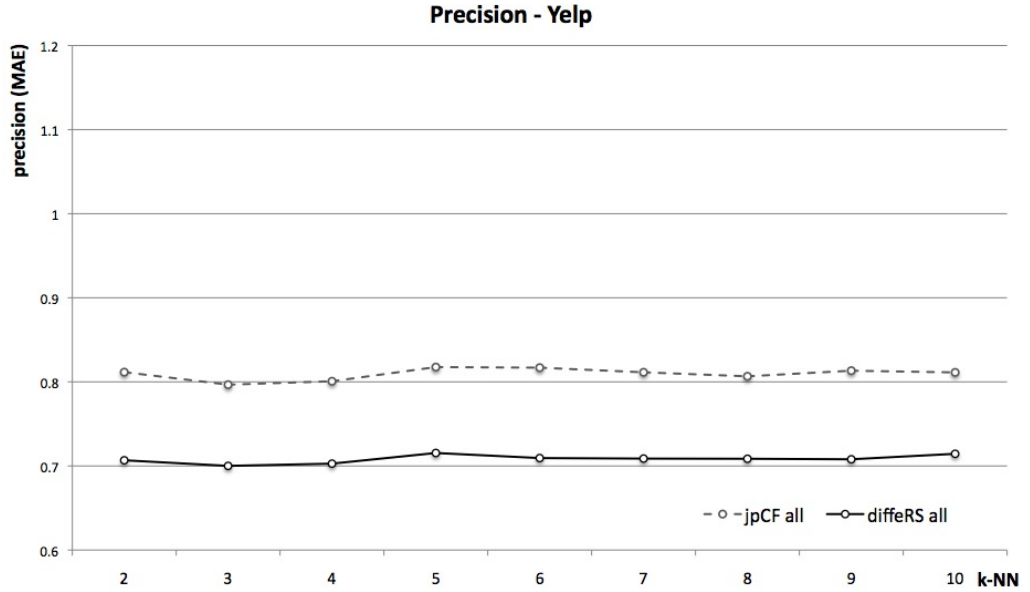


Figure 3.3: Accuracy for All Users (Yelp)

As shown, *diffeRS* performs better than *jpCF* for any value of  $k$  when considering all users (Figure 3.1 and Figure 3.3). The gap in accuracy is not trivial. While the MAE for *jpCF* is in the range  $[0.85, 0.87]$  (MovieLens) and  $[0.80, 0.82]$  (Yelp) for any  $k$ , that of *diffeRS* is in  $[0.76, 0.79]$  (MovieLens) and  $[0.70, 0.71]$  (Yelp). We also observe (Figure 3.2 and Figure 3.4) that *diffeRS* improvement in accuracy is greater for mass-like minded users, confirming our hypothesis that, for this category of users, there is no need to run computationally expensive algorithms, and simple rating averages will do just fine. As

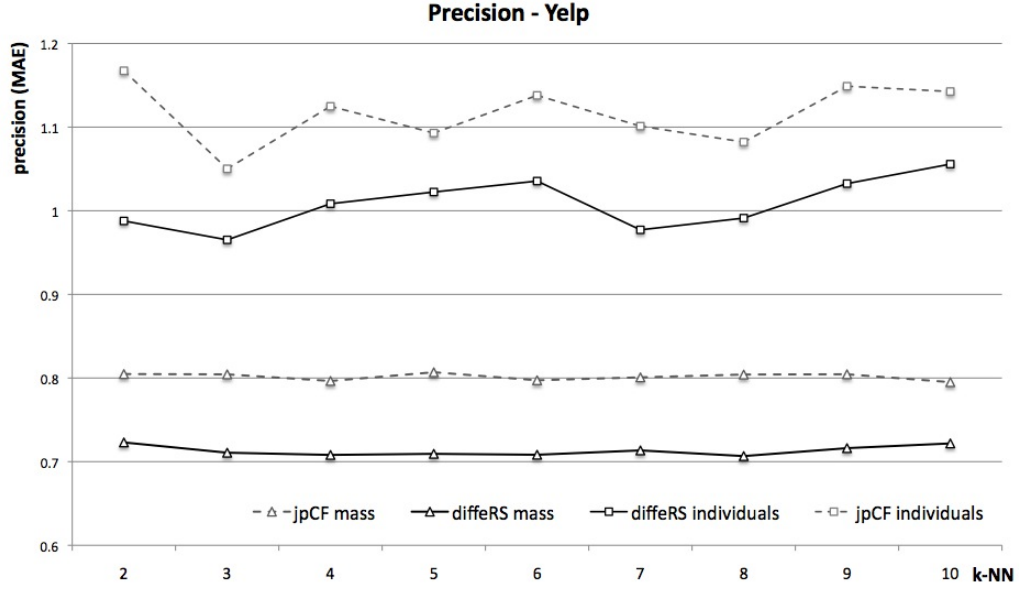


Figure 3.4: Accuracy by Cluster (Yelp )

expected, atypical users are the most difficult to predict; nonetheless, the MAE reported by *diffeRS* is consistently lower than that reported by *jpCF* even for this category of users. Note the limited dependence on the value of  $k$  both for *diffeRS* and *jpCF*: this may be due to the use of Jaccard correlation as a measure of the reliability of similarity coefficients (as well as an additional measure of similarity). In fact, the Jaccard correlation allows achieving more reliable similarity measures and users' neighbourhoods. Hence, the benefit of enlarging the size of users' neighbourhoods may be then compensated by the polluting effect of pondering the preferences of users that are less similar to the active one.

The second set of experiments aimed to assess both accuracy and coverage of *diffeRS* when deployed in a totally distributed mobile setting.

We first discuss results for the mapping between MovieLens and MIT Reality Mining. Figure 3.5 and 3.6 illustrate accuracy (MAE) and coverage respectively for the most dynamic three months (i.e., the months for which a lot more users' encounters occur, so knowledge spreads more quickly), and in Figure 3.7 and 3.8 for the three most representative of the whole MIT Reality mining dataset.

On the  $x$  axis, rather than reporting the amount of time passed, we report how many profiles have been exchanged; we believe this gives a better indication of how much knowledge has been spread in the system (as time may pass by, with no new encounters happening). As shown, the greater the number of users' profiles exchanged, the better both the coverage (higher) and the accuracy (lower MAE), irrespective of the algorithm used. If we take a closer look at accuracy (Figure 3.5 and Figure 3.7), we notice that *diffeRS* constantly provides better predictions, with a weighted average MAE of 0.841 and 0.868 respectively,

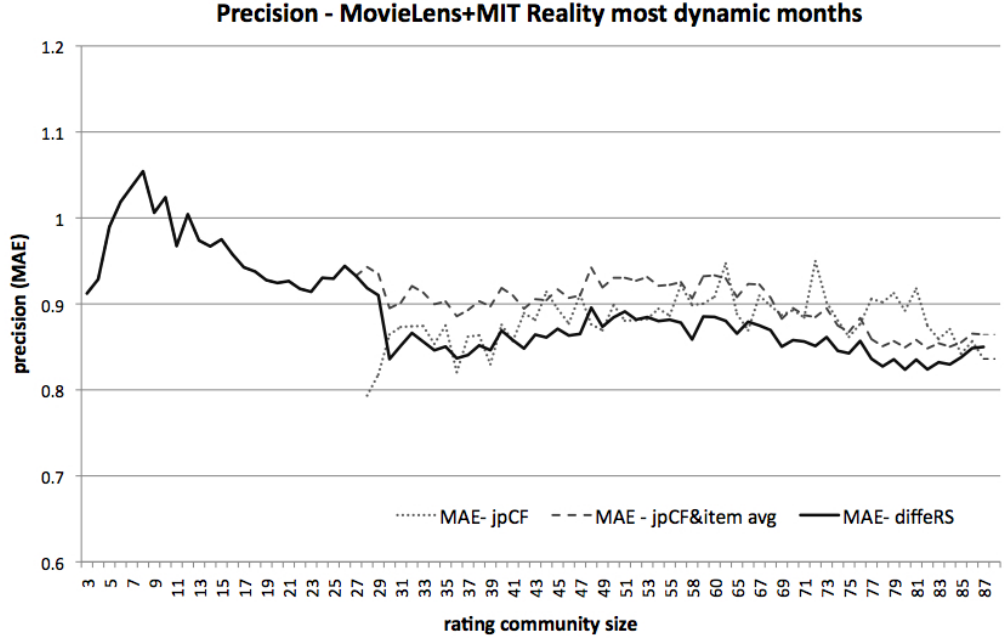


Figure 3.5: Most Dynamic Months - Accuracy (MovieLens+MIT Reality)

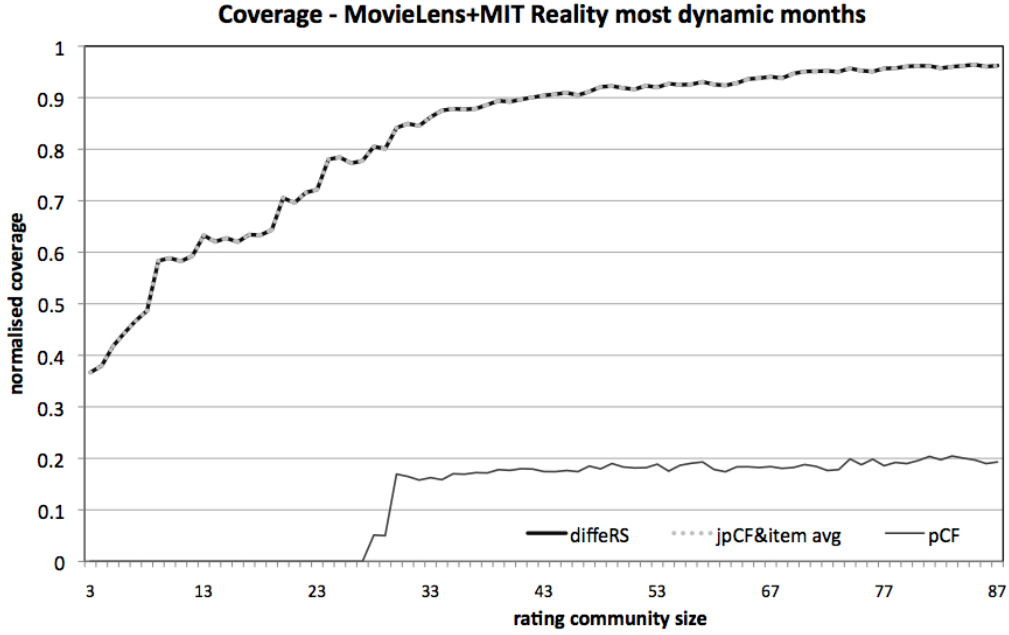


Figure 3.6: Most Dynamic Months - Coverage (MovieLens+MIT Reality)

thus improving over both the 0.863 and 0.905 of *jpCF&item avg* and the 0.874 and 0.889 of *jpCF*. In order to assess how far off we are from the performance that a centralised deployment would entail, we also computed the MAE that *jpCF* achieves as a centralised

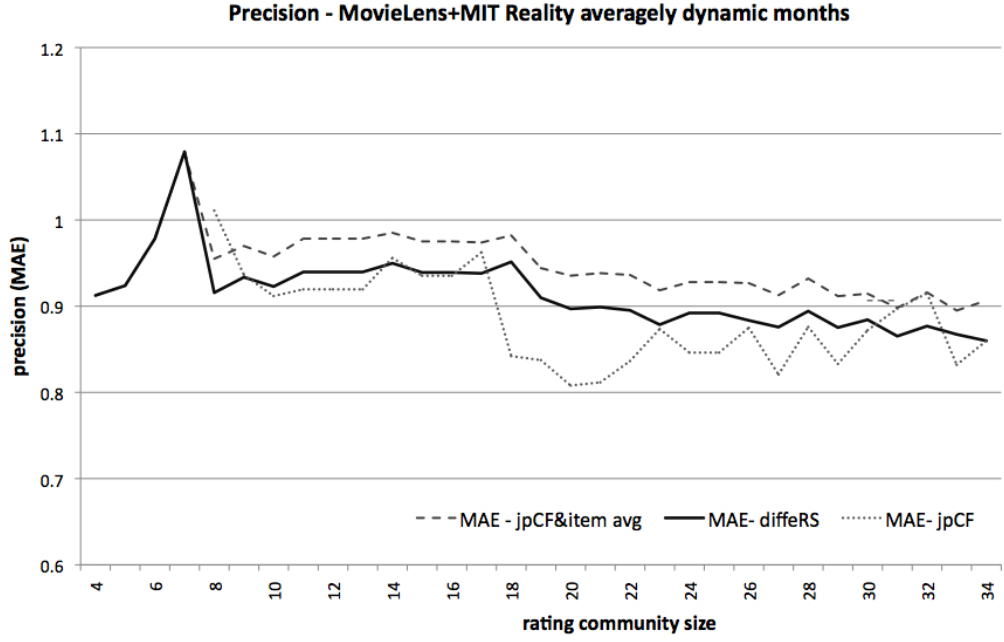


Figure 3.7: Representative Months - Accuracy (MovieLens+MIT Reality)

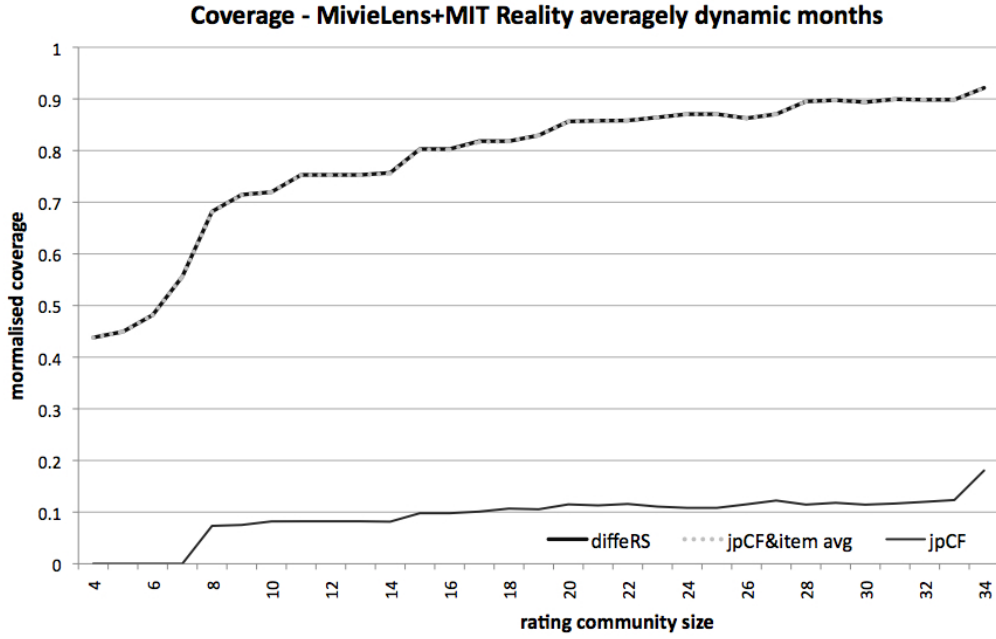


Figure 3.8: Representative Months - Coverage (MovieLens+MIT Reality)

recommender system, that is, with no mobility involved, and with a split of the MovieLens dataset as 90% training and 10% test; the recorded MAE was 0.85 (with  $k = 10$ ). This means that the accuracy achieved by *differRS* as a *totally decentralised* mobile recommender

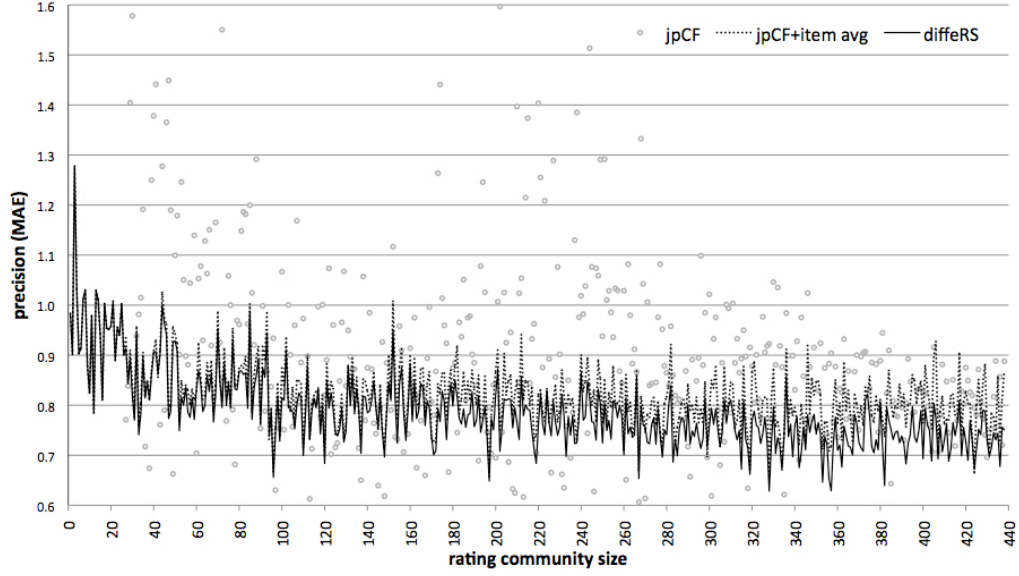


Figure 3.9: Accuracy (Yelp + Cabspotting)

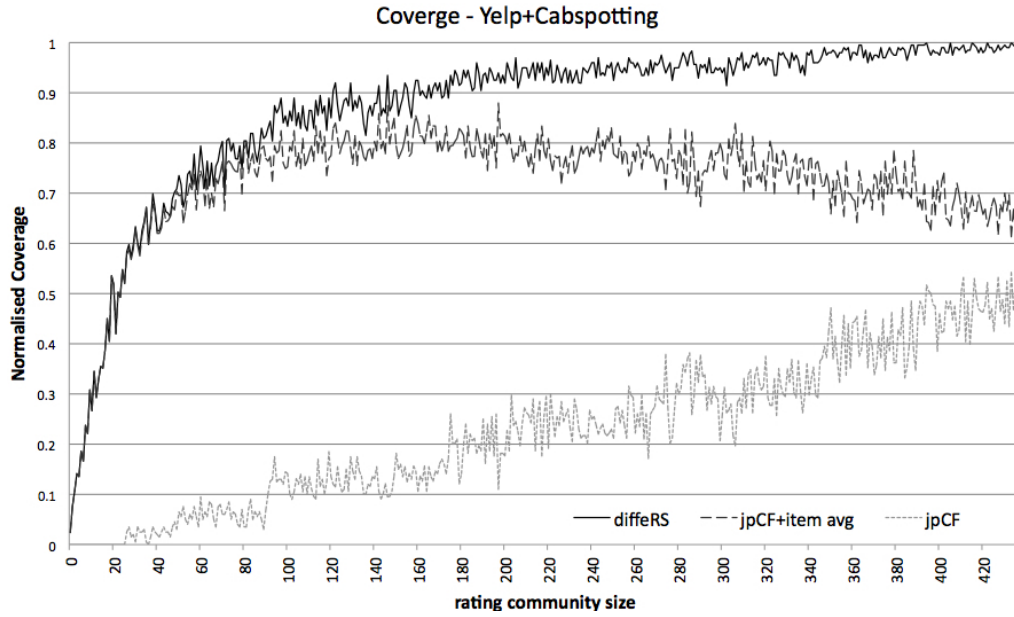


Figure 3.10: Coverage (Yelp + Cabspotting)

service (0.841) is comparable to (and indeed even slightly better than) that observed using a traditional centralised deployment. If we now turn our attention to coverage (Figure 3.6 and Figure 3.8), we can see that *diffeRS* is capable of exploiting the little information it collects to quickly achieve a very high coverage (above 90%), while a traditional *k*NN approach (*jpCF*) never reaches above 20% in this setting. These results combined confirm *diffeRS* suitability as a mobile recommender service.

These results were confirmed also when analysing the same mobile experiments for the Yelp/Cabspotting mapping. Results for accuracy and coverage are reported in Figure 3.9 and Figure 3.10 respectively. As before, the greater the number of users' profiles exchanged, the better both the coverage (higher) and the accuracy (lower MAE), irrespective of the algorithm used. When looking at accuracy (Figure 3.9), *differRS* constantly provides the best estimates, with a weighted average MAE of 0.787, improving over the 0.983 of *jpCF@item avg* and the 0.893 of *jpCF*. As before, we also computed the MAE that *jpCF* achieves as a centralised recommender system, with no mobility involved, and with a split of the Yelp dataset as 90% training and 10% test; the recorded MAE was 0.8 (with  $k = 10$ ). Once again, the accuracy achieved by *differRS* as a *totally decentralised* mobile recommender service (0.787) is better than that observed using a centralised deployment. In terms of coverage (Figure 3.10), it is interesting to note that, while *differRS* and *jpCF@item avg* reach peaks above 90% (on average during the whole period), *jpCF* coverage is always less than 50%, and is well below 20% for users who have exchanged profiles with less than 100 (out of 500) other users. This is because it is the only algorithm not exploiting item averages when users' neighbourhoods cannot be computed. It is worth noting that even *jpCF@item avg* coverage remains lower than that achieved with *differRS*: this is because, once more than about 30% of profiles have been exchanged, users' neighborhoods are no longer empty and  $k$ NN is computed (instead of returning items' averages); however, such neighbours may not have rated the items of interest, thus failing to compute a prediction for them. Once again, these results combined confirm *differRS* suitability as a mobile recommender service. The only question yet to answer is what overhead differRS entails on a mobile device; we answer this question next.

### 3.2.2 Overhead

In order to assess the overhead that differRS entails, we have analytically quantified its memory allocation and processing cost.

#### Primary Memory Allocation.

To assess differRS overhead in terms of memory allocation, we have compared it against the case where a full user-item rating matrix is used, as well as current (industry) best practices. As a reference for current best practices, we consider Taste [Sean, 2005], an Open Source recommender system now part of the Apache Mahout framework [mah, 2010]. We consider Taste rather than other recommendation frameworks as a benchmark as (i) it is open source, so we can have access to the source code, (ii) it is part of a framework of an established authority and (iii) it has been developed by an experienced software engineer – a former Sr. Software Engineer at Google. In the following, we define as  $U$  the set of users,  $I$  the set of recommendable items, and  $P$  the whole set of preferences expressed,

with  $n = |U|$ ,  $m = |I|$  and  $l = |P|$ , and we assume that users can express a preference just once for each item (i.e.,  $|P| \leq |U||I|$ ).

We observe that:

- Recommender systems that use a *rating matrix structure* to persist rating information, often use it in conjunction with a structure that stores, for each user, its top  $k$  neighbours and their similarities (proactively computed during periodic batch processes). In this case, the amount of memory allocated would thus be:  $|U| * |I|$  to store ratings, and  $|U| * k$  to store neighbourhoods (each consisting of a list of  $kNN$  user IDs and their similarity). The memory allocation for recommender systems using the rating matrix structure is thus  $O(n * m)$ .
- In deployed recommender systems, the rating matrix is often very sparse (i.e., each user expresses ratings for a very small subset of items). That is,  $|P| \ll |U| * |I|$ . Taste exploits this observation to actually loads in memory only  $|P|$  ratings, together with all user profiles  $|U|$  (each consisting of the user's identifier, the users' preferences and an item-to-rating map),  $|I|$  item entities, and  $|U| * k$  neighbourhoods. To do so, Taste builds on four main types of objects: *Users*, *Preferences*, *Neighborhoods* and *Items*.

*User* entities model users and assemble for each user her ID, the collection of her *Item-Preference* pairs, an array of *Preference* objects – one for each preference expressed – and her neighbours – captured and pre-cached in a *Neighborhood* object. In addition, *Item* entities save item IDs and their recommendable status.

Even without considering object references, Taste entails the load in memory of  $|P|$  *Preference* objects,  $|U|$  *Users*,  $|U|$  *Neighborhood* objects and  $|I|$  *Items*. The memory allocation implied by Taste is thus  $O(l + n + m)$ .

- differRS uses a radically different approach: as explained in Chapter 2, in fact, it only stores item entities (simply corresponding to item IDs) and, for each of them, the sets  $P_{i,r}$ , each collecting the IDs of users that have expressed a preference for item  $i$  equal to  $r$ . In so doing, it only stores  $|I|$  items and  $|P|$  ratings (i.e., user IDs), with an overall memory allocation of  $O(m + l)$ . Hence, even if we assume, as in the worst case scenario, that all users are atypical, differRS's Rating Hash Tree structure significantly reduces memory overhead, and it scales well with a growing number of rating users. In practice, the number of individual users (for which ratings are loaded in primary memory) is much smaller than the whole rating community, so that the amount of data (i.e., ratings) loaded is much less than  $|P|$ , as estimated by the worst case scenario. In particular, we observe that, both for the MovieLens Light and Yelp datasets, with  $\alpha = 1$  (the value we used throughout our experiments), above 80% of users would be classified as mass-like-minded, and as such only the preferences of the remaining 20% of atypical users would have to be loaded in memory.

### Processing

We evaluate diffeRS processing gain in comparison to existing recommenders in terms of additions, subtractions, multiplications and divisions needed. The main difference between existing recommender systems and diffeRS is in the computation of users' neighbourhoods. We thus analytically quantify the processing cost of computing similarities using diffeRS correlation and a standard Jaccard Pearson correlation. In other words we size the processing needed for *diffeRS* to compute:

$$w_{a,u} = jaccard(a,u) \frac{\sum_{i \in I_a \cap I_u} \left(1 - \frac{|r_{u,i} - r_{a,i}|}{|R|}\right)}{|I_a \cap I_u|} \quad (3.1)$$

and for usual recommenders to calculate:

$$w_{a,u} = jaccard(a,u) \frac{\sum_{i \in I_a \cap I_u} (r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i \in I_a \cap I_u} (r_{a,i} - \bar{r}_a)^2 \sum_{i \in I_a \cap I_u} (r_{u,i} - \bar{r}_u)^2}} \quad (3.2)$$

We can preliminarily observe that the Jaccard correlation is used in both similarity measures. As we are interested in understanding the gain of *diffeRS* over standard methods, we can hence avoid to gauge the effort to calculate it. We also remark that the  $|I_a \cap I_u|$  is indeed calculated already when estimating the Jaccard correlation. In fact:

$$jaccard(a,u) = \frac{|I_a \cap I_u|}{|I_a \cup I_u|}$$

Hence, no extra processing is needed for it. Also, it is worth noting that, as a subtraction can be thought of as the addition of a positive value to a negative one, and a division can be also modelled as the multiplication of real numbers, for simplicity we assess by number of additions (i.e., additions or subtractions) and multiplications (i.e., multiplications or divisions).

In the following analysis, we consider the worst possible case in terms of calculations needed; therefore we assume all users to have graded all items, and all items to have received ratings that span the full range  $R$  of possible rating values. This second assumption is used for the worst scenario in *diffeRS*. In fact, as explained in Section 2.4.1,  $|r_{u,i} - r_{a,i}|$  is just computed once for all users in  $P_{i,r}$ . In the worst scenario, all items would have votes for all possible values in  $R$ .

As before, we define  $n = |U|$  the number of users,  $m = |I|$  the number of items, and  $\rho = |R|$  the (constant) number of distinct rating values; we use the symbol  $\oplus$  to denote sums (or subtractions),  $\otimes$  for multiplications (or divisions) and  $\nabla$  for root squares.

- Recommender systems that use Jaccard Pearson Correlation to compute users' sim-

ilarity  $w_{a,b}$  would require the following computation per pair of users:

$$w_{a,b} = \frac{|I_a \cap I_b|}{|I_a \cup I_b|} \frac{\sum_{j=1}^m (u_a[j] - \bar{u}_a)(u_b[j] - \bar{u}_b)}{\sqrt{\sum_{j=1}^m (u_a[j] - \bar{u}_a)^2 \sum_{j=1}^m (u_b[j] - \bar{u}_b)^2}}$$

As highlighted, the Jaccard coefficient  $\frac{|I_a \cap I_b|}{|I_a \cup I_b|}$  is used in diffeRS too, and as we are interested in quantifying just the gain of diffeRS over standard methods, we can avoid to consider it. In this case, the above equation requires two subtractions, one multiplication and two squares (i.e., two multiplications) for *each item* that users  $a$  and  $b$  have rated in common. In the worst case, all users have rated all items, so these operations have to be repeated  $m$  times. Furthermore, the addends in each of three series have to be summed up (requiring  $m$  sums each), those at the denominator have to be multiplied once, and a final square root has to be computed. This results in:

$$(2 \oplus + 3 \otimes) * m + m \oplus + m \oplus + m \oplus + \otimes + \nabla =$$

$$5m \oplus + (3m + 1) \otimes + 1 \nabla$$

If we look at the processing cost to compute the similarity of all users to an active user  $a$ , we observe that then the effort is:

$$5(n-1)m \oplus + (n-1)(3m+1) \otimes + (n-1) \nabla$$

- diffeRS uses Equation 3.1 (reported below for clarity) in order to compute users' similarity:

$$\begin{aligned} w_{a,b} &= \frac{1}{|I_a \cup I_b|} \sum_{j \in I_a \cap I_b} \left( 1 - \frac{|u_b[j] - u_a[j]|}{\Delta_r} \right) \\ &= jaccard(a, b) \frac{\sum_{j \in I_a \cap I_b} \left( 1 - \frac{|u_b[j] - u_a[j]|}{\Delta_r} \right)}{|I_a \cap I_b|} \end{aligned}$$

As before, we ignore the Jaccard coefficient. As we have discussed in Section 2.4.1, this formula is computed incrementally, by exploiting the Rating Hash Tree structure. In particular, when all users' similarities to a same user  $a$  have to be computed, the sets  $P_{j,r}$  are traversed,  $j \in I$ ,  $r \in R$ . For each of these sets, diffeRS calculates  $|r - u_a[j]|$ ; therefore, if we assume all items to have received ratings across all distinct values in  $R$ , then to *each item*  $j \in I$  corresponds  $\rho$  subtractions, thus  $\rho * m$  subtractions for all  $m$  items. In the worst case scenario, each user has rated all items, so that to *each user* will correspond  $m$  sums, for a total of  $m(n-1)$  additions, when pondering all users but  $a$ . Finally, a normalisation step is required; we can do that either on a  $P_{j,r}$  set basis, or once for each user when finalising a user similarity. In the first case, we would execute  $\rho m$  divisions, in the second case  $n-1$ . Consequentially,

to determine *all* users' similarity to *a given user*, we would execute:

$$((n-1)m + \rho m) \oplus + \min(n-1, \rho m) \otimes$$

By comparing the two efforts to calculate all users' similarity, we derive that diffeRS processing reduction is (on a per operation type) basis:

$$[4m(n-1) - \rho m] \oplus$$

$$\max(3m(n-1), (3m+1)(n-1) - \rho m) \otimes$$

$$(n-1) \nabla$$

To give a flavour of the actual gain achieved, the following table reports the cost incurred to compute users' similarities in the worst case scenario described above (i.e., all items rated by all users, and all rating values used), when using Pearson correlation (CF) and when using diffeRS (dRS) with the settings used during the experimental evaluation discussed in Section 3.2.1 (i.e., threshold deviation to detect atypical users  $\alpha = 1$ , rating scale  $\rho = 5$  and MovieLens Light (ML) and Yelp (YLP) datasets).

	<i>method</i>	$\oplus$	$\otimes$	$\nabla$
$w(a, x) \forall x \in U - ML$	CF	7767k	4661k	942
$w(a, x) \forall x \in U - ML$	dRS	231k	153	0
$w(a, x) \forall x \in U - YLP$	CF	23511k	14108k	999
$w(a, x) \forall x \in U - YLP$	dRS	1809k	35	0

As shown, diffeRS reduces cost by several orders of magnitude overall, partly because of a cheaper way to compute (atypical) users' similarities, and partly because, for mass-like-minded users, simple item averages are returned at zero-cost.

### 3.3 CReSy Evaluation

#### 3.3.1 Performance

We now turn our attention to the CReSy component. Note that CReSy is not a standalone recommender service; rather, it aims to complement one (like diffeRS), to mitigate the so called cold-start problem, that is, lack of enough training data to be able to infer a prediction for composite items. In the following, we first define the methodology we have

been following when conducting our evaluation, the metrics applied, the datasets exploited and the benchmarks; we then analyse the results obtained.

## Methodology

When analysing CReSy we are interested in its qualities in advising users on (i) composite items and (ii) elementary items. To assess them, we examine the underpinning algorithms separately: we first study CReSy as a compositional recommender system; then, we focus on CReSy as a de-compositional recommender. As for differRS, for both cases, we follow the same de-facto standard methodology used in recommender systems research: we take a rating dataset and split it into a *training set* and a *test set*. The former represents knowledge that the recommender system can leverage; the latter provides both the test cases (i.e., the user  $u$ -item  $i$  pairs for which CReSy will try to compute a prediction  $p(u, i)$ ), as well as the actual preferences  $r_{u,i}$  registered by the user. We feed the training set to the recommenders in study; we then run the use cases in the test set and compare estimated preferences against real preferences; subsequently, we analyse the results and we derive key metrics. This methodology is used for assessing CReSy as a recommender for composite items or elementary ones with a main difference: in the first case, the test set includes just preferences expressed on compositions (as we just examine the predictions on composite items); in the latter case, the test set consists solely of preferences expressed on elementary items (as we analyse just the estimates on component items).

## Metrics

By design, CReSy increases prediction coverage, so that a user’s preference about a never-rated-before composition can be computed starting from (a subset of) its constituents; and vice-versa, the preference for a never-rated-before component can be predicted starting from the compositions it contributed to. We are thus not interested in measuring coverage; rather, we aim to quantify how accurate (in terms of Mean Absolute Error) these new predictions are.

## Datasets

In order to evaluate CReSy, we required a dataset containing users’ feedback on both composite items and their constituents. AlbumRankings<sup>6</sup> is an online music community web site that provides this information, as it collects users’ feedback on both albums/playlists (i.e., our compositions) and their songs (i.e., our constituents). We crawled the website in October 2009, and collected 13,176 ratings about 8074 albums/playlists, and a further

---

<sup>6</sup><http://www.albumrankings.com/>

38,294 ratings about 33,312 songs, thus a total of 51,470 ratings (all in a scale 0.5 to 5) made by 707 users. The basic idea here is that songs can be listened to and appreciated in isolation (like individual services); however, they can also be listened to as part of albums/playlists, thus impacting the feedback that the user gives to the aggregated content as a whole (the composite service).

### Benchmarks

To assess the accuracy of CReSy in recommending composite items, we have adopted two reference benchmark methods: the first method – *comp average* – estimates a user’s preference on a composition as the average feedback expressed by the user on its components; the second method – *comp minimum* – is more conservative, and returns the minimum rating that the user has given to any element in the composition.

To evaluate CReSy’s accuracy in predicting preferences on elementary items, we compare CReSy against two other benchmarks. Both these methods consider all compositions an elementary item  $i$  is element of and compute the value  $x$  that equals  $r_c = f(x, \text{ratings on known elements})$  to then return their average  $\bar{x}$ . In the first method (*reverse average*),  $f$  is the *average* function; whilst in the second method (*reverse diversity theorem*),  $f$  is the *diversity* function.

### Setup

We have conducted two sets of experiments. One studying CReSy accuracy in predicting users’ preferences on compositions, and one aimed at examining CReSy when advising users on elementary items.

For the first set of experiments, we have split the data in AlbumRankings so to have as training set all users’ preferences on elementary elements, and as test set all ratings on compositions. Hence, we have trained each of the algorithms in study with the training data set and we have run all the use cases in the test set to then compare results.

The experiment setup for measuring CReSy accuracy in predicting users’ preferences on elementary items was more traditional and slightly different. More precisely, we split the rating dataset in an 80% training set and a 20% test set (we used an 80-20 split, rather than a 90-10 as we did for differS, in order to have a higher number of test cases); the split was repeated 10 times at random. We then feed the training set in input to CReSy, compute predictions for the test set, and compare the estimated preferences against actual ones. In this case, the training set contains both elementary and composite items, as both their ratings are used by the benchmark methods.

## Results

The first set of experiments we conducted on CReSy aims at assessing its performance as a recommender service for *composite* items; for each composition in the test set, a preference has been predicted using CReSy, and the MAE (with respect to the actual user's preference) recorded. Figure 3.11 plots the recorded error on a per user basis, together with our benchmark results (i.e., MAE recorded using *comp average* and *comp minimum*); to ease comparison, the average MAEs recorded with each of the three methods, over the whole experiment, are also plotted (as continuous lines). As shown, CReSy outperforms both benchmarks, and achieves a MAE well below 0.5, thus demonstrating the suitability of the Diversity Prediction Theorem (Equation 2.8) to model users' preferences over composite items.

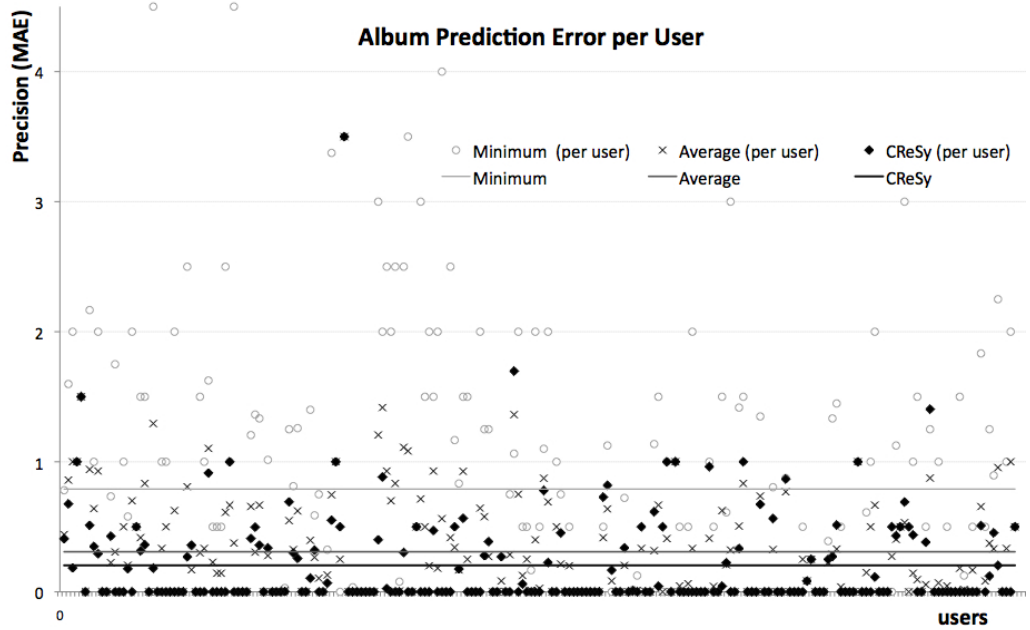


Figure 3.11: Album (Compositions) Predictions

The second set of experiments appraises CReSy in predicting users' preferences on component items which have never been used in isolation. Figure 3.12 reports the results of the recorded MAE, in comparison with the error recorded when using *reverse average* and *reverse diversity theorem* as benchmarks; as before, the average MAEs recorded with each of the three methods, over the whole experiment, are also plotted (as continuous lines). As shown, CReSy outperforms both benchmarks, even though the gain is less marked in this case. Most importantly, the MAE is well below 0.5, thus demonstrating once again that CReSy is a good model to represent users' preferences over component items that have not been used in isolation before.

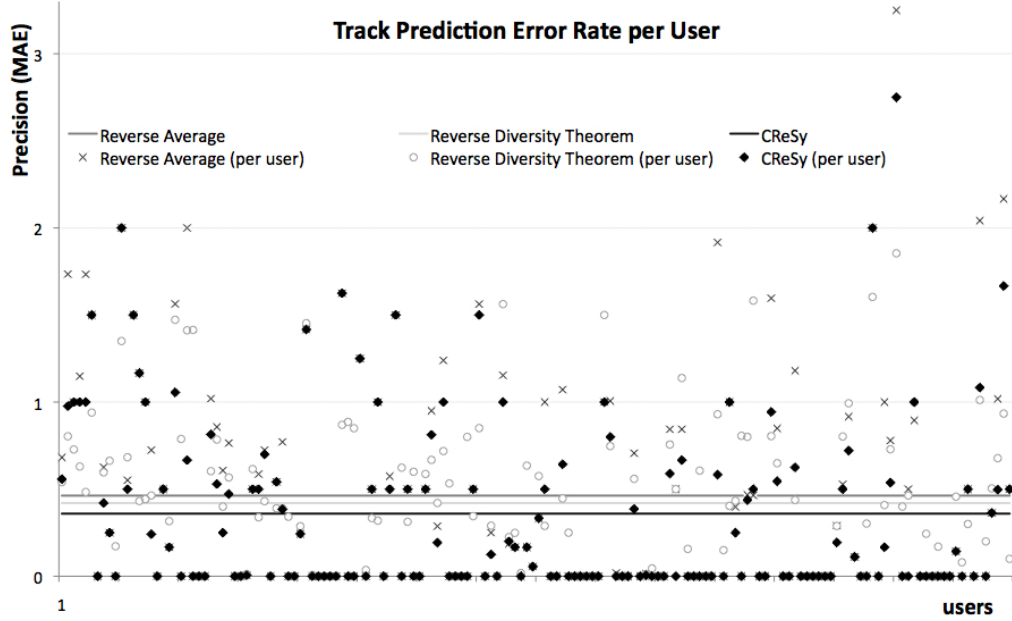


Figure 3.12: Track (Component) Predictions

### 3.3.2 Overhead

We now turn our attention to the overhead of CReSy in terms of memory allocation and processing cost.

#### Primary Memory Allocation

By its very nature, CReSy has a very limited footprint. This is because it builds on top of an existing recommender system (diffERS, in the moRe framework). In fact, CReSy does not persist users' ratings itself; instead, it relies on the underpinning recommender to make them accessible. In addition to users' rating data, CReSy exploits information about the composite nature of the elements that can be recommended. When advising users on composite items, CReSy capitalises on data about their components, whilst when predicting users' preferences for items never used in isolation, CReSy leverages on information about items they are an element of. As explained in Section 2.4.2, in the first case, CReSy does not need to store new data, whilst in the second case CReSy requires to persist pointers to the elements they are component of. These mapping bags are persisted, thus not impacting primary memory allocation; however if, for performance reasons, we decided to load and cache part of the item-to-compositions map in memory, the extra memory footprint would be very limited. For instance, for the AlbumRankings dataset, if we were to cache maps for those individual items affecting at least 5 users in the rating community, the overall additional memory allocation would be less than 1 kB; if we

were to load maps about any item impacting at least one user, that would entail an extra memory footprint of 29 kB overall.

### Processing Cost

In order to assess CReSy processing overhead, we have analytically quantified its cost in terms of additions, subtractions, multiplications and divisions needed. As before, we indicate with  $I_c$  the set of items which are part of a composition  $c$ , and  $C_i$  the set of compositions an item  $i$  has been part of; we use the symbol  $\oplus$  to denote sums (or subtractions),  $\otimes$  for multiplications (or divisions) and  $\nabla$  for root squares.

To estimate the preference that an active user  $a$  would attach to a composite item  $c$ , CReSy uses Equation 2.8, that we can write in the following form if we use  $x_i = u_a[i]_{i \in I_c}$ :

$$p(a, c) = \overline{x_i} + \sqrt{\frac{\sum_{i \in I_c} x_i^2}{|I_c|} - \overline{x_i}^2}$$

We observe that  $\overline{x_i}$  corresponds to  $|I_c|$  sums and 1 division; also,  $\sum_{i \in I_c} x_i^2 / |I_c|$  entails  $|I_c|$  products,  $|I_c|$  sums and 1 division. Overall, to compute  $p(a, c)$  CReSy would execute:

$$2(|I_c| + 1) \oplus + (|I_c| + 2) \otimes + 1 \nabla .$$

To predict a user's preference for an item used in isolation, Equation 2.9 would be used, which simply entails the computation of an average:

$$|C_i| \oplus + 1 \otimes .$$

## 3.4 Summary

In this chapter we have analysed the recommender system proposed in Chapter 2, both in terms of performance and resource allocation. To do so, on one side, we have studied the precision of the predictions made and their coverage. On the other side, we have assessed its processing cost and memory allocation. Overall, we have demonstrated that the mobile recommender system proposed achieves an accuracy/coverage that is comparable (and in some cases better) to traditional recommenders, but it does so for a negligible resource overhead compared to them.



## Part III

# Reliable Discovery



## Chapter 4

# Reliable Discovery of Pervasive Services

Service providers as we know them nowadays are the always on ‘static’ web service providers that aim at Five 9s availability (99.999%). Formal, or de-facto, standards, such as WSDL [Christensen et al., 2001], WADL [Hadley, 2009], OpenSearch [Clinton, 2009] and BPEL [Coalition, 2003], have become technology enablers for the easy discovery, use and coordination of such services. However, we envisage tomorrow’s services to become increasingly pervasive, being deployed within buildings, transport systems, markets, as well as people portable devices. Such services will be, by their very nature, simple and fine-grained; as a consequence, service composition will become crucial to deliver rich functionalities that satisfy end users’ requests. Composing services in mobile environments opens up significant challenges. In particular, the Five 9s availability assumption no longer holds: the higher the dynamic nature of the environment, the higher the chances that services will move out-of-reach before the composition completes, causing the service as a whole to fail.

We argue that, in order to enable the successful completion of compound services in mobile environments, the reliability of the composition must be measured and reasoned about. In order to do so, we reason about service providers’ historical co-location patterns and the composition semantics of composite services. On one side, we learn providers’ collocation patterns over time to predict the duration of co-location between component services. On the other side, we analyse the composition semantics to deduce the co-location requirements on component services that would ensure composite services to be executed successfully. Consequently, we ponder the co-location estimates against the co-location requirements inferred to then (i) decide whether a composition should be attempted or not and (ii) to select the providers’ in the surroundings that maximise the reliability of a composition. In addition, we monitor unforeseen changes to co-location patterns, triggering re-bindings during service execution.

In the following, we first discuss existing research on the subject of service discovery, composition and adaptation (Section 4.1). We then look at the challenges and requirements the discovery of composite services in mobile environments entails (Section 4.2). Hence, we present our main contributions in this area: a Mobility Predictor, a Composition Semantic Reasoner, and an Adaptive Binder, the underpinning observations (Section 4.3) and algorithms (Section 4.4).

## 4.1 Background

The work presented in this chapter relates to four different areas: (i) service discovery; (ii) context/aware services and adaptation; (iii) service composition and orchestration; (iv) session management. In the following, we present the state of art for each of these areas.

### 4.1.1 Service Discovery

A relevant number of research projects have been focusing on service discovery for mobile distributed systems and, separately, on service discovery for web-based composed services. Very little attention has been accorded so far to service discovery of composite services in mobile environments.

Most of the activity related to mobile environments has been directed to conceive architectures and protocols that could best satisfy the seamless connectivity of mobile networks. As well as for traditional networks, the approach taken for mobile networks entails the use of service directories, either implemented as centralized repositories or distributed ones (e.g., Jini [Waldo, 1999], UDDI [Coalition, 2000], UPnP [Thomas et al., 2004], Bluetooth discovery protocol [Mettala, 1999], etc). For what concerns distributed methods and nomadic systems, the research work has been aimed to optimize the bandwidth, the energy consumption, as well as the number of user agents a service is advertised to. Approaches presented in literature include: broadcasting, single-hop bordercasting, multi-hop bordercasting, routing based algorithms, rendez-vous discovery, semantic routing, etc. [Sailhan and Issarny, 2005, Capra et al., 2005]. Some of them integrate ad-hoc networks routing protocols with discovery mechanisms to reach the best results [Chakraborty et al., 2006]. Regardless of the distributed or centralized nature of service repositories and the approach taken, most of the methods investigated could effectively return a multitude of possible service instances. On this matter, a considerable amount of research has been carried on models and protocols considering QoS and contextual parameters when selecting services among the ones returned, to provide adaptive service discovery [Capra et al., 2005] [Liu and Issarny, 2004]. In terms of discovery of composite services, some research has recently

been investigating the service discovery in terms of service aggregation, with the goal of satisfying a client request expressed as another service. This activity has mainly been focused on rich interoperable service descriptors [Patil et al., 2004a] [Brogi and Popescu, 2006] [Williams et al., 2003] and service aggregation methodologies [Brogi and Popescu, 2006] that, given a registry of (advertised) services and a client service request, automatically generate compositions of services that satisfy the client request. Three main types of methodologies exist when it comes to service aggregation: manual, semi-automatic and automatic. Manual service composition entails the requester to browse the registry, find the desired service operations, model their interactions into a flow structure (mainly with BPEL [Coalition, 2003]) and then expose the final service as a unique service using WSDL [Christensen et al., 2001]. This methodology is effectively employed today within the Web Service Industry [Brogi and Popescu, 2006]. Semi-automatic composition of services usually involves a service composition system that interacts with the requester in an iterative manner in order to obtain information about the requested service, and to construct aggregated services out of the registered ones [Brogi and Popescu, 2006] [Svensson Fors et al., 2009]. The automatic composition, instead, demands the existence of a discovery agent that receives a service request and then it generates a structure of services/operations of some registered services based on the information provided in the request [Brogi and Popescu, 2006]. Most automatic aggregation approaches rely on services to be richly described either by means of semantic Web Service languages (OWL-S [Martin et al., 2004] and WSCI [Arkin et al., 2002]) or Web Composition industry languages as BPEL [Coalition, 2003]. In both approaches, services are usually exposed using WSDL [Christensen et al., 2001].

The latest approach – autonomic discovery – is the one that has recently attracted the most interest from research, also due to the recent uptake of service matchmaking and selection challenges (e.g., S3 [S3, 2007] contest and Semantic Web Services Challenge [SWS, 2008]). It mainly focuses on identifying services and service providers able to fulfil the requirements of a service request. Several methods have been investigated and proposed. They mainly use one of three approaches [Klusch et al., 2006]: structural discovery, lexical discovery and logical discovery.

- Structural discovery compares syntactical information, like the interface description and the data exchanged. It requires the service requester to specify structural requirements (e.g., operation signature, data type, parameters and services).
- Lexical discovery approaches use natural language descriptions. They use operation names or descriptions that come with standards like WSDL [Christensen et al., 2001]. The lexical algorithms operate as follow: they first normalise the descriptions (by removing stop words and expanding keywords to their synonyms) and then they compute similarity scores between the service request and any possible matching service.

- Semantic discovery uses formal methods to describe and reason on service capabilities and properties. It effectively relies on accurate ontology descriptions and it uses machine reasoning to identify possible candidates for a service request. The main idea is that services can be ontologically described in terms of their classification, input, output, preconditions and postconditions sets. Subsumption relationships can be then used on those entities to decide the degree of matching of two services (e.g., exact match, plug-in, subsume, subsumed by).

Service composition in pervasive environments requires this kind of automatic (de)-composition. However, the very nature of the environment opens the door to new challenges that limit the applicability of current technologies. For example, resource limitations on the target devices have called for novel solutions to enable efficient ontology-based semantic matching of services [Mokhtar et al., 2006]; more flexible approaches enabling the on-demand creation of an agreed ontology are being investigated too [Williams et al., 2005].

Our work is orthogonal to the issue of semantic service matching, and we thus leverage on top of existing solutions. More closely related to our work are approaches that take into consideration mobility of devices, and thus services. In [Capra et al., 2005] [Liu and Issarny, 2004] single service discovery protocols have been proposed that aim to find the device capable of delivering the best quality of service, given the dynamicity of the current environment; the discovery protocol has also been extended to consider multi-hop networks [Sailhan and Issarny, 2005]. We argue that, while important, QoS reasoning must come *after* co-location reasoning, especially for services that require more than just a few seconds to complete. For the same reason, multi-hop service discovery and delivery are promising only for services that execute very fast, and/or are deployed in fairly stable scenarios.

Very little work has been done to ponder providers' mobility (and mobility patterns) and adapt the service discovery to a particular composition semantic - let alone composition - in mobile environments. In mobile systems, service providers may or may not be available for the entire duration of the service; if we relate this to the dynamic nature of a given mobile system (meant as a measure of the number of service providers that appear/disappear and the overall heterogeneity of the services on reach), this may be a problem or actually an advantage. If a component service is not found in an highly dynamic system, chances are that it may become available before the completion of the compound service; on the other hand, even if instances for all the component services are available at a given instant, it is possible that those instances will be no longer accessible by the time they are needed within the composition. Our research considers these and similar aspects when modelling service discovery.

We build our proposal on top of previous research and model discovery independently from the actual discovery method used. More precisely, we refine the underpinning discovery

mechanism to maximize the likelihood a composite service is executed successfully. To do so, two main concepts are introduced: stability area and composition stability.

With the first one, we refer to the virtual area for which it makes sense to select services providers from. As later explained, this area is not defined by geographic properties but by a minimum stability level, which in real terms corresponds to a minimum co-location interval. As such, it groups all providers that are believed to stay co-located with the consumer long enough for the execution of the service requested. A stability value – minimum co-location interval – defines the size of this set. The higher is the stability requested by the system and/or application, the narrower is the area and vice-versa. In practise, the stability range is used as a QoS parameter for the supporting discovery method in use.

The second one, composition stability, refers to the overall stability of the possible set of service instances returned. Here the goal is to maximize the chances that a service is either completely executed or not executed at all. Because the intent is to privilege the overall stability, our framework will advantage the set of service instances running by the same host (or host infrastructure). In order to do so, when a request for a component service is received, it is iteratively analysed to outline the component services which can themselves be compound. At each refinement step, the composite services are searched for: if an instance for all of them does not exist, then the composite service is further analysed until none of the component service is a compound service itself.

### 4.1.2 Context-Aware Services and Adaptation

Until recently, most research conducted in context awareness has focused on location, time, and proximity-based contextual factors. On this basis, Schilit and Theimer [Schilit and Theimer, 1994] defined context-aware computing as software that ‘adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time’. Advances in wireless networking, and the advent of an increasing number of new appliances (often constrained) accessing the Internet, provided the basis for a more general definition of context, and a widening of the scope of context-enabled applications. Early acceptance of the Schilit and Theimer definition of context-aware computing has been largely replaced by the more general definition offered by Dey and Abowd: ‘A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task’ [Abowd et al., 1999] and ‘the context can be considered to be any information describing the situation of an entity’ [Abowd et al., 1999]. Therefore, context information generally includes and describes the attributes that can be utilized in adaptation and personalization of services.

Different ubiquitous and pervasive computing architectures have been studied and proposed supporting adaptation and context-awareness [Sailhan and Issarny, 2005], [Capra et al., 2005], [Chakraborty et al., 2006] and [Brogi and Popescu, 2006], and frameworks

have been designed to abstract the complexity of context-aware services [Liu and Issarny, 2004].

Historically, the proposed solutions made strong assumptions on the entities delivering, demanding or adapting a service, as well as on the entities supplying or retrieving the context. The mobile devices were mainly modelled as entities consuming adaptable services, whilst appliances in the proximity were modelled as contextual entities. The architectures developed then regarded mobile devices as just high-tier terminals and did not take into account other constrained appliances. Recent research [Brogi and Popescu, 2006] looked more in depth to mobile environments and it did consider situations where (i) the service provider, adaptor and consumer are connected in peer-to-peer or ad-hoc networks; (ii) the same entities retrieving the context and/or adapting the services are themselves mobile, battery dependant and constrained; (iii) the profile of the client device can vary, perhaps even during the execution of the service (e.g., change in network bandwidth, battery power, connectivity, etc). More precisely, recent activities have been focused on mobile computing middleware to enhance the construction of adaptive and context-aware mobile applications. As a result, they have provided highly effective and flexible models and means to handle and describe declaratively context and context policies, as well as algorithms to resolve conflicts between those last [Brogi and Popescu, 2006]. Some research [Julien and Roman, 2002] [Murphy et al., 2001] has also been paying a lot of attention to the dynamic aspect of mobile context-aware services as opposed to usual context-aware applications. In fact, the mobility of hosts and software agents adds a new degree of complexity to this changing environment. As hosts and agents become more mobile and travel to completely new environments, mobile environments exhibit frequent configuration changes and a great deal of resource variability. Therefore, services executing under these circumstances need to react continuously and rapidly to changes in operating conditions and must adapt their behaviour accordingly. This has lead to redefine the notion of context and the mechanisms used to manage contextual changes. New means to represent contextual information have been introduced: for example, tuple spaces [Gelernter, 1985], which then have evolved to an agent centred notion of it called views [Picco et al., 1999], where a view abstracts and collects information about the data and resources within a subnet surrounding an agent. Multiple views can be associated to an agent and their content is defined declaratively and may change over time.

Little research has been conducted so far to handle contextual information for composed services. Not only contextual information will drive the selection of service providers, but also it will feed the overall composed service while it executes.

Whilst we rely on existing research [Julien and Roman, 2002] [Murphy et al., 2001] to handle context policies within mobile environments for each of the services, we will be orthogonal and independent to it as for the scope of this work. As a consequence, it would be transparent to our research how adapting services/modules will perceive contextual changes and it is here assumed they do so atomically, as if in a static environment (i.e.,

they will inspect the contextual changes they are notified about against the contextual policies they are configured with and will adapt accordingly). We will simply develop on top of context-aware services an extra layer to adapt and adjust the composition to the mobility context (i.e., providers in the surrounding and their mobility patterns).

### 4.1.3 Service Composition and Orchestration

Service Composition, that is, the development of customized services by discovering, integrating and executing existing services [Chakraborty et al., 2005], has received a lot of interest since the advent of Web Services and has become a workable and broadly adopted technology thanks to real or de-facto standards such as WSDL [Christensen et al., 2001], SOAP [Box et al., 2000], UDDI [Coalition, 2000] and BPEL4WS [Coalition, 2003] and more recently lightweight standards for stateless services like REST services [Fielding, 2000], WADL [Hadley, 2009] and OpenSearch [Clinton, 2009]. This attention has mainly concerned the Internet and hence wired-environments, where the service providers are static and well known.

Research on Service Composition for the Internet has followed, in the past, one of two main directions. The first one aims to augment service descriptions in order to improve service matching, it extends the properties of service composition and automates both service discovery and composition itself. Various languages have been developed to enhance service description by adding (i) semantic information and/or (ii) protocol information. Examples of the first type of enhancing language are WSDL-S [Akkiraju et al., 2005], OWL-S [Martin et al., 2004], or METEOR-S [Patil et al., 2004a]; examples of the second one are BPEL4WS [Coalition, 2003], WSCDL [WSCDL Coalition, 2005], METEOR-S [Patil et al., 2004a], OWL-S [Martin et al., 2004], and recently YAWL [van der Aalst and ter Hofstede, 2005] add protocol information to service descriptions.

A second direction has focused on to the interoperability between composed services. In fact, semantic descriptions by definition rely on an ontology [Williams et al., 2005] [Dogac et al., 2003]. Two main approaches exist: one makes use of AI to weight words or compute concept/terms ‘distance’, sometimes focusing on local consensus; others are built around additional operational information and data used to describe services.

Another aspect that is fundamental to Service Composition and has been the subject of recent research [Vukovic et al., 2007] [Chakraborty et al., 2005], is the ability to translate a requested service in a set of composing ones. This activity has mainly resulted in the development of methodologies for aggregating services with the goal of satisfying a client request.

The challenges are multiples: firstly, services are still developed using a variety of models and languages thus preventing a common understanding; secondly, their successful proliferation imposes to look up for services across either numerous or large repositories causing

non trivial scaling issues; thirdly, because of their dynamic nature – services can be created/updated on the fly - flexible compositions effectively have to be formed at run time (i.e., the composition plan needs to be created at run time). Several methods have been proposed for enabling service composition (in addition to service matchmaking). Two main approaches exist: workflow composition and AI planning.

The first approach is mainly used for services, whose composition meta-flow is known before execution. As for this approach, a composite service is a more or less sophisticated workflow controlling, monitoring and transforming the results of a set of atomic services. As such, it allows to automatically bind service providers, execute data transformation, monitoring and alert at run time, but it does not permit to create the composition structure on the fly. In the recent years, research on the workflow approach has been more and more focused on the dynamic aspect of the composition. On one side, it has aimed mainly to ease the reuse and update of composition descriptions and, on the other side, it has focused on loosely-coupled business rules and the composition (or core business process). The interest here rises, as business rules tend to change faster than the core business processes. They are in fact very volatile, as they tend to adapt to business requirements [Cibran and Verheecke, 2005]. Languages as BPEL, for instance, depict business processes as monolithic specifications. They do not support the definition of business rules in a clean, modularized and reusable way, so that the specification of the rules gets tightly coupled with the composition itself. Changes to the workflow, due to changes in business requirements, need most of the time to redefine the whole composition. This research has produced models for which business rules are decoupled from actual composition. Those models have been mainly built around the notion of aspects and Aspect Oriented Programming (AOP). They basically allow specifying composition rules in a very declarative way, without having to be aware of specific AOP constructs. They do so by using AOP as an underlying layer to realize the connection of the business rules with the core composition.

AI Planning has targeted the dynamic composition at a greater extent, but it mainly remains a research approach not yet embraced by the industry. It is built upon the notions of actions and the world they act upon. In AI Planning, each service represents an *action*, whilst the *preconditions* and *effects* are the input and the output parameters of the service respectively; the execution environment is then the world they act upon. As such, the precondition is the status of the execution environment (world) before the execution of a service (the action), whilst the new state of the execution environment after the execution of a service is simply the effect of it. Each service alters the states of the world after its execution. Hence, a composition involves the construction of sequences of actions (the component services) to achieve a goal (the end status and output of the composite service). As a composite service is described by the preconditions and effects required by it, then the AI planner reasons about the effects of actions to produce automatically a ‘composition’ plan.

Lately, the dynamic aspect of Service Composition has been further enhanced with the increasing interest of the academia in ‘pervasive computing’ environments. As part of it, recent research on Service Composition has been focusing on those factors that wired-infrastructure based service composition architectures did not consider and which are critical to mobile environments (i.e., device and service heterogeneity, resource mobility, variability and constrained processing power, battery, connectivity, reliability and availability). Some research work has been looking at service composition in infrastructure-less environments [Basu et al., 2003] [Chakraborty et al., 2004] [Gao et al., 2007], but it has ignored the mobility of devices.

Research on mobile environments is in very early days; the research conducted so far has produced mainly criteria to assess solutions targeting mobile environments [Chakraborty et al., 2005] [Svensson Fors et al., 2009].

To the best of our knowledge, no research has been conducted to make use of composition semantic information, as well as mobility, to drive on demand composition, failure recovery and on-the-fly adaptation of composed services.

In our work, we build orthogonally to existing autonomic composition research and focus on exploiting the composition stability, the environment mobility and the composition semantic to drive the composition, its dynamic context adaptation, as well as its self-healing. The goal is to minimise failure rates and maximise execution confidence.

#### 4.1.4 Seamless Session Management

Mobile environments entail a new concept of session. This is because the physical mobility of the devices may result in an interaction between hosts to need more time to complete than the interval of connectivity between them. The aim of the new session definition is primarily to allow reliable service provision, i.e., to maximize the chances that an interaction between the client and the service provider, once begun, reaches completion. In order to do so, research has been mainly focusing on mechanisms enabling the client to partially complete the task with the help of some host, pause its work, and resume it on another host.

In order to ease the development of applications for mobile environments, current research has focused on implementing a layer of abstraction to the pause-transfer-resume programming paradigm that is called follow-me session; follow-me sessions preserve, within limits, the sense of interaction between a client application and a service despite defective connectivity. Two main approaches can be used to implement the follow-me session: client-based and proxy-based (e.g., Jini [Waldo, 1999]). The major difference between the two models is that, in the first one, a service is essentially a process running on a well-known remote host and it is up to the client to know the correct protocol to contact and use the server.

However, in proxy-based systems, a service is a combination of a process running on a remote server and the proxy that it ships to the client. The client therefore needs to only make local method calls to the proxy to interact with the remote service. Four main strategies to delivery of ‘follow-me’ sessions exist: 1) strong migration of a process to an alternate host, 2) partial results migration to an alternate provider offering similar service, 3) client to temporarily disconnect from the service provider while the provider continues processing, and 4) client temporarily stores partial results until a suitable alternate service provider can be found. The combination of these strategies, depending on the actual context, has been proven to be effective [Handorean et al., 2006].

Beside session transfer, research has explored the use of different session semantics to drive service coordination and/or composition in mobile environments. As a result, a model has been developed to allow applications to specify the functional properties of the services to which they need to connect. In this model, composition needs are expressed in terms of application sessions, loosely defined by a set of interactions with remote resources, thus enabling to delegate the construction and maintenance of the communication links to an underlying middleware [Julien and Stovall, 2006].

So far, no research exists that has been looking at sessions that change their typology during the application and/or composing service lifetime. Nor we are aware of any research activity that aims to relate session typology to composition semantic. We address these two aspects next.

## 4.2 Challenges

To point out the main research challenges for selecting providers of composite services, we analyse and refine a scene of the scenario discussed in Section 1.2. In the following, we report the scene.

*Alice is on the underground on her way to work. At the time Alice gets in the underground, she has an application running on her mobile phone: the Smart Media Channel. The Smart Media Channel is an application creating personalised podcast channels for its users by gathering and playing audio and visual content for free from other devices, mocking the functionalities of radio and TV channels. Advertisements are injected from time to time, either interrupting the audio/video streaming, or by means of interactive banners. The content to be played, as well as the adverts to be shown/reproduced, are selected based on what is currently available in the environment, taking into consideration Alice’s tastes. Alice is listening to some audio news played by her Smart Media Channel. The Smart Media Channel has in fact selected an initial sequence of news among all service/content available in the neighbourhood in accordance to Alice’s tastes. As she gets on the tube, she loses global connectivity, so the Smart Media Channel application has to elect new*

*content and new service providers for it. In order to do so, it elects the items to be played next, based on Alice's preferences, and streams them.*

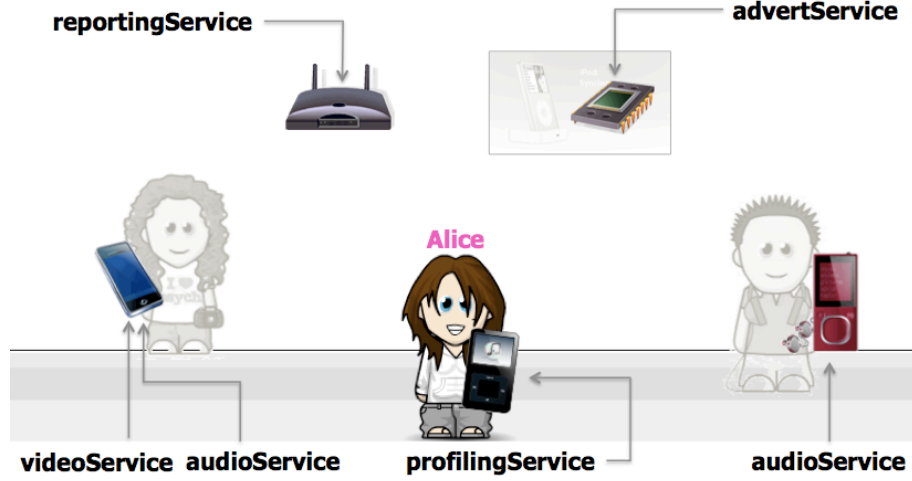


Figure 4.1: Scenario

This scene (illustrated in Figure 4.1) describes the coordination of several mobile services and it introduces a variety of composition semantics. For example, the media content selector and the advertising service both filter content based on Alice's profile first; as such, they need to be composed *sequentially* (in sequence) to an eventual profiling Service

$$\text{profilingService} \text{ seq } \text{content\&adService}$$

Depending on the actual context and the user's preferences, advertising may be shown or played, either *in parallel* to the content or in *any order* (e.g., before or after the content). In order to enable the selection of one or the other strategy, a *choice* composition semantics will be needed. The compound service *content\&advertService* can then be broken into:

$$\begin{aligned} &\text{choice } \langle \text{guard condition} \rangle \text{ in } ( \\ &\quad \langle v1 \rangle: \text{contentService} \text{ anyorder } \text{advertService}, \\ &\quad \langle v2 \rangle: \text{contentService} \text{ parallel } \text{advertService} \\ & ) \end{aligned}$$

The Smart Media Channel updates the list of the next-to-come audio, music content or

videos at a regular basis, so that the overall composition *loop* is started again:

```

loop < iterator, guard condition > (
  profilingService seq (
    choice < guard condition > in (
      < v1 >: contentService seq advertService,
      < v2 >: contentService parallel advertService
    )
  )
)

```

The *content service* picks and reproduces, at each time, just one of the media content actually available for the current user's profile. Depending on it, the content service may either just consider audio and/or video content. The content service can hence be decomposed in the following manners:

```

choice < guard condition > in (
  < v1 >: audioService,
  < v2 >: videoService,
  < v3 >: anychoicein (audioService, videoService)
)

```

In the first case, the content service would pick one of any of the audio services available; according to the second case, the selection would just consider video services, and in the third case the content service would elect the media service to execute among both the audio and video services available. In addition, the *advertising service* may notify another external one in the form of an asynchronous signal; that could be for instance the case for a reporting/logging service:

```

advertService signal reportingService

```

The full composition semantics of the Smart Media Player can thus be described as follow:

```

loop < iterator, guard condition > (
  profilingService seq (
    choice < guard condition > in (
      < v1 >: choice < guard condition > in (
        < v1 >: audioService,
        < v2 >: videoService,
        < v3 >: anychoice in
          (audioService, videoService)
      )
      seq (advertService signal
        reportingService),
      < v2 >: choice < guard condition > in (
        < v1 >: audioService,
        < v2 >: videoService,
        < v3 >: anychoice in
          (audioService, videoService)
      )
      parallel (advertService signal
        reportingService)
    )
  )
)

```

As the above scenario shows, pervasive services (e.g., Smart Media Channel) are often compound services, provided by aggregating more basic functionalities according to a variety of semantics (e.g., sequential, any order, parallel, choice in, any choice in, loop, signal). Some of these services will be local to the client's device (i.e., the device who is consuming the compound service), while others will be available from a combination of stationary providers (e.g., those embedded in the local space) and mobile providers (e.g., those provided by other people personal devices). Given the dynamic nature of the target scenario, with services appearing and disappearing all the time, as perceived by the client's device, it becomes crucial to: (1) reason about the providers' movement relative to the client's device; (2) select those providers that will maximise the chances of a successfully completed compound service; (3) capture and react to environment changes.

In the next section we present a mobile service selection framework – *moSS* – that supports this type of run-time reasoning, selection and adaptation, while hiding the exact topology of services making up a composition from both application engineers and end users.

### 4.3 Conceptual Model

moSS (*mobile Service Selector*) is a flexible and expansible framework that eases the development of applications requiring the dynamic and reliable composition of services in mobile environments, thus enabling mobile users to seamlessly and successfully consume compound services while on the move. More precisely, moSS looks at service and content discovery focusing on reliability and the dynamic aspects of pervasive services. This is because the main remarkable difference of compositions carried in usual web environments and those in mobile ones is that service (and content) providers are mobile with respect to the service consumer. Therefore, they may be not available for as long as required. On the other hand, successful pervasive services need to be perceived by users as supplied by a unique entity that is reachable and available for the duration of the service. To address this issue, moSS provides a mobility-driven filtering mechanism that allows to (i) initiate just compositions likely to be successfully completed, (ii) elect service providers that maximise reliability – also referred to as proximity availability (i.e., one-hop connectivity of a service provider to a service consumer) of the overall composition, and (iii) monitor and react to changes in the environment so to maximise the number of compositions completed. To do so, moSS builds upon the following three observations.

**Observation 1** – *Pervasive Services are provided by devices carried by people or by devices embedded in buildings or transports that people traverse.*

People show a high degree of regularity in their activities. They organise their time in agendas structured around the concept of days, weekdays and hours. They often travel to/from work on the same train, or fly home with the same flight at the same time of the same weekday. Also, they follow routines during their working days: they visit the same pub or restaurant on some weekdays; they go to the gym at regular times and days, and so on. So, although the number of unknown devices we encounter at any time will always be high, a non-negligible set of devices, either stationary or mobile, will be re-encountered regularly [Eagle and Pentland, 2007]. The availability of pervasive services (and content) is naturally correlated to users' habits. In fact, the relative mobility between service providers and consumers relates to people mobility pattern, either because those services (and content) are made available through people personal devices, or because they are embedded within structures that people traverse. Hence, pervasive services will exhibit the same temporal patterns characterising humans, which can be learned and crucially predicted.

**Observation 2** – *Not all component services are indeed needed for the whole duration of the composition they are part of.*

When we look at service compositions, we remark that a composite service is delivered as set of component services possibly run at different times and by different providers. Hence, if we were to estimate whether a service consumer could use a composite service, we would have to ponder each provider of its component services and impose all elected providers to be co-located with the service client for the entire duration of the overall service. We remark, though, that not all components are indeed needed for the entire duration of the composite service. For instance, if we look to the Smart Media Player service, the profiling Service is just required at the beginning of each loop.

***Observation 3** – Despite following regular patterns most of the time, humans do behave unpredictably from time to time.*

We observe that patterns describe typical behaviours and so they fail to capture exceptions; also, we remark that humans do break their habits or they change them, as such we need to cater for these exceptions and alterations as they occur. In other words, during the execution of a composition, changes to the environment and the composition providers set should be monitored, thus triggering re-bindings if necessary.

In moSS we exploit these observations as follow:

- From *Observation 1*, we propose a mobility prediction method that builds on people habits to estimate the co-location time for two devices at any given instant. In other words, we exploit users' device presence patterns to develop a technique that allows to estimate whether a service provider will be co-located to a consumer long enough for a service to be successfully executed
- From *Observation 2*, we build a method that examines the semantic of a composition - here defined as its structure - in order to determine *when* a component service (hence, its provider) is required and for how long.
- From *Observation 3*, we propose a method that monitors and reacts to environment changes (e.g., devices appearing or disappearing) and, where possible, it predicts such changes.

Overall, moSS leverages upon these remarks to estimate the probability that a given provider will remain co-located with the client's device (where the composition framework is deployed) for a given amount of time; then it uses these predictions, together with the specific composition semantics, to determine if a composition can be attempted (i.e., the probability of successful completion is high enough) and, if so, what instances to rely upon. Also, moSS monitors changes to the environment (e.g., devices appearing or disappearing) to then re-assess bindings while the service is being executed, in order to be able to react to unforeseen departures of elected providers.

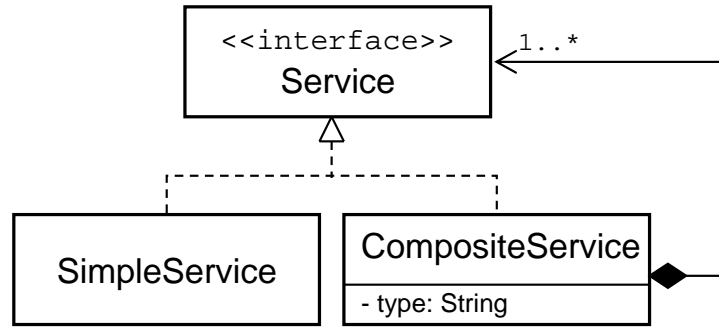


Figure 4.2: Service Entity

To do so, moSS builds on two key concepts: the *Service Entity*, that statically describes the composition structure of a service, as well as its classification according to a pre-defined ontology; and the *Binding Entity*, that captures the run-time association between a service entity and the node(s) which is(are) currently in charge of delivering such functionalities in the most reliable way.

These two entities are presented next.

#### 4.3.1 The Service Entity

A moSS Service consists of a *classification* and a *composition structure*. Classification information consists of two unique identifiers: the identifier of the service taxonomy (or ontology) in use, and the identifier of the service type within such taxonomy. We are making the assumption that a taxonomy can be identified by a unique universal ID (e.g., an URI) and that a local ID univocally identifies each service type within such taxonomy. The composition structure information is conveyed by the very same nature of the Service Entity, which is represented by means of the composite pattern depicted in Figure 4.2. Starting from basic services, composite services can be created and described using any of the moSS-supported composition semantics: **sequence**, **parallel**, **any order**, **choice in**, **any choice in**, **signal** and **loop**. In the following, we introduce them, together with their notations.

##### Sequence

The *sequence* directive describes the composition of two or more services that have to be performed in sequence. This is the case, for instance, of the profiling and content&advertising services in our scenario (Section 4.2), as the latter requires first to gather the user profile; as such, it is composed in sequence after the profiling service. Services must be performed in the same sequence that they are defined. The notation we use for

this directive is  $s_0$  **seq**  $s_1$ .

### Parallel

The *parallel* directive depicts the composition of services to be executed concurrently. In our scenario (Section 4.2), depending on the current context and user's preferences, this is the case of the content and the advertising services, as they may be invoked at the same time. Its notation is  $s_0$  **parallel**  $s_1$ .

### Any Order

The *anyorder* directive is used to describe the composition of one or more services that must be run in sequence, but do not have to be executed in any specific order. In the scenario (Section 4.2), depending on the current context and user's preferences, this is the case for instance of the content and the advertising services, as they are independent from each other and can be executed one after the other regardless of the actual order. The notation used for this composition semantic is  $s_0$  **anyorder**  $s_1$ .

### Signal

The *signal* directive enables to define a synchronization point for a set of services that cannot be launched (set  $S_{out}$ ) till they have received a completion signal from all the services they are synchronized with (set  $S_{in}$ ). This is the case for instance in our scenario (Section 4.2) of the advertising and the logging/reporting services, where the advertising service fires the logging/reporting one. For the signal directive, the control returns once all synchronized services have been launched. We use the notation  $S_{in}$ **signal** $S_{out}$  (e.g.,  $s_1$ **signal** $s_2$ ) in the case where both  $S_{out}$  and  $S_{in}$  contain one element, or more generally  $\{s_0, s_1\}$ **signal** $\{s_2, s_3, s_4\}$ .

### Choice In

The *choice in* directive depicts the composition of a set of services among which just one should be performed and its nature depends on a guard condition. This is the case of the strategy for composing the content and the advertising service, in our scenario. We use the following notation for a choice semantic: **choice**  $\langle guard\ condition \rangle$  **in**( $value : s_1, value : s_2, value : s_{null}$ ); where  $s_{null}$  is a *null* service.

### Any Choice In

The *anychoice in* directive depicts the case where only one, among a set of services, should be executed and this service can be anyone in the set. This is the case of the content service, in our scenario, which is further decomposed as the execution of any of the audio or video services available. We use the following representation for an any choice composition: **anychoice in**( $s_1, s_2$ ).

### Loop

The *loop* directive expresses the case where a service is executed repeatedly depending on the value of a guard. In the scenario, this is the case of the Smart Media Player that updates the list of the next-to-come songs or videos iteratively. We use the following notation for it: **loop**  $\langle$  *iterator*, *guard condition*  $\rangle$  ( $s_0$ ).

Figure 4.3 illustrates an example of a moSS Service Entity. As shown, service  $S$  has a valid service type within the taxonomy used ( $tID$ ), and it can thus be delivered as a single service; however,  $S$  can also be decomposed as  $S = S1$  **seq** ( $S2$  **parallel**  $S3$ ), where  $S3$  can be further decomposed as  $S3 = S4$  **seq**  $S5$  **seq**  $S6$ . Note that  $S2$  **parallel**  $S3$  does not correspond to any service type instead ( $taxonomyID = null$  and  $serviceID = null$ ), and thus it can only be delivered as a composition of services.

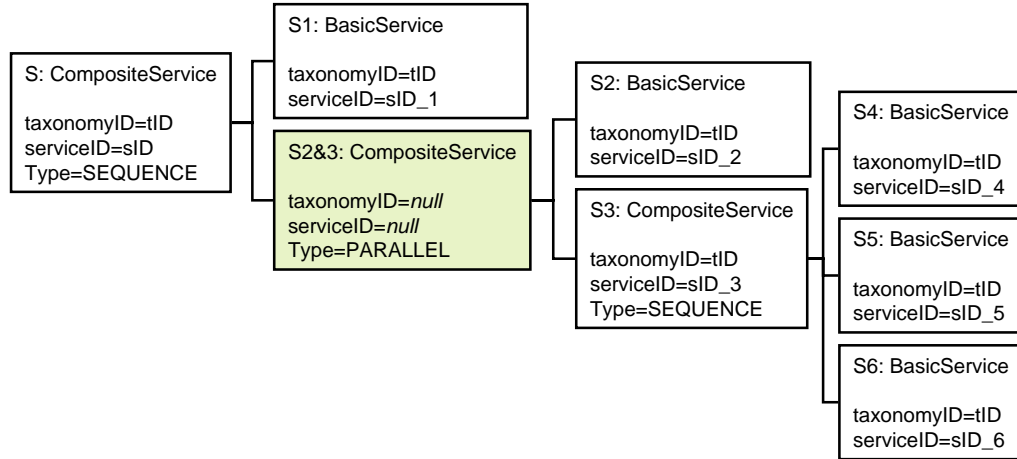


Figure 4.3: Example of a MoSCA Composite Service

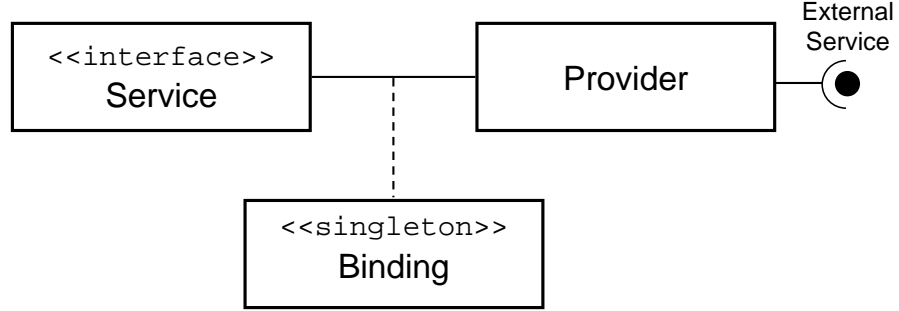


Figure 4.4: Binding Entity

### 4.3.2 The Binding Entity

A Service Entity is a static element. In order for a service to be executed, at least one service provider, delivering the functionalities of such service, must be available at service request time. When a request for service type  $S$  is received, a *Binding* to the most reliable provider of  $S$  (or set of providers collectively delivering composite service  $S$ ) currently available in the surrounding is created (Figure 4.4). Note that Service Entities and Providers do not have direct access to each other; rather, access is provided by means of the Binding Entity, thus facilitating dynamic reactions to changes in the environment (e.g., updating the set of composing service instances delivering a composite service  $S$  at execution time impacts the Binding Entity only). Note also that Provider entities are effectively wrappers for services external to the framework, and supplied by devices in the surrounding.

Let us revisit the previous example (Figure 4.3) of a service  $S$  that can be decomposed as  $S = S1 \text{ seq } (S2 \text{ parallel } S3)$ , with  $S3$  that can either exist alone or be further decomposed as  $S3 = S4 \text{ seq } S5 \text{ seq } S6$ . As exemplified in Figure 4.5, depending on the providers available in the surrounding (and on their predicted reliability), the executed composition may vary: in the example on the left,  $S$  is delivered as a single service by provider  $P2$ , while in the example on the right it is delivered by combining the services provided by  $P4$ ,  $P1$ , and  $P5$ .

## 4.4 Concrete model

To realise moSS conceptual model, three main components have been developed: the Mobility Predictor, the Semantic Reasoner and the Adaptive Binder. The former forecasts the co-location time between two devices at any specific time; the second examines the structure of a composition to determine co-location requirements for each component: whilst the latter observes alterations to the set of chosen providers and services available

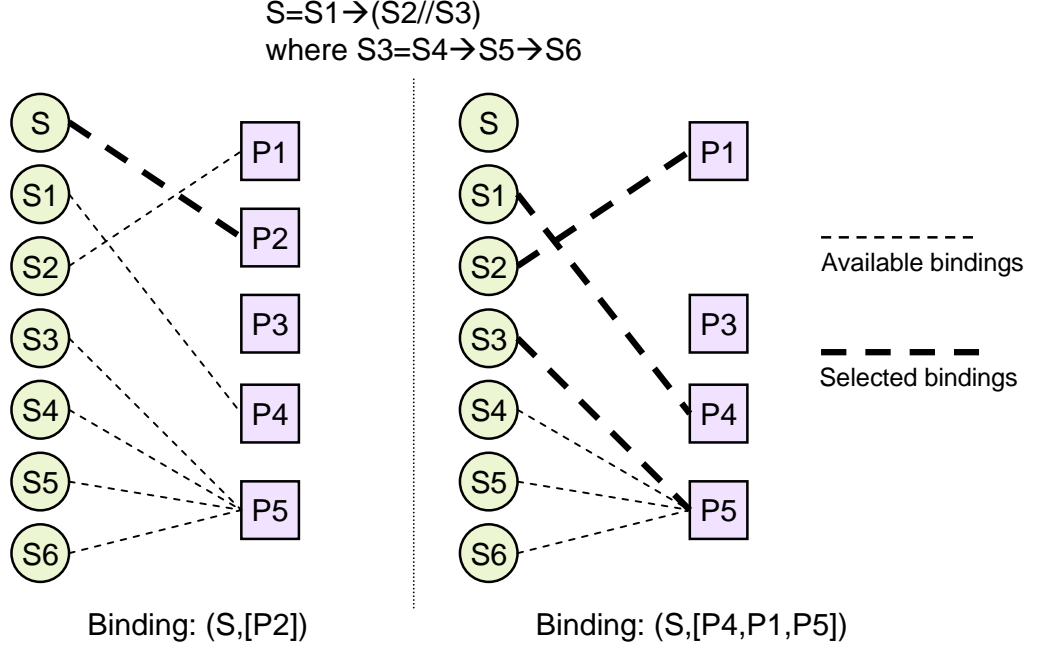


Figure 4.5: Example of MoSCA Bindings

in the surroundings to adjust service bindings, if required. The details on each method are presented next.

#### 4.4.1 Mobility Predictor

The main goal of the Mobility Predictor is to estimate the time that a provider will remain directly connected (i.e., within single hop distance) to the device that initiated the request. This time is used to approximate the probability that such provider will indeed be available to participate in the composition (i.e., we make the assumption that if a service provider is busy serving others clients, it will not send a beacon to notify its presence). The basic observation underpinning the Mobility Predictor is that people exhibit regular behavioural patterns in their daily activities (Observation 1). Based on this observation, we have looked at highlighting behavioural patterns in terms of weekdays and hours; this is because humans organize themselves around these notions, as the structure of agendas and timetables witnesses (e.g., John attends a digital photography class every Wednesday from 7:00 pm to 9:00 pm; Camilla takes the train to London everyday Monday to Friday at 8:05 am, etc.). Based on this observation, we have defined a simple, yet effective, prediction mechanism that aims at learning human behavioural patterns from past activities; for every day of the week  $d$ , and for every hour  $h$  within a day, a device  $i$  logs the duration of its encounters with any another device  $j$ . We use the symbol  $\delta_{i,j}(d, h)$  to refer to the historical co-locations between devices  $i$  and  $j$  in the specific time slot  $(d, h)$ . Recording

co-location statistics incurs a small amount of storage and processor usage; this amount scales linearly with each additional host that is tracked. To reduce this overhead, only records about *familiar strangers*, that is, hosts we have been encountering with at least a certain frequency, can be kept.

Given two service instances  $i$  and  $j$ , and the current time  $t$  which falls in slot  $(d, h)$ , the predicted duration of co-location is then computed as follows:

$$\sigma_{i,j}(t) = \begin{cases} \min(\delta_{i,j}(d, h)) & \text{if } \text{avg}(\delta_{i,j}(d, h)) \leq \alpha \cdot \text{stddev}(\delta_{i,j}(d, h)) \\ \text{avg}(\delta_{i,j}(d, h)) - \alpha \cdot \text{stddev}(\delta_{i,j}(d, h)) & \text{otherwise} \end{cases} \quad (4.1)$$

Intuitively, the higher the variation in past co-location durations, the lower is the predicted co-location time (i.e., the lower the confidence in how long the current co-location will last). In general, we prefer to underestimate the duration of co-location, rather than overestimating it, to later minimise the number of initiated but then failed compositions; we thus always set parameter  $\alpha$  to be a positive constant value, whose impact on service completion rate will be analysed experimentally in Chapter 5.

#### 4.4.2 Semantic Reasoner

In order to deliver a compound service  $S$ , moSS could use the collocation predictions computed by the Mobility Predictor to decide what service providers to bind to (within the current environment), so to have high probability that the composition will successfully complete (i.e., that no component will become unreachable during service delivery). For example, if  $n$  component services are needed to deliver  $S$ , and if it is estimated (e.g., from past experiences, from Service Descriptors, etc.) that it takes up to  $\Delta t$  seconds for  $S$  to complete, then the  $n$  providers  $P_1, \dots, P_n$  would be chosen so that each of them is estimated to remain co-located with the client's device at least  $\Delta t$  from the time the service request begins. In other words, we could require *each* component service to remain available for the *whole* duration of the composition.

This requirement may become quite stringent in highly dynamic environments, especially for compositions requiring the participation of many components, resulting in many service deliveries not even being attempted.

However, not all services are indeed needed for the whole duration of the composition (observation 2). For example, if two services  $S_1$  and  $S_2$  are *sequentially* composed (i.e.,  $S_1 \text{ seq } S_2$ ), and it is estimated that each will take 30 seconds, then  $S_2$  will be needed for the whole duration of the composition (i.e., 60 seconds) while  $S_1$  will be needed for the first 30

seconds only. The semantics of the composition quite precisely identify *when* and *for how long* a specific service instance is going to be needed. Based on co-location predictions, the *Semantic Reasoner* component thus leverages the specific composition semantics in use to determine the *minimum co-location requirement* for each service instance  $S_i$  (and corresponding provider  $P_i$ ) individually as follow.

### Sequence

In the case of  $n$  services composed in *sequence*  $S_1 \text{ seq } \dots \text{ seq } S_n$ , and assuming each service  $S_i$  takes  $\Delta t_i$  seconds to execute, then the minimum co-location requirement for any provider of  $S_i$  is  $\Delta t_i^* = \Delta t_i + \sum_{j=0}^{j < \bar{i}} (\Delta t_j + \Delta t_{j,j+1})$ , where  $\Delta t_{j,j+1}$  is the maximum tolerated interval of time between the completion of  $S_j$  and the launch of  $S_{j+1}$ .

### Any Order

If services are to be executed sequentially one after the other but in *any order*, then the Semantic Reasoner favours the sequencing for which the services that are estimated to be co-located for shorter times are executed first. If we indicate with  $\bar{i}$  the position of  $S_i$  in this sequential order, then the minimum co-location requirement for any provider of  $S_i$  is  $\Delta t_i^* = \Delta t_i + \sum_{j=0}^{j < \bar{i}} (\Delta t_j + \Delta t_{j,j+1})$ .

### Parallel

If services are composed in *parallel*, then their minimum co-location requirement for any provider  $S_i$  is  $\Delta t_i^* = \Delta t_i$ .

### Choice In

The minimum co-location requirements for services composed according to the *choice in* composition semantic depend on whether the guard condition can be pre-computed or not. In the first case, being  $S_i$  the service that should be executed, then the following applies:  $\forall j \neq i \Delta t_j^* = 0$  and  $\Delta t_i^* = \Delta t_i$ ; otherwise  $\forall i, \Delta t_i^* = \Delta t_i$ .

### Any Choice In

In the case services are composed according to the *anychoice in* composition semantic, then the Semantic Reasoner favours the service with the most favourable estimated co-

location time compared to the time it takes to run it  $\Delta t$ . If  $S_i$  is such service, then the minimum co-location requirements are:  $\forall j \neq i \Delta t_j^* = 0$  and  $\Delta t_i^* = \Delta t_i$ .

### Loop

The minimum co-location requirement for service  $S$  that has to be iteratively executed  $n$  times is computed as the sequential composition of  $n$  instances of  $S$ , that is:  $\Delta t^* = \Delta t + \sum_{j=0}^{j < n} (\Delta t_j + \Delta t_{j,j+1})$ .

### Signal

The minimum co-location requirements for  $S_{in}\mathbf{signal}S_{out}$  are computed as  $S_{in}\mathbf{seq}S_{out}$ . In case  $S_{in}$  consists of more than one service, they are then treated as if they were composed in parallel (and the same for each service in  $S_{out}$ ). That is,  $\forall S_i \in S_{in}, \Delta t_i^* = \Delta t_i$  and  $\forall S_j \in S_{out}, \Delta t_j^* = \Delta t_i + \Delta t_j + \Delta t_{i,j}$ .

The minimum co-location requirements for compositions that use more than a directive (e.g.,  $S1 \mathbf{seq} (S2 \mathbf{parallel} S3)$ ) are computed by breadth-first traversing each level of the composition structure (inferred from the same Service Entity).

Based on the minimum co-location requirements of each component service, the Semantic Reasoner on the client device  $C$  decides whether to launch a composition, and (if so) on what instances to rely on, using Algorithm 4.

To begin with, providers of services needed in the composition are found in the environment (step 1). A prediction of the *remaining* co-location between each of these providers and  $C$  is computed ( $\rho_{C,k}$ ); only those providers who are expected to remain available for the minimum co-location requirement are kept (step 2). If more than one provider is available for a given component  $S_i$ , then the one with the longest remaining predicted co-location time is selected. If, at the end of the process, set  $P$  contains one provider for each component  $S_i \in S$ , then the composition is attempted.

Note that the minimum co-location requirement could be set even looser than what we have described thus far. In Algorithm 4, we require providers of all component services to be available at the beginning of a composition, for the service to be started; however, depending on the composition semantics (e.g., sequential), some services may only be needed at a later stage. A composition could thus be started even if instance  $S_i$  is currently not available, provided that there is a high probability that  $S_i$  will become available by the time it is needed. Such probability could be computed based on: the maximum waiting

---

**Algorithm 4** Service Components Selection Algorithm
 

---

**Input Parameters**

- semantics de-composition of service  $S$  into  $\{S_1, \dots, S_n\}$ ;
- $\rho_{C,k}$  a prediction of the *remaining* co-location time between the client device  $C$  and provider  $P_k$ ;
- $\Delta t_i^*$  the interval of time  $S_i$  is requested to be available for, as estimated based on the composition semantics;

**Returns:** set  $P$  of service instances  $\{P_1, \dots, P_n\}$  to bind to, to deliver the composite service  $S$ .  $P = \emptyset$  if no stable composition can be attempted.

$P = \emptyset$

{Step (1) - Functional Matching}

$F = \emptyset$

**for all**  $P_k$  in the environment **do**

**if**  $S_k$  delivered by  $P_k$  belongs to  $S$  **then**

$F = F \cup \{P_k\}$

**end if**

**end for**

{Step (2) - Stability Filtering}

$T = \emptyset$

**for all**  $S_i \in S$  **do**

**if**  $\exists P_k \in F \mid (S_k \equiv S_i) \wedge (\rho_{C,k} \geq \Delta t_k^*)$  **then**

$T = T \cup \{P_k\}$

**end if**

**end for**

{Step (3) - Stability Maximisation}

**for all**  $S_i \in S$  **do**

$P_k = \max_{\rho_{C,j}} \{P_j \in T \mid P_j \text{ provides service } S_i \in S\}$

$P = P \cup \{P_k\}$

**end for**

return  $P$

---

time by which  $S_i$  will be needed (e.g., for services composed sequentially, the *time before*  $S_i$  is needed is  $\sum_{j=0}^{j<i} \Delta t_j + \Delta t_{j,j+1}$ ); and historical/contextual information about what services were available at a given time (and possibly place) in the past. If historical information were not available, an estimate could be obtained by looking at the dynamicity of the mobile environment (the average growth per second of the number of services available, as perceived by the device managing the composition), and the homogeneity of the available services.

#### 4.4.3 Adaptive Binder

No matter how sophisticated the mobility predictor and semantic reasoner can be, there are intrinsic limits to the estimated reliability of a service composition as, despite following regular patterns most of the time, humans do behave unpredictably from time to time (Observation 3). The Adaptive Binder component is used to improve the reliability of moSS, enabling dynamic re-binding of those services whose elected providers have disappeared during execution.

When a request is issued to moSS to find and elect providers for a composite service, the Adaptive Binder is activated and asked to keep record (within the Binding Entity) of *all* the providers available in the surroundings that deliver any of the services making up a composition currently under request. For as long as the composite service is being executed, the Adaptive Binder updates these sets, based on providers (dis)appearances. If a component service  $S_i$  has to be executed but the elected provider  $P_i$  has since disappeared (i.e., the Binding Entity  $\langle S_i, P_i \rangle$  is no longer valid), moSS inspects the list of providers of  $S_i$  currently available (as maintained by the Adaptive Binder), sorts the list in decreasing order of remaining co-location time, and finally selects a new provider  $P_{i'}$  to bind to, if available, as the one providing the highest estimated reliability at this point in time. The Binder component is useful also in environments where co-location cannot be estimated (e.g., unfamiliar environments where no familiar strangers can be found to provide the services needed): in such cases, rather than refusing compositions on the ground of unavailable co-location information, the framework can be configured to attempt compositions anyway, relying on the capability of the Binder to quickly react to providers' disappearance.

#### Predicting changes to the Environment

Whenever moSS runs on a device that supports multi-threading efficiently and returns the signal strength for devices in the surroundings, the Adaptive Binder can be configured to act pro-actively (rather than reactively) and *predict* changes in the environment so that re-bindings are initiated at an earlier stage.

To do so, the Adaptive Binder builds on a the simple observation that a provider is available to a mobile consumer as far as they can establish a connection whose signal is adequate enough for communication (i.e., the signal strength of the provider, as perceived by the consumer, is greater than a minimum value). Also, we remark that, as two devices move, the signal strength between them will change and an higher increase (or decrease) of signal strength will likely correspond to higher relative speed.

Based on these observations, we have defined a simple yet effective prediction mechanism that aims at forecasting a device departing from the surrounding of another device. This method builds on the concept of connection lifespan, which is a measure (in seconds) of the time required to loose a connection given its current signal variation speed (i.e., the signal decrease that would cause the loss of connectivity, divided by the signal variation speed). More in detail, if we define  $\omega_{P_i, C_j}$  as the intensity of the signal of the provider  $P_i$  as perceived by a consumer  $C_j$ ,  $\sigma_{P_i, C_j}$  as the lifespan of such connection and  $\omega_{min}^{tech}$  as the minimum signal strength for the wireless technology in use, then:

$$\sigma_{P_i, C_j}(t) = \begin{cases} 0 & \text{if } \omega_{P_i, C_j}(t) \leq \omega_{min} \\ 1 & \text{if } \frac{d\omega_{P_i, C_j}(t)}{dt} = 0 \text{ and } \omega_{P_i, C_j}(t) > \omega_{min} \\ \frac{\omega_{P_i, C_j}(t) - \omega_{min}}{\lceil |(d\omega_{P_i, C_j}(t)/dt)_{mW/sec}| \rceil} & \text{otherwise} \end{cases} \quad (4.2)$$

In practice, we approximate  $d\omega/dt$  with the standard deviation and the average of its sampling variation, i.e.:

$$d\omega/dt = \text{avg}(\Delta\omega/\Delta t) + \alpha \text{stddev}(\Delta\omega/\Delta t) \quad (4.3)$$

This is because we are interested in the worst case (i.e., estimate the speed variation with an equal or greater speed). Intuitively, from a theoretic point of view, the higher is  $\alpha$ , the greater is the percentage of cases for which the actual signal variation speed is equal or lower to the one predicted and, as a consequence, the estimated departing time is equal or lower to the actual one. This model does not consider whether the signal strength is increasing or de-creasing but just its speed. This is to also cater for the case of users traversing open spaces. In this case, in fact, no real assumption can be made on the continuity of their walking direction.

It is also worth noticing that this notion of connection lifespan is technology agnostic and does not require the provider nor the consumer device to be enriched with positioning technologies, thus enabling to target a wider set of machines compared to models that require the location and the speed of such devices to be known. Whilst algorithms that use signal strength offer poor accuracy in computing device position, here the intent is not to co-locate devices but to estimate their reciprocal accessibility; the signal strength is a natural measure of it, as two machines may well be co-located but they are effectively

accessible to each other only if their connectivity is good enough.

## 4.5 Summary

In this chapter we have presented moSS, a service discovery framework that enables the rapid development and deployment of reliable composite services and applications. moSS provides end users, as well as application engineers, the abstraction of a moSS Service as a single, locally available service. moSS transparently binds a Service to the set of available providers that are capable of collectively delivering the composite service with the highest reliability. It does so by reasoning about the composition semantics and the dynamically learned co-location patterns with other providers. Unforeseen changes to such patterns are being monitored and possibly predicted during service execution, thus triggering re-bindings if necessary. In the next chapter, we present a thorough evaluation of the reliability achieved by moSS, using real mobility traces.

## Chapter 5

# Evaluation of Reliable Discovery

This chapter presents the evaluation of our Reliable Discovery system, moSS. As moSS actually consists of three distinctive yet complementary methods, we introduce their evaluation in turn. First, we discuss the Mobility Predictor in Section 5.1. Hence, we focus on assessing the Semantic Reasoner in Section 5.2, to then conclude examining the Adaptive Binder in Section 5.3. For each of them, the results of both an experimental and an analytical evaluation are presented, the former quantifying their performance, the latter quantifying the resource overhead they entail.

### 5.1 Mobility Predictor

#### 5.1.1 Performance

In this section, we discuss the performance evaluation of the Mobility Predictor; first, we depict the methodology applied, the datasets used and the metrics adopted before presenting the actual experiments setup and their results.

##### Methodology

In order to evaluate the accuracy of the Mobility Predictor, we have analysed a set of real co-location traces for a real life scenario, and we have compared co-location estimates against the actual co-location intervals. This set of traces contains start and end times of co-locations. We have ordered these logs from the oldest to the most recent, fed them gradually to the Mobility Predictor as *training set*, and computed co-location estimates from them. Note that, in pervasive deployments, each user has just a limited knowledge of the habits of other people, based on their encounters thus far. Such knowledge (i.e.,

the training set) grows over time, as new encounters are made. So, when assessing the Mobility Predictor evaluation, we aim to study the impact that recurrent encounters have on the qualities of the co-location estimates. To do so, whenever two devices bump into each other, we feed to the Mobility Predictor their encounters history as observed up to that instant as *training set* and we predict their co-location time for that encounter. We then compare the estimates against the actual values.

## Metrics

The main intent of the Mobility Predictor is to provide reliable estimates on the duration of an encounter so that we can gauge whether to start a service or not. Intuitively, if the Mobility Predictor underestimates such time, the risk is to not start a service that could indeed be completed successfully; alternatively, if the Mobility Predictor overestimates such time, the risk is to initiate a service that cannot be completed. In the second case, an error in a prediction will translate in a bad user experience, as the end user will be well aware of the problem. For this reason, we intentionally look at underestimating co-location times rather than over-assessing. More precisely, we compute as “correct predictions” the estimates whose value is lower or equal to the actual encounter duration, while we record a failure in the case the predicted co-location is greater than the actual one.

On the other hand, we remark that, although a more conservative prediction corresponds to a greater probability for it to be right (i.e. the actual co-location time is greater or equal to the estimated time), it also true that a more conservative estimate has less value (as it causes missed opportunities).

In order to cater for both these aspects we adopt two metrics when assessing the Mobility Predictor:

- the ratio of correct predictions, i.e.:

$$\frac{|\text{correct predictions}|}{|\text{predictions}|} \quad (5.1)$$

- the percentage of underestimation (for each successful prediction), defined as:

$$\frac{\text{actual colocation time} - \text{estimated colocation time}}{\text{actual colocation time}}. \quad (5.2)$$

## Datasets

In order to evaluate the Mobility Predictor, we need real encounter traces over a period long enough to exhibit human temporal behavioural patterns. We have used the connection

logs from real life Bluetooth devices, available from the CRAWDAD [Kotz and Henderson, 2005] resource archive. In particular, we have elected the MIT Reality Mining dataset [mit, 2005] as our reference scenario. This dataset contains the Bluetooth ID of devices in proximity, as detected by one hundred Nokia 6600 phones, which were given to MIT staff and students, over a period of nine months over the course of the 2004-2005 academic year. Each phone had been configured to scan the environment and log co-located devices (i.e., within one hop distance). The final dataset contains in excess of 500,000 hours of data about human activity. The reasons we focus on this dataset are many: to begin with, unlike many real data sets, this one is big enough to enable accurate evaluation of our model, without having to rely on synthetic (unreal) mobility traces. Second, it has been collected in what we consider a typical setting for the consumption of composite services (i.e., a university campus, with some services being available centrally, others from devices embedded in buildings, and others still from peer Bluetooth devices). Finally, the dataset spans a long enough period of time for temporal behavioral patterns to emerge, and for our model to learn and exploit them.

## Benchmarks

To evaluate the Mobility Predictor we have adopted as reference benchmark “God’s view”. In other words, we have compared our algorithm to an ideal and perfect prediction method that estimates co-location time with their exact actual encounter duration (oracle).

## Setup

In order to assess the accuracy of the Mobility Predictor component, independent of any service composition, we have conducted the following experiment. Whenever two devices  $i$  and  $j$  entered within connectivity range, and thus an event was logged within the encounter dataset (i.e., MIT Reality Mining dataset), we have predicted  $\sigma_{i,j}(t)$ , that is, for how long device  $i$  expects to remain co-located with device  $j$ , based on the co-location statistics observed up until time  $t$  (see formula 4.1 in Section 4.4.1). We have then compared our estimated co-location duration with the actual one and, in case our estimate was lower or equal to the actual one, we recorded a success, otherwise we recorded a failure. Note that, in our framework, it is important not to overestimate co-location durations, to minimise the risk of started but uncompleted compositions due to service/device departure. Nonetheless, an overly conservative model that excessively underestimate co-locations would inhibit services that would have been successful otherwise. For each successful (under)estimation, we have thus quantified the actual percentage of underestimation. Results about this set of experiments are reported in the next section.

## Results

The prediction success rates for the Mobility Prediction component, along with the observed average co-location times, are provided in the Table 5.1 for different values of  $\alpha$  in Equation 4.1 (i.e., 0.5, 1.0, 1.5 and 2.0). Obviously from a theoretic point of view, the higher is  $\alpha$  the greater is the percentage of cases for which the actual prediction is equal or lower to the actual co-location time. The empirical findings confirm the theory as highlighted below.

$\alpha$	0.5	1	1.5	2
Prediction success	66.86	89.38	92.73	94.24
Actual avg in secs	653.72	587.64	521.55	455.46

Table 5.1: Mobility Prediction. Success Rates

Figure 5.1 illustrates the percentage of correct predictions (i.e., predictions which are equal or lower to the actual colocation time) made by the Mobility Predictor, for different values of  $\alpha$ , and broken down for different levels of *familiarity* between each pair of devices. Given an observing (client) device  $i$ , we define **familiarity level** of a device  $j$  for weekday  $d$  and hour  $h$  as the percentage of occurrences that  $i$  was connected to  $j$  on  $d$  days and at  $h$  hour, as observed prior to the current time. The expectation is that, the more familiar a device is, the more accurate the prediction becomes. As expected, the higher the value of  $\alpha$ , the more conservative the model becomes, with the percentage of correct predictions reaching 95% for  $\alpha = 2$ . However, the price to pay is loss of accuracy. Figure 5.2 illustrates the amount of underestimation, computed as per formula 5.2, for the same values of  $\alpha$ : as shown, the prediction can miss up to 50% of the actual colocation duration when  $\alpha = 2$ . A good balance between correct predictions and underestimation can be achieved for  $\alpha = 1$ , where the former reaches roughly 90%, while the latter varies between 22% for strangers, down to 12% for familiar devices. In the following experiments, we have thus set the value of  $\alpha$  to 1.

It is interesting to note the impact of familiarity level on the behaviour of the Mobility Predictor: familiarity plays little importance in the percentage of correct predictions (Figure 5.1), unless the standard deviation is only marginally considered ( $\alpha = 0.5$ ), thus revealing a predictor model that tends toward the cautious side; however, it does play an important part in reducing the amount of underestimation, which neatly decreases the more frequently the pair of devices have met (Figure 5.2).

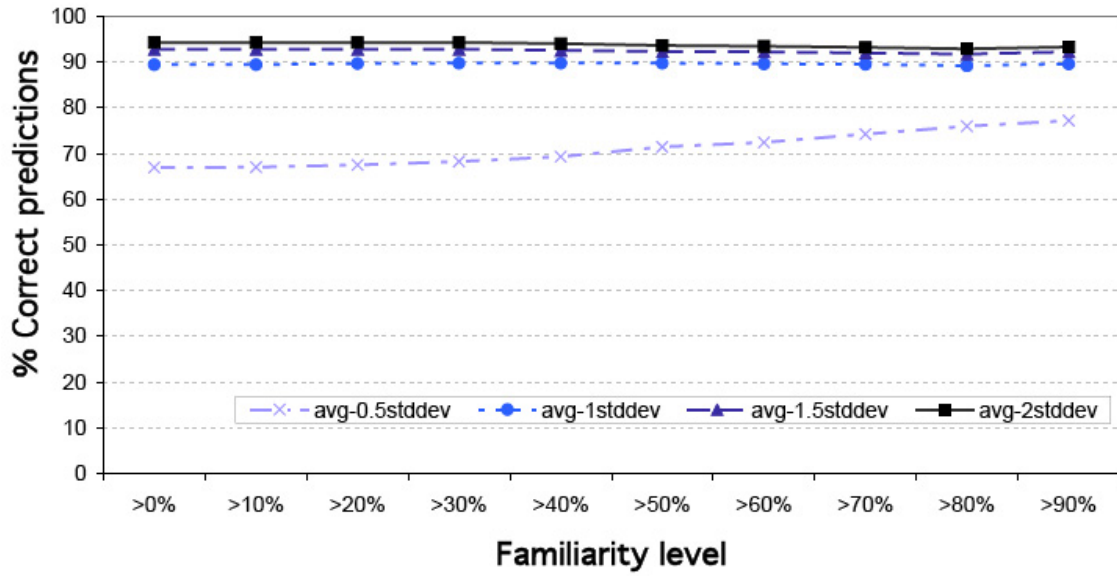


Figure 5.1: Mobility Predictor - Correct Predictions

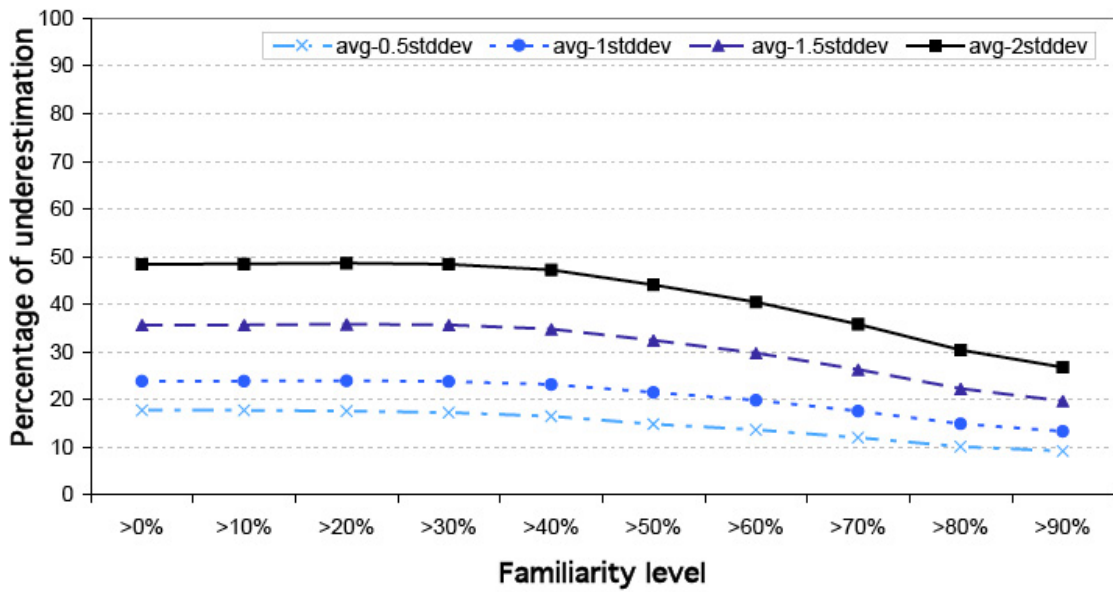


Figure 5.2: Mobility Predictor - Underestimation

### 5.1.2 Overhead

In the following we assess the overhead of the Mobility Predictor by quantifying analytically its memory allocation and processing cost.

$\alpha$	0.5	1	1.5	2
> 0%	66.86	89.38	92.73	94.24
> 10%	66.93	89.41	92.74	94.25
> 20%	67.49	89.61	92.77	94.27
> 30%	68.21	89.70	92.73	94.23
> 40%	69.28	89.71	92.55	94.00
> 50%	71.38	89.77	92.30	93.63
> 60%	72.38	89.52	92.16	93.45
> 70%	74.23	89.46	91.96	93.20
> 80%	75.89	89.18	91.70	92.88
> 90%	77.19	89.60	92.17	93.28

Table 5.2: Mobility Prediction. Success Rates by minimum familiarity level

$\alpha$	0.5	1	1.5	2
> 0%	17.75	23.85	35.63	48.38
> 10%	17.73	23.87	35.68	48.45
> 20%	17.58	23.93	35.84	48.66
> 30%	17.26	23.78	35.66	48.41
> 40%	16.51	23.18	34.80	47.23
> 50%	14.84	21.55	32.45	44.08
> 60%	13.66	19.84	29.77	40.45
> 70%	12.00	17.61	26.35	35.79
> 80%	10.16	14.96	22.31	30.37
> 90%	9.22	13.32	19.73	26.80

Table 5.3: Mobility Prediction. Underestimation Rates by minimum familiarity level

### Primary Memory Allocation

The Mobility Predictor estimates co-location times by pondering the historical data of previous encounters; more precisely, it considers their average and standard deviation as per equation 4.1 (repeated here for convenience), where  $\delta_{i,j}(d, h)$  refers to the historical co-locations between devices  $i$  and  $j$  on weekday  $d$  at hour  $h$ .

$$\sigma_{i,j}(t) = \begin{cases} \min(\delta_{i,j}(d, h)) & \text{if } \text{avg}(\delta_{i,j}(d, h)) \leq \alpha \cdot \text{stddev}(\delta_{i,j}(d, h)) \\ \text{avg}(\delta_{i,j}(d, h)) - \alpha \cdot \text{stddev}(\delta_{i,j}(d, h)) & \text{otherwise} \end{cases} \quad (5.3)$$

Rather than storing all encounter times and computing the  $avg(\delta_{i,j}(d, h))$  and  $stddev(\delta_{i,j}(d, h))$  values from scratch at every encounter, we observe that the average of  $N + 1$  numbers  $\bar{x}_{N+1}$  can be calculated from the average of  $N$  elements  $\bar{x}_N$  and the remaining  $N + 1$  element  $x_{N+1}$ . In fact:

$$\bar{x}_{N+1} = \frac{N\bar{x} + x_{N+1}}{N + 1}$$

Hence we can express  $\bar{x}_{N+1}$  as:

$$\bar{x}_{N+1} = \frac{\sum_{i=1}^N x_i + x_{N+1}}{N + 1} \quad (5.4)$$

Similarly, we remark that the standard deviation of  $N$  values can be expressed as:

$$s_N = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2} = \sqrt{\frac{1}{N} \sum_{i=1}^N x_i^2 - \bar{x}^2} \quad (5.5)$$

Hence, the standard deviation for  $N + 1$  values can be computed as:

$$s_{N+1} = \sqrt{\frac{1}{N + 1} \left( \sum_{i=1}^N x_i^2 + x_{N+1}^2 \right) - \bar{x}_{N+1}^2}$$

Overall, this implies that we can compute those values incrementally and that, for each device encountered on weekday  $d$  at hour  $h$ , we just need to store a tuple consisting of five values:

- the encountered device ID;
- $\sum_{i=1}^N x_i$  – the sum of previous co-location intervals;
- $\sum_{i=1}^N x_i^2$  – the sum of the squares of previous co-location intervals;
- $N$  – the number of times the device has been encountered on weekday  $d$
- $\min(x_i)$  – the minimum co-location time recorded (i.e.,  $\min(\delta_{i,j}(d, h))$ ) at hour  $h$ .

Note that not all tuples are needed at all times. In fact, at any instant  $t$ , just the historical data of encounters occurred in the weekday  $d_t$  and at hour  $h_t$  of the current instant  $t$  are needed. Therefore, while we persist all tuples in secondary memory (in files, one for each pair  $d, h$ ), we load in memory just tuples relating to the current weekday and hour. As a

result, being  $n$ , the number of ‘familiar stranger’ users, the primary memory overhead is  $O(n)$ . In practice, for the MIT Reality Mining dataset, the number of ‘familiar strangers’ collectively encountered for any give weekday  $d$  and hour  $h$  is always lower than 43 and, on average, less than 3.

## Processing

In terms of processing, there are two kinds of computations that are required: computations to capture historical data (*historical processing*) and computations to estimate the co-location times between pairs of devices (*prediction processing*). The first kind of calculations are performed for every encounter, whilst the second ones are executed exclusively for devices whose current co-location time needs to be predicted (i.e., just for devices matching a service request that are in the surroundings at the time a request is issued). Hence, *prediction processing* has a relatively limited impact on the overall processing cost.

First we analyse the overhead due to *historical processing*. As observed in Section 5.1.2, for each encounter, only the minimum co-location interval, the number of past encounters, the sums of co-location intervals and the sum of their squares need to be calculated. Also, these two sums can be computed incrementally:

$$\sum_{i=1}^{N+1} x_i = \sum_{i=1}^N x_i + x_{N+1} \quad (5.6)$$

$$\sum_{i=1}^{N+1} x_i^2 = \sum_{i=1}^N x_i^2 + x_{N+1}^2 \quad (5.7)$$

The first one (Equation 5.6) entails a sum at every encounter, whilst the second (Equation 5.7) involves a product and a sum. Therefore, the processing cost for each encounter is one sum (to compute the number of encounters  $N + 1$ ), plus two sums and a product. If, as in Chapter 3, we use the symbol  $\oplus$  to denote sums (or subtractions),  $\otimes$  for multiplications (or divisions) and  $\nabla$  for root squares, the historical processing cost per encounter is equal to  $3 \oplus + 1 \otimes$ . Although within the MIT Reality Mining there are 285,512 encounters recorded, the maximum number of encounters recorded in a day is 4072, meaning an average of less than 170 encounters per hour and hence, at maximum,  $510 \oplus + 170 \otimes$  calculations per hour.

As for computing the minimum co-location interval, we observe that this entail a comparison for each encounter and, hence, 170 comparisons per hour when we consider the MIT Reality Mining scenario.

In regards to the overhead due to *prediction processing*, we observe that, as per Equa-

tion 5.3, we need to workout the value of the average and standard deviation of historical encounters. Now, the average is simply:

$$\bar{x} = \frac{\sum_{i=1}^N x_i}{N}$$

where both  $\sum_{i=1}^N x_i$  and  $N$  are already known (from *historical processing*) and hence it simply implies  $1 \otimes$ . The standard deviation instead can be computed as per Equation 5.5. Where both  $\sum_{i=1}^N x_i^2$  and  $\bar{x}$  are already known (from *historical processing* and previous calculation of  $\bar{x}$ ) and hence, it simply implies  $1 \oplus + 2 \otimes + \nabla$ . To then predict the co-location time, the co-location Standard Deviation has to be multiplied for a factor  $\alpha$  and subtracted to the co-location Average (see Equation 4.1), leading to an additional sum ( $1 \oplus$ ) and multiplication ( $1 \otimes$ ).

Overall, the cost of predicting the co-location time between a pair of devices is about

$$2 \oplus + 4 \otimes + 1 \nabla$$

In MIT Reality Mining, no more than 15 devices are ever in the surroundings of another device; so, no more than  $30 \oplus + 60 \otimes + 15 \nabla$  additional computations are performed per a service request.

Overall, we remark that the overhead due to the Mobility Predictor is limited and scales linearly with the number of ‘familiar strangers’.

## 5.2 Semantic Reasoner

### 5.2.1 Performance

We now turn our attention to the performance evaluation of the Semantic Reasoner.

#### Methodology

In order to evaluate the effectiveness of the Semantic Reasoner, we have looked at the various composition semantics, each in isolation, and we have analysed the impact of semantic reasoning - over the Mobility Predictor - in terms of number of compositions initiated and successfully completed, compared to traditional discovery methods that select service providers regardless of the composition flow and their expected proximity availability.

As we did for the Mobility Predictor, we also analyse its conservative nature and examine negative misses (i.e., compositions misjudged as not likely to be carried out successfully). For both cases, we look at the impact of the size of the composition set (i.e., the number of items/services in the composition) and the maximum duration of each service.

### Metrics

As for the Mobility Predictor, the more conservative the Semantic Reasoner is in launching compositions, then the more accurate its predictions are when it comes to estimating if a composite service can be completed. On the other hand, a conservative may cause good opportunities to be missed. In order to cater for both aspects we adopt two metrics when assessing the Semantic Reasoner:

- the ratio of compositions started and successfully completed, here defined as

$$\frac{|\text{compositions started and successfully completed}|}{|\text{compositions started}|} \quad (5.8)$$

- the ratio of compositions not started that could have been successfully completed, defined as

$$\frac{|\text{compositions not started that would have been successfully completed}|}{|\text{compositions not started}|} \quad (5.9)$$

### Datasets

To assess the Semantic Reasoner we use the same dataset used for the Mobility predictor to simulate people/devices encounters. As for the service compositions, we simply define a set of  $n$  distinct service types, and then map each device in the mobility dataset to only one service type. We do it randomly and we repeat the same experiments for various mappings.

### Benchmarks

We evaluate the Semantic Reasoner against a traditional service discovery method and so we have adopted as reference benchmark a service selector that would elect service providers arbitrarily (as long as they are of the required service type).

### Setup

In order to evaluate the reliability achieved by the Semantic Reasoner component, we have randomly associated, to each of the devices within the mobility dataset (i.e., the 100 devices within the MIT Reality Mining dataset), one single service; the type of the service is chosen among  $n$  possible values, where  $n$  is a parameter indicating the number of services being composed. All experiments have been set so to require the composition of  $n$  distinct service types. In order to assess the quality of the selections made by the Semantic Reasoner when faced with a choice of providers, we have run a first set of experiments, where we have focused on situations where at least  $n \times 3$  devices were co-located; we have considered here sequential compositions only (which are those expected to gain the most from semantic reasoning), in a period of three months which exhibits high regularity in terms of co-location behavioural patterns. We have then extended our experiments so to cater for the whole 9 months worth of traces, considering all situations with at least  $n$  co-located devices (where compositions can be attempted - with or without choice of providers), and operating on all composition semantics. In both sets of experiments, we have recorded the percentage of successfully completed compositions, out of all those started, when using the Semantic Reasoner on top of the Mobility Predictor, and compared the results against a random selection of providers among those currently in reach (this is equivalent to a purely functional composition, with no reasoning about mobility). We have also quantified the number of compositions that we do not start but that would have been successfully completed. The experiments have been conducted on different values of  $\Delta t_i$  (i.e., average amount of time to execute a single service) and  $n$  (i.e., number of services within a composition). The obtained results are reported next.

### Results

Figure 5.3 refers to encounters extracted from the MIT Reality Mining dataset and reports the results for the first set of experiments assessing the reliability of the *choices* made by the Semantic Reasoner, while varying the number  $n$  of services that make up a composition and the duration of a component service  $\Delta t_i$  (maximum values of  $n$  and  $\Delta t_i$  where chosen so to test up to the limit cases as recorded in the available dataset). As mentioned already, these experiments focus on sequential compositions only, within a period of three months that present the highest degree of regularity in terms of co-location behavioral patterns.

As shown, the reliability of the composition performed by the Semantic Reasoner is consistently very high, in excess of 97% successful completion rate, regardless of the actual duration of the compound service. In fact, it remains almost constant while varying the number  $n$  of services being composed, and the time  $\Delta t_i$  required to execute each of them. On the contrary, the reliability of randomly made compositions is highly susceptible to increases of  $n$  and/or  $\Delta t_i$ : the achieved reliability is high only when just a couple of services

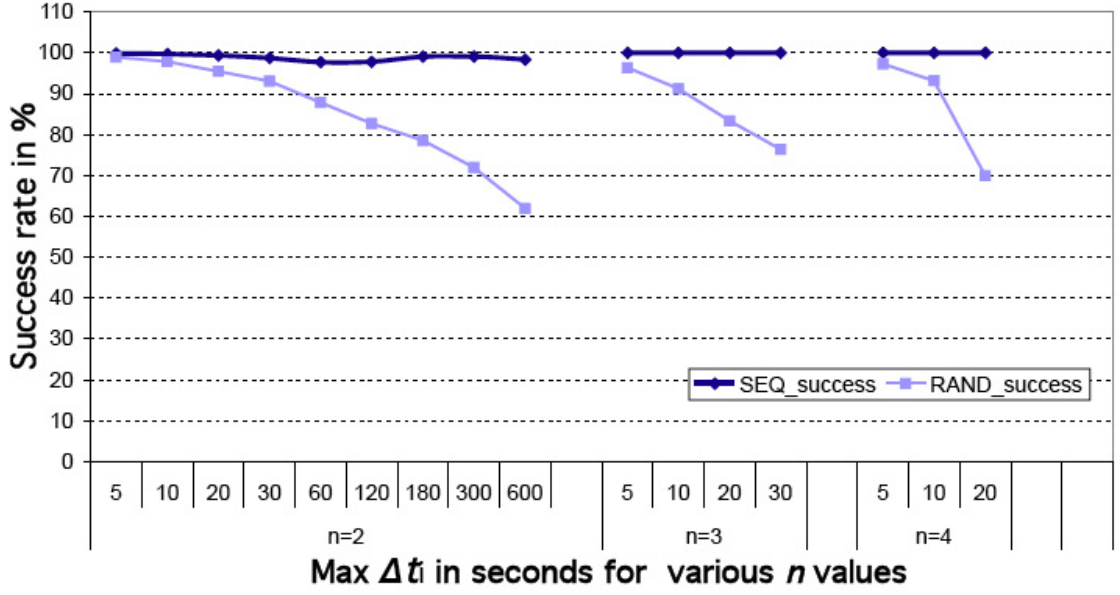


Figure 5.3: Semantic Reasoner - Successfully Completed Compositions

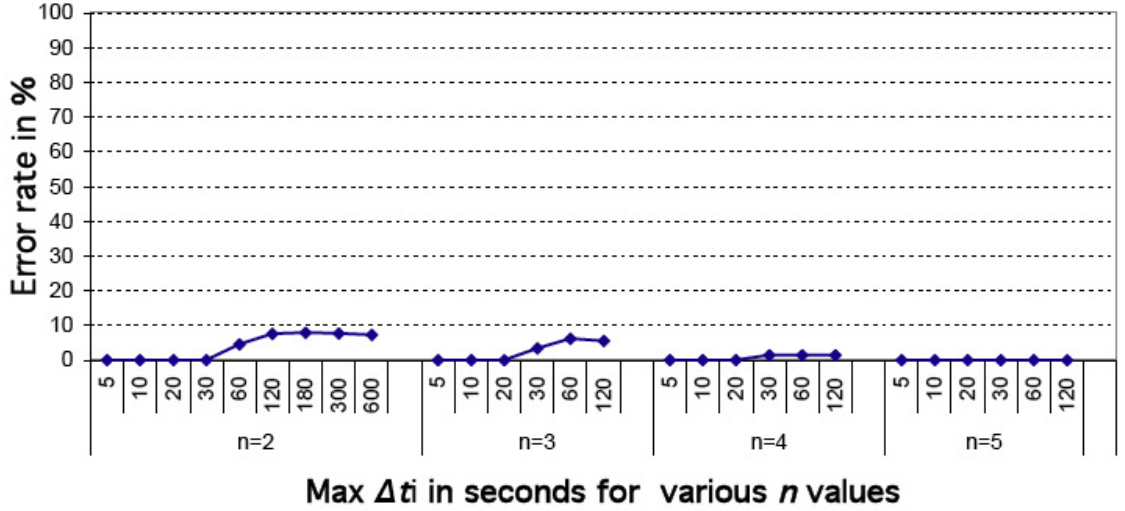


Figure 5.4: Semantic Reasoner - Missed Opportunities

that run for less than 10 seconds each are considered, as device mobility plays a smaller role in these situations; however, for compositions consisting of 3 services or more, and for service instances running for 20 seconds or longer each, the rate of successfully executed random compositions dramatically drops, with a decrease of up to 35% with respect to the reliability achieved by the Semantic Reasoner.

Under the same experiment setup, we have also quantified the number of compositions that the Semantic Reasoner chose not to start (as they were considered unlikely to complete),

but that would have been successfully completed instead. Figure 5.4 illustrates the results. As shown, at most 8% of the compositions were mistakenly not started, and these refer to the more risky cases where long running compositions were considered (i.e.,  $\Delta t_i$  in the order of minutes). Note also that the Semantic Reasoner does not attempt any composition involving 5 devices or more (no cases of successful compositions were reported in Figure 5.3 for  $n = 5$ ), and it correctly does so, as the error (miss) rate is exactly 0% in this case.

$n$	$\Delta t_i$	%Reasoner	%Random
2	5	99.8006	98.9118
2	10	99.7009	97.7798
2	20	99.4018	95.4547
2	30	98.7537	93.0204
2	60	97.7436	87.8065
2	120	97.8378	82.6581
2	180	99.1071	78.5455
2	300	99.1071	71.9505
2	600	98.3607	61.9029
3	5	100	96.294
3	10	100	91.2292
3	20	100	83.323
3	30	100	76.3434
4	5	100	97.2603
4	10	100	93.1507
4	20	100	69.863

Table 5.4: Percentage of successfully completed compositions among the ones started

The first set of experiments suggests that the Semantic Reasoner achieves high reliability, without being too conservative in initiating service compositions. Moreover, results are only marginally dependent on the duration of the compound service; this is in sharp contrast to the performance achieved by a random selection of service instances, where the successful completion rate dramatically drops as soon as service duration exceeds  $\Delta t_i = 20''$ . To confirm these results, we have conducted a second set of experiments that considered all composition semantics, and that covered the whole MIT Reality Mining data set life span (thus including months with low and very low degree of behavioural regularity). Compositions were attempted whenever  $n$  devices were co-located (for compositions of  $n$  services). The results are shown in Figure 5.5-5.8.

As all charts confirm, moSS achieves very high level of reliability across all composition semantics; the improvement over a random selection of services is more striking when increasing the number  $n$  of the composed services and their duration  $\Delta t$ , where peaks of 80% difference in reliability between the two approaches appear. In comparison to the experiments limited to the three most regular months within MIT Reality Mining dataset, the lower degree of regularity in people’s movement has an impact on the number of missed

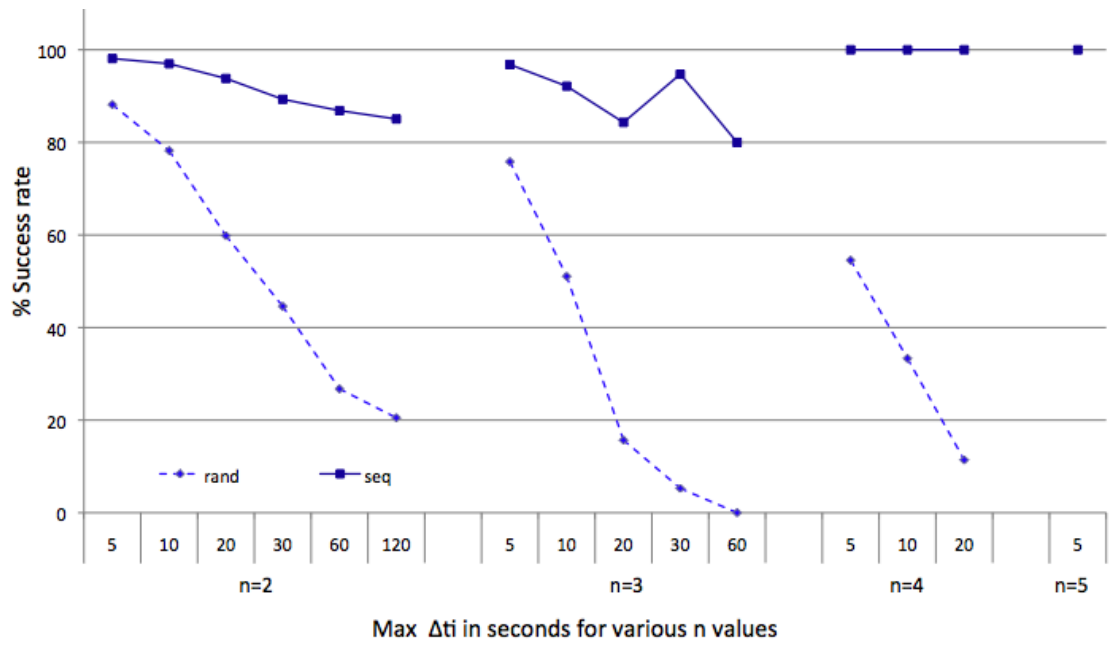


Figure 5.5: Semantic Reasoner - Success rates for Sequence Compositions

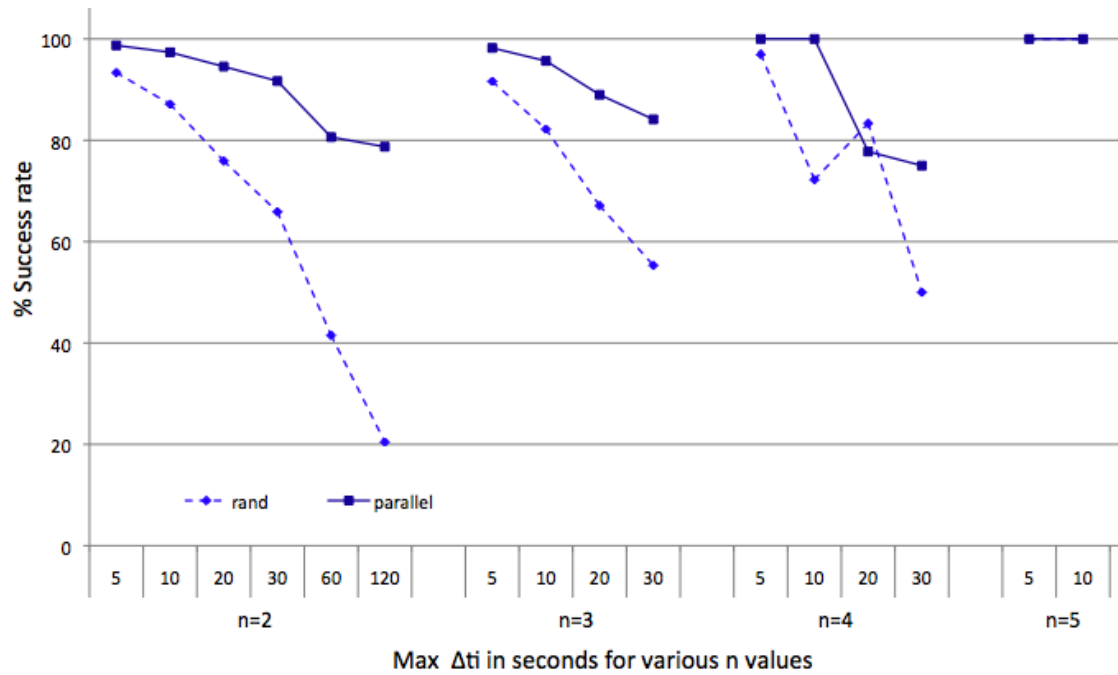


Figure 5.6: Semantic Reasoner - Success rates for Parallel Compositions

opportunities instead: in such case, co-location predictions are more conservative and less accurate, thus causing moSS to attempt fewer compositions than what would be possible

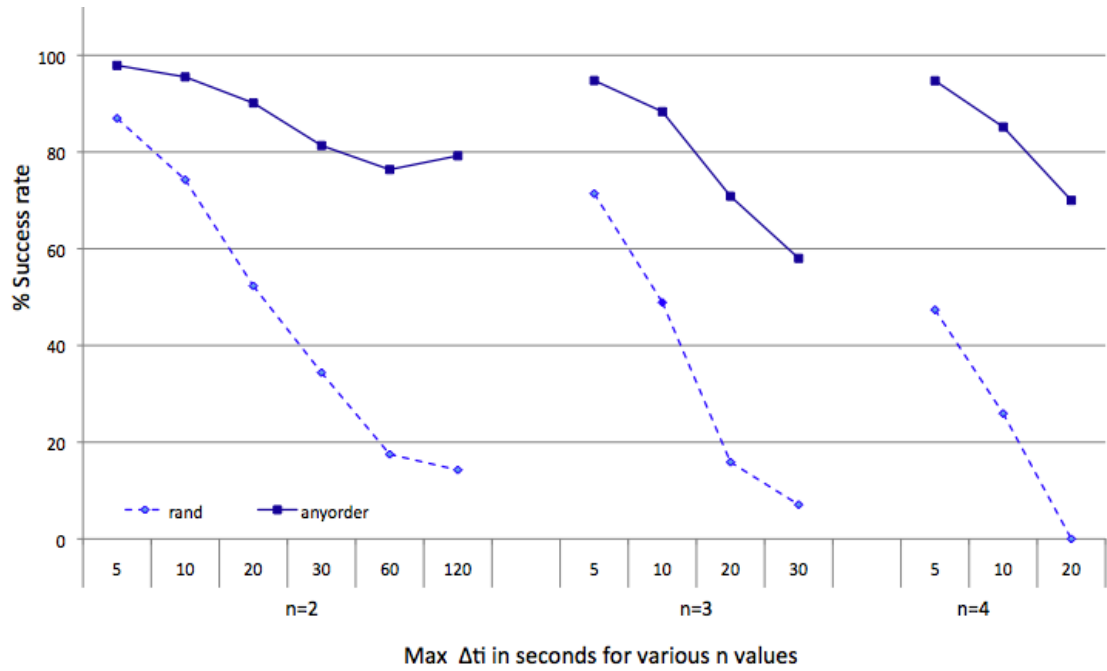


Figure 5.7: Semantic Reasoner - Success rates for Any Order Compositions

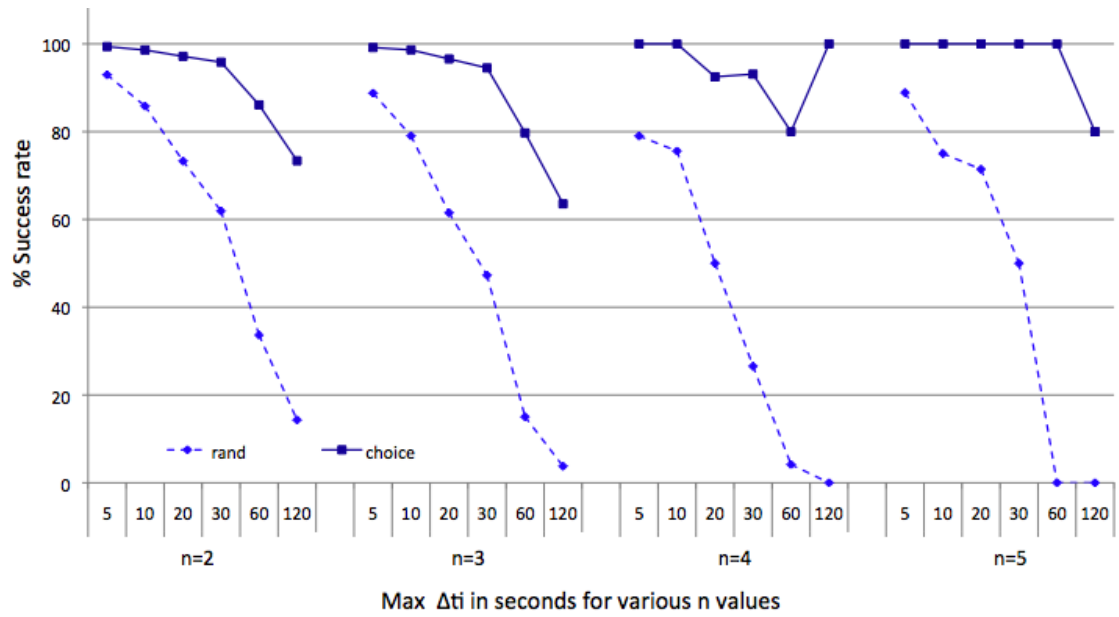


Figure 5.8: Semantic Reasoner - Success rates for Choice Compositions

(as Fig 5.9 illustrates). In order to cope with this kind of scenarios, the Adaptive Binder component has been introduced and its evaluation can be found in Section 5.3.

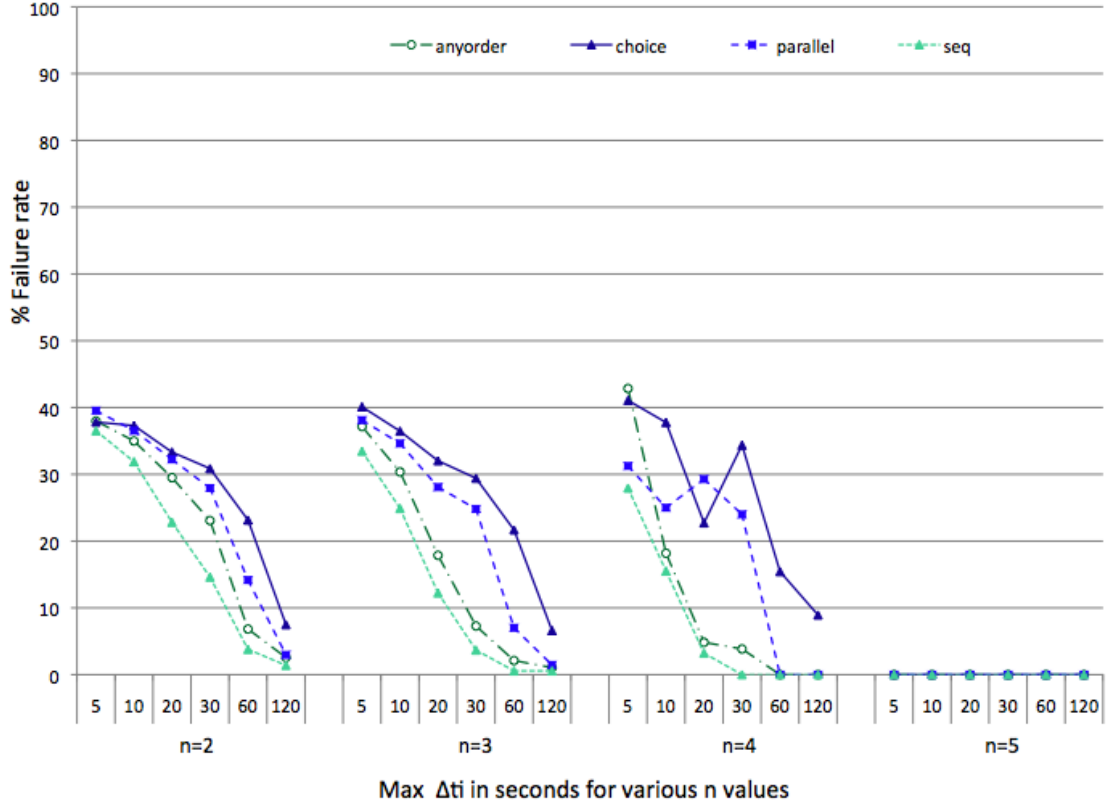


Figure 5.9: Semantic Reasoner - Missed Compositions

### 5.2.2 Overhead

In order to assess the overhead that the Semantic Reasoner entails, we quantify analytically its memory allocation and processing cost.

#### Primary Memory Allocation

By its own very nature, the Semantic Reasoner has a very limited footprint. In fact, it does not persist any additional data, other than that required for service analysis, and it only exploits information available at the time it is processed (i.e. the maximum time  $\Delta t_i$  it takes for each matching service  $S_i$  to complete). The memory allocation for such values is minimal. For instance, in the MIT Reality Mining experiments, there are never more than 16 co-located devices and therefore no more than 15 values are loaded in memory at any time.

## Processing

In terms of processing overhead, we observe that, among all composition semantics, the most expensive one is the *Any Order* composition. First, it requires to select, for each of the  $n$  service types of the composition, the provider that is predicted to be co-located the longest, thus implying a  $l - 1$  comparisons among  $l$  services of a same type and, overall, a computational complexity equal to  $O(m)$ , being  $m$  the number of services in the surroundings, matching any of the service types of the *Any Order* composition.

Once  $n$  services have been picked up, the *Any Order* semantic dictates to order them so to run first the services that are deemed to be co-located for the shortest time, thus implying further processing of computational complexity  $O(n \log n)$ .

Finally, for each  $S_i$ ,  $\Delta t_i^*$  (i.e.,  $\Delta t_i + \sum_{j=0}^{j < \bar{i}} (\Delta t_j + \Delta t_{j,j+1})$ ) need to be computed, entailing additional processing of complexity  $O(n^2)$ .

Consequently, the overall processing overhead is  $O(n) + O(n^2) + O(n \log n)$ , i.e.,  $O(n^2)$ .

## 5.3 Adaptive Binder

### 5.3.1 Performance

Last but not least, we turn our attention to a performance evaluation of the Adaptive Binder.

### Methodology

We have assessed the impact of the Adaptive Binder in two main scenarios:

1. as a tool added to the Mobility Predictor and the Semantic Reasoner to recover from incorrect predictions:
2. as an alternative to the Mobility Predictor and the Semantic Reasoner to addresses those cases for which no historic data is available (e.g., a user visits a new city).

To do so, we focus on sequence compositions and we look at the impact of the Adaptive Binder on the number of successfully completed compositions, both in the case where services are elected randomly (case 1), and by the Semantic Reasoner (case 2).

As for the Adaptive Binder ability to predict disappearing devices, we take a very different approach and we (i) analyse the correlation between physical movements and signal strengths variation for a wireless connection in real test-beds (WiFi) to then actually (ii) examine the predicting algorithm in a real human scenario (i.e., device dynamics in a London airport during a busy day).

## Metrics

The main and first use of the Adaptive Binder is to recover from failing compositions by adjusting to changes in the surroundings. Hence, to assess the Adaptive Binder in regards to this feature, we adopt as scoring metric the ratio of compositions started and successfully completed. As for the departure prediction algorithm, our preliminary analysis looks at the estimates of the time to disappear for a provider.

## Datasets

To assess the Adaptive Binder in terms of compositions successfully recovered, we use the same dataset used for the Mobility Predictor and Semantic Reasoner to simulate people/devices encounters.

As for the preliminary analysis on a possible method for predicting departing devices, we have looked at a real scenario and we have collected the signal strengths of a device moving within Gatwick South Terminal air-side ground and first floors. Although we have captured and analysed different transitions (about 10) around the London Gatwick terminal, in the following we just discuss the dataset (and results) for a specific transition; this is because we obtained very similar results for all other transitions captured.

This transition describes the following scenario. On a Friday late afternoon before half term holiday, a user moves around Gatwick Airport South Terminal Airside with a wireless device connected to the local WLAN from the first floor main departure screen monitor to go to the Deli Bar opposite to FCUK where she seats; she then walks to the lift, gets the lift to go to the ground floor, travels around the chairs area to then proceed to gate 24 (see picture 5.10). The dataset so obtained includes traces that elapse on an interval of about 152 seconds. In the following, we refer to this dataset as the Gatwick Airport Dataset.

The traces in the dataset relate to a device connected to an Access Point via WiFi 802.11b, with the wireless driver forced to maintain the connection with the Access Point the device first connects to and set to use the maximum sampling rate possible (i.e., about 150 ms). Also, the user's walking speed is a high pace (i.e., 6.8 km/h) as monitored through a Nike iSense sensor. We aimed to such high walking speed in order to put ourselves in the worst mobility scenario for prediction purposes. The Access Point the user is connected to is located near the entrance after passport control. If we break up the scenario depicted above, the following steps can be outlined:

1. the user moves from the first floor main departure screen monitor to go to the Deli Bar;
2. the user seats at the Deli;

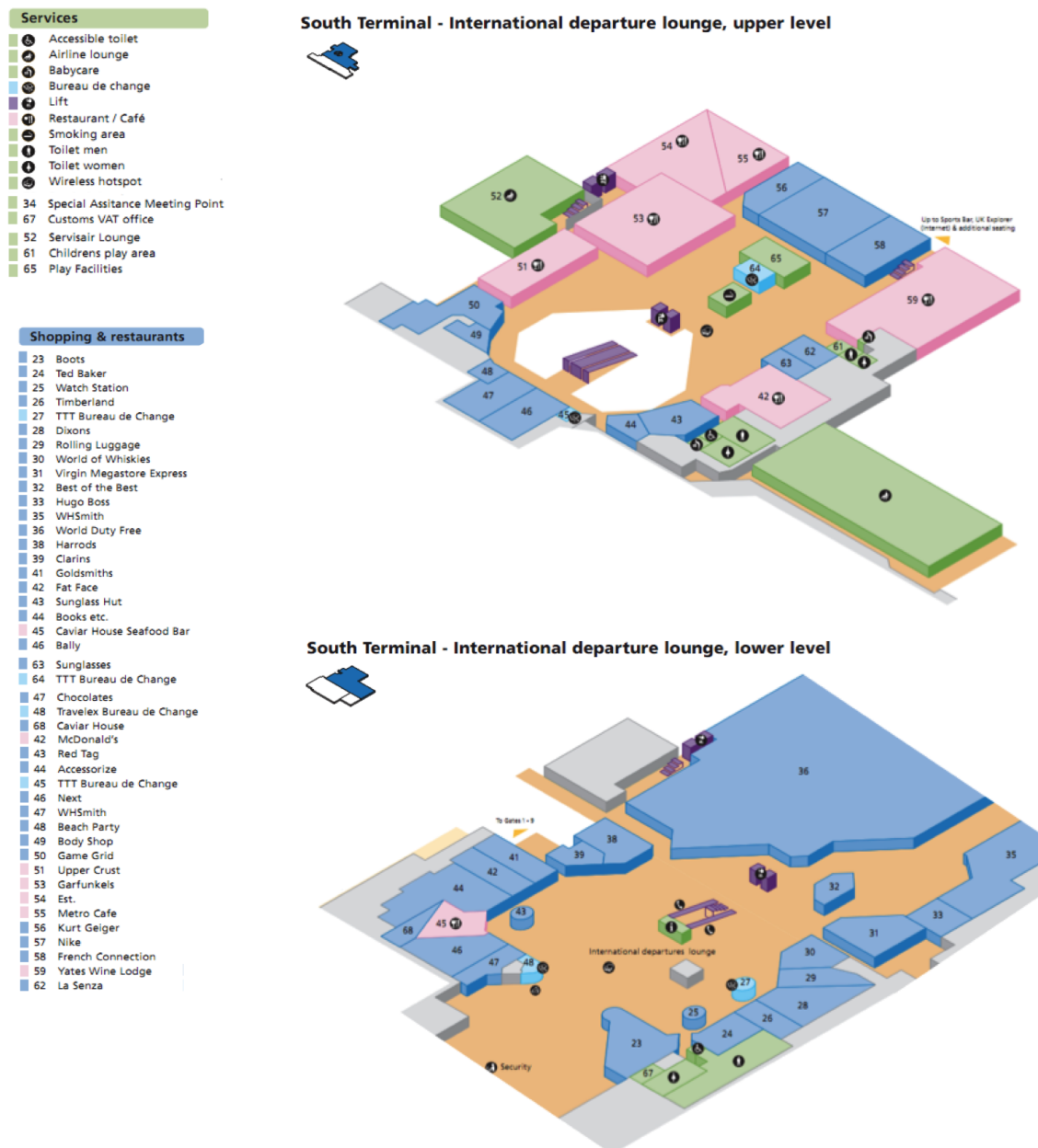


Figure 5.10: Gatwick Airport South Terminal, Airside

3. the user walks to the lift;
4. the user gets the lift to go to the ground floor;
5. the user travels around the chairs area;
6. the user walks to gate 24.

If plot the signal strength monitored, we can noticed that there is a strong correlation between the steps and the signal strength values recorded, as the diagram 5.11 shows.

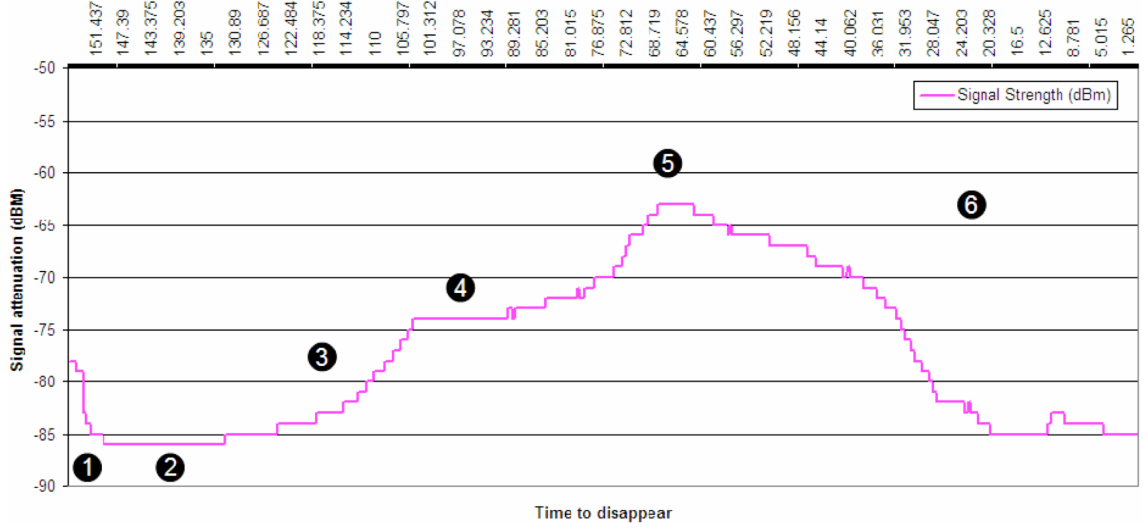


Figure 5.11: Time-to-disappear

This dataset – although encompassing a very short period of time – is significant as all the main factors affecting the signal propagation are considered (e.g., significant human presence, metallic walls, ceiling and floors, tunnelling effect).

## Benchmarks

We have assessed the Adaptive Binder in terms of recovered composition both for a traditional random discovery tool and the reliable discovery method built on the Mobility Predictor and the Semantic Reasoner. In doing so, we have compared:

- the random selector augmented with the adaptive binder against the simple random selector;
- the Semantic Reasoner built on the Mobility Predictor augmented with the Adaptive Binder against the Semantic Reasoner on the Mobility Predictor alone.

As for the Adaptive Binder Predictor for disappearing devices, we have adopted as reference benchmark “God’s view”. In other words, we have compared our algorithm to an ideal and perfect prediction method that estimates departure time based on exact actual

times.

### Setup

To assess the added reliability brought in by the Adaptive Binder, we have conducted two sets of experiments: in the first, we have used the Binder in combination with the Mobility Predictor and the Semantic Reasoner; in the second, we have used the Binder as an alternative to the Mobility Predictor (and Semantic Reasoner) instead. The former caters for scenarios where there is a high degree of regularity, and thus the Binder is simply expected to help with (sporadic) unforeseen deviations for estimated co-locations; the latter caters for situations where providers (and their behavioural patterns) are unknown instead, and thus their co-location cannot be estimated.

In both cases, as for the Semantic Reasoner, we have replayed the connectivity traces and we have associated to each device in the encounters dataset (i.e., the MIT Reality Mining) one and only one service type. Hence, we have considered the composition in *sequence* of  $n$  services of  $n$  distinct types and considered just situations where at least  $n \times 3$  devices were co-located. We have repeated these same experiments varying both  $\Delta t_i$  and  $n$ .

In order to preliminary assess the Adaptive Binder Prediction method for departing devices we have conducted two main sets of experiments.

A first set of experiments simply looked at validating the underpinning assumption that the signal strength of a wireless link is a reliable physical measure of relative movements. This was needed because algorithms that use signal strength are known to offer poor accuracy in computing a device position. To do so, at first we have monitored the signal strength between two devices that are static at different distances (i.e. very close, close, distant and very distant). In this experiment we have looked at a device connected to an Access Point via WiFi 802.11b, with the wireless driver forced to maintain the connection with the Access Point the device first connects to, and set to use the maximum sampling rate possible (i.e., about 150 ms). With the same driver settings, we have then gauged the suitability of the signal strength attenuation to provide information about the movement direction of a device in relation to an Access Point (i.e., getting closer and moving away) across different controlled scenarios with the signal propagation more or less subject to tunnelling effect, but not to random human presence. The second set of experiments aimed at validating the prediction method in a real scenario with human presence. To do so, we have processed all traces in the Gatwick Airport Dataset and we have computed, for each sampling, the forecasted time-to-disappear as for Equation 4.2 and Equation 4.3. In particular we have looked at the case where the signal variation speed  $d\omega/dt$  is averaged for the last 4 or 8 or 16 or 32 samplings.

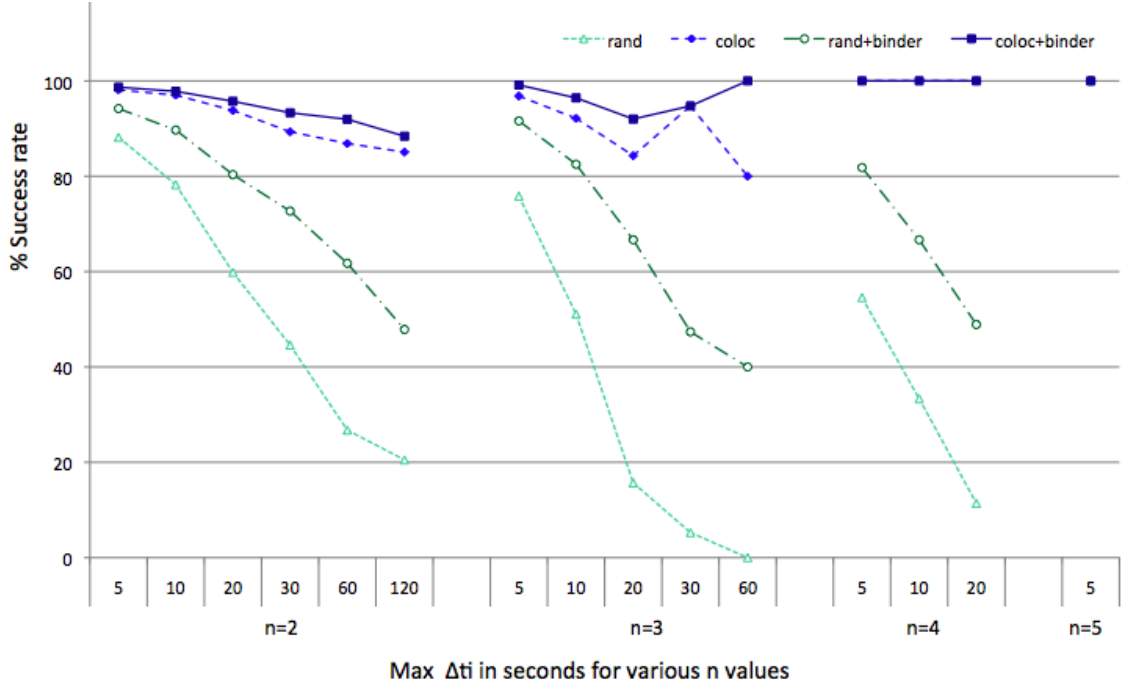


Figure 5.12: Adaptive Binder - Success rates

## Results

Figure 5.12 illustrates the results, in terms of increased reliability, obtained when using the Adaptive Binder component, both on top of the Mobility Predictor (and the Semantic Reasoner), and independently of them (thus in conjunction with a random selection) for sequential compositions only. As expected, the use of the Adaptive Binder improves the performance both for random service selection and for the discovery tool built on the Mobility Predictor (and the Semantic Reasoner). In particular, the increase of reliability experienced when using the Binder in conjunction with the random selection is such to suggest that compositions can be attempted even in environments where behavioural patterns have not been observed, thus decreasing the number of missed opportunities overall.

As for the experiments on the method forecasting device departs, we observe that experiments of the first set successfully gauged the suitability of the signal strength attenuation to provide information about device movements. In fact, we found that there was a strong correlation between the distance of two devices (i.e., very close, close, distant and very distant) and their signal level. More precisely we reported an average attenuation of 33 dBm, 47 dBm, 71 dBm and 84 dBm for very close, close, distant and very distant devices, respectively. Also, we observed a variation of signal attenuation of only 1-2 dBm between contingent samplings, and of 3-4 dBm between samplings recorded few minutes away.

Figure 5.13 compares the estimated time-to-disappear to the actual one measured (black line). The time is provided in seconds. X-Values represent the seconds after which the device lost connection with the Access Point, whilst Y-Values represent the predicted time-to-disappear for the cases where the signal variation speed  $d\omega/dt$  has been averaged across the last 4 or 8 or 16 or 32 samplings and it is not rounded to Equation 4.3.

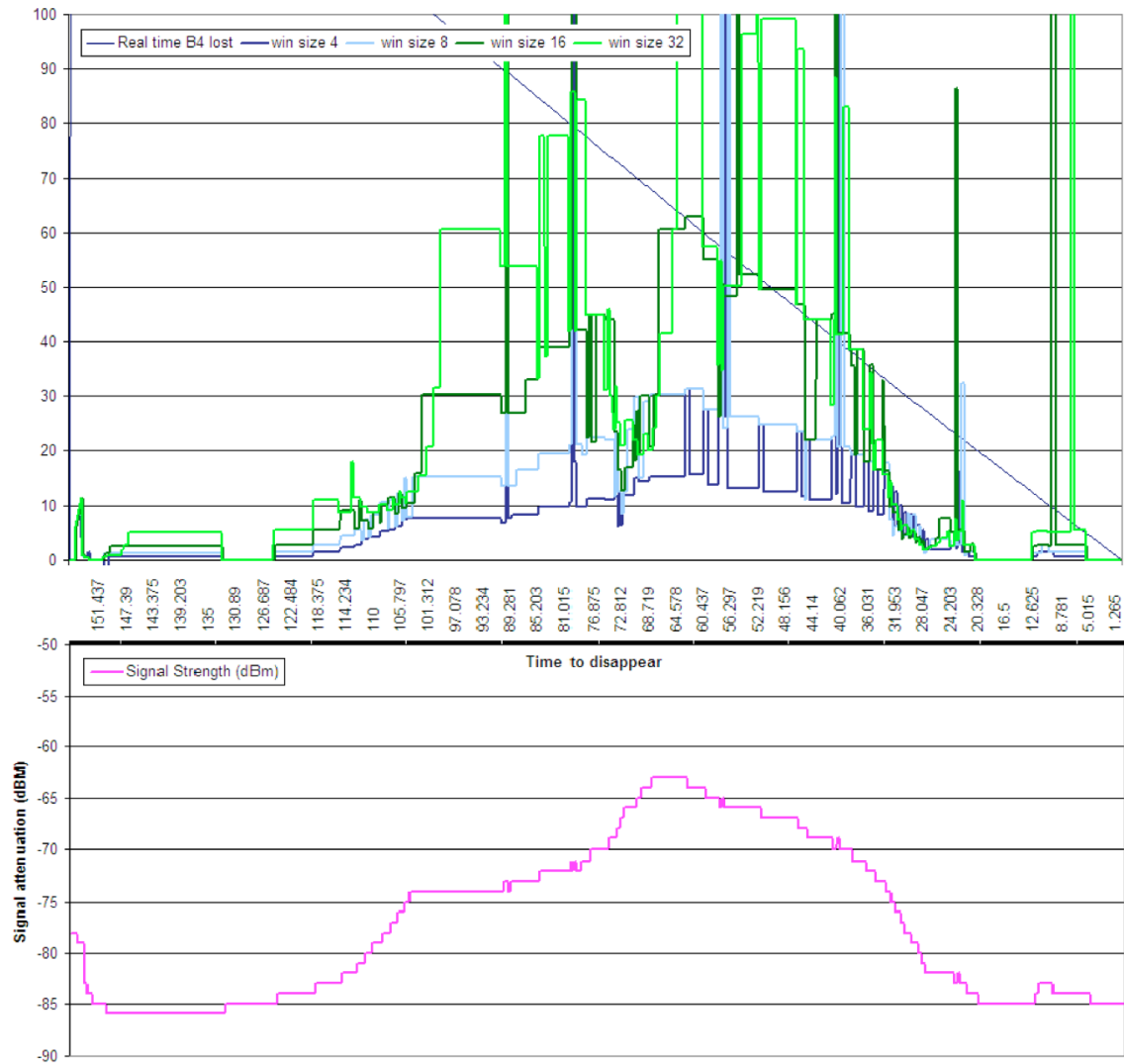


Figure 5.13: Predicting changes to the Environment - Time-to-disappear

Some key observations can be made:

- The prediction curves (top) are strongly correlated with the signal allure (bottom) and hence the movement dynamics;
- The longer the interval across which the signal variation speed is averaged, the lower the ability of the prediction model to react to changes;

- Although all prediction curves present peaks, such peaks are punctual and could be lowered or cancelled by filtering out or smothering great punctual variations (e.g., by checking the signal strength derivative).

The impact of the overestimating device mobility  $d\omega/dt$  as per Equation 4.3 is shown in Figure 5.14. In fact, Figure 5.14 depicts the estimated time-to-disappear when we use Equation 4.2 and Equation 4.3 as we vary  $\alpha$ . The empirical findings confirm the theory that, the greater the value of  $\alpha$ , the lower is the time-to-depart estimated compared to the actual one.

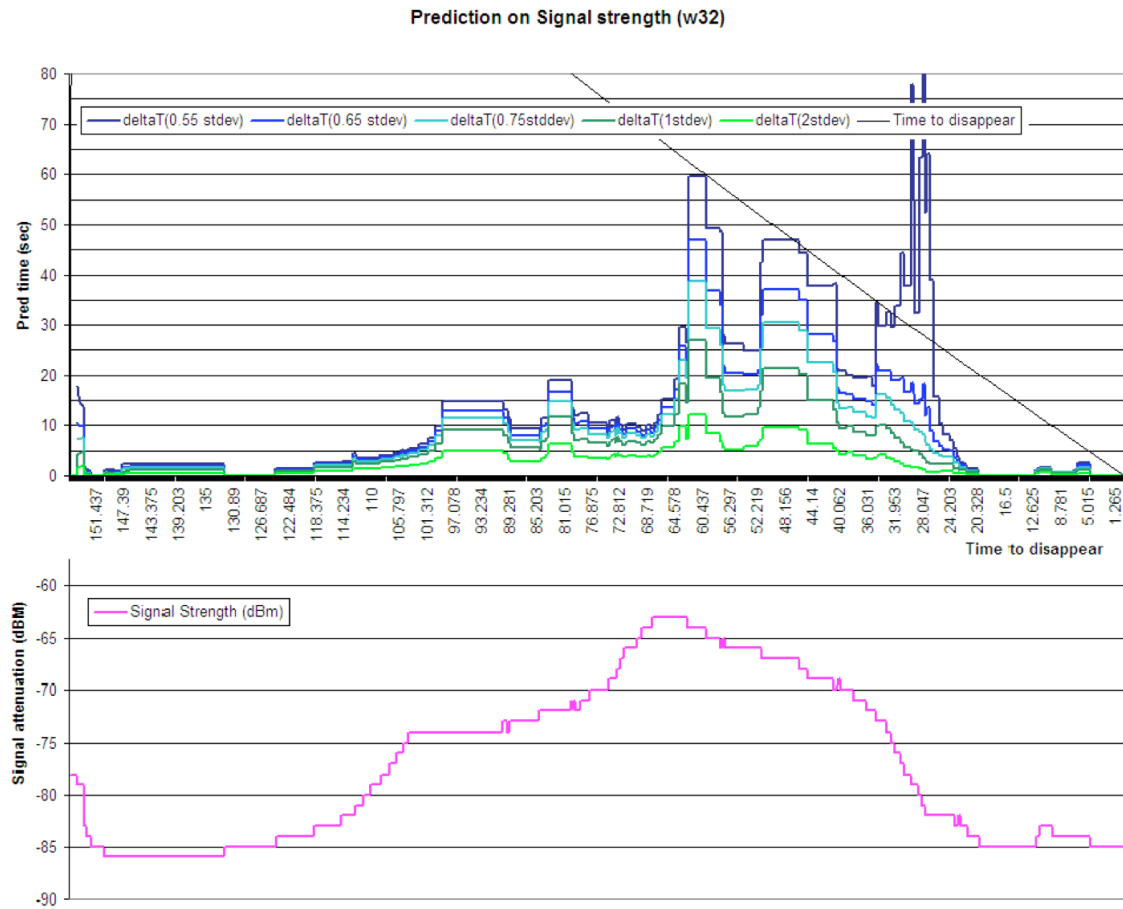


Figure 5.14: Predicting changes to the Environment - Time-to-disappear varying on alpha

It can be noticed that the use of the standard deviation enables to provide more conservative and hence more reliable prediction models.

### 5.3.2 Overhead

To assess the overhead caused by the Adaptive Binder, we have analytically quantified its memory allocation and processing cost.

#### Primary Memory Allocation

The Adaptive Binder keeps track of all the service providers in the surrounding that match any of the service elements of the composition requested. This is a fraction of all the available providers, as not all providers would match a component element. Being  $n$  the number of providers in the neighbourhood, the memory overhead would be  $O(n)$ .

As for the Departure Prediction tool, we observe that for each of the  $k$  component services we would need to compute Equation 4.2, reported here for convenience:

$$\sigma_{P_i, C_j}(t) = \begin{cases} 0 & \text{if } \omega_{P_i, C_j}(t) \leq \omega_{min} \\ 1 & \text{if } \frac{d\omega_{P_i, C_j}(t)}{dt} = 0 \text{ and } \omega_{P_i, C_j}(t) > \omega_{min} \\ \frac{\omega_{P_i, C_j}(t) - \omega_{min}}{\lceil (d\omega_{P_i, C_j}(t)/dt)_{mW/sec} \rceil} & \text{otherwise} \end{cases} \quad (5.10)$$

where we approximate  $d\omega/dt$  with Equation 4.3 i.e.:

$$d\omega/dt = \text{avg}(\Delta\omega/\Delta t) + \alpha \text{stddev}(\Delta\omega/\Delta t) \quad (5.11)$$

As remarked in Section 5.1.2, both the Average and Standard Deviation of a series of values  $x_i$  can be computed incrementally and require to persist just 3 values:  $N$ ,  $\sum_{i=1}^N x$  and  $\sum_{i=1}^N x^2$ ; hence, if we consider  $k$  component services the memory overhead is  $O(3k)$ , i.e.,  $O(k)$ .

#### Processing

The Adaptive Binder involves new calculations just as a service disappears (or is deemed to disappear). In such case, it would simply organise, by descending predicted co-location times, the  $m$  other providers in the surroundings that can provide the service in question. The additional processing it is just  $O(m \log m)$ .

As for the Departure Prediction mechanism, we observe that Equation 5.10 implies  $3 \oplus +1 \otimes$  computations to update historical values and  $2 \oplus +4 \otimes +\nabla$  operations to calculate  $d\omega/dt$ . The value obtained is then used to divide the difference between the current

signal level and a minimum one (Equation 5.11). Therefore, the overall cost per device per sampling is  $6 \oplus + 6 \otimes + \nabla$ .

In the empirical experiments run at Gatwick Airport, the signal was sampled every 150 ms, thus needing to run about  $42 \oplus + 42 \otimes + 7 \nabla$  operations per second per monitored device. We conclude that, whilst the Adaptive Binder itself requires little extra overhead, the overhead due to the Departure Prediction tool is not negligible and thus we recommend its use only in very dynamics and unfamiliar environments and only for composition with very few component services.

## 5.4 Summary

In this chapter we have analysed both the performance and resource overhead of the discovery methods proposed for reliable compositions.

First, we have studied the accuracy of the Mobility Predictor; and subsequently, we have examined the ratio of completed compositions in the case the Semantic Reasoner is adopted alone, or in combination with the Adaptive Binder; finally, we have assessed the Adaptive Binder when employed by itself.

The processing cost and memory allocation of both the Mobility Predictor, the Semantic Reasoner and the Adaptive Binder have also been provided.

Overall, we have demonstrated that, by reasoning on device mobility patterns and the constraints imposed by the actual composition semantics, the reliability of composed services (i.e., the number of compositions started and successfully completed) can be significantly improved with a negligible additional resource allocation.



## Part IV

# Framework and Future Work



## Chapter 6

# Middleware Realisation

The previous chapters have provided new data structures and algorithms in support of the discovery of pervasive services (and content) that are reliable and relevant to end-users. We observe that the discovery of services (and content) makes sense just if it is instrumental for their execution. In other words, a discovery tool is useful only if it integrates within a framework that provides developers with abstractions to invoke and run (or play) the services (or content) discovered.

With this in mind, in this chapter we discuss a reference framework we propose for pervasive services. We first describe its overall architecture (Section 6.1), we then focus on the discovery component (Section 6.2), which realises the main contributions of this work (i.e., *personalised* and *reliable* discovery). We briefly touch upon its implementation (Section 6.3) and possible deployments (Section 6.4). Finally, we conclude with an examination of its programmatic complexity (Section 6.5).

### 6.1 Architecture

The middleware architecture we propose handles the selection and composition of pervasive services in mobile environments, thus abstracting this complexity away from the developer. We call such middleware *MoSCA* (for **M**obile **S**ervice **C**omposition **A**PI).

MoSCA achieves this goal building on three key concepts: a *Service Entity*, that statically describes the composition structure of a service, as well as its classification according to a pre-defined ontology; a *Binding Entity*, that captures the run-time association between a service entity and the device(s) that is(are) currently in charge of delivering such functionalities in the most reliable way; and a *Token Entity*, a container element that collects session information (e.g., input/output data) of a running service request.

The Service Entity and the Binding Entity correspond to the entities our Reliable Discov-

ery method relies on. A detailed description of these can be found in Section 4.3.

The Token Entity is a new concept instead, and represents a running service request. It serves three purposes: first, it serves to collect the input information for a request (if any); second, it is used to capture and share, among the various component services, the status of the request (e.g., input and output parameters); third, it controls the service execution flow by means of a classic token passing protocol. Let us consider, for example, a request to sequentially execute services  $s1$ ,  $s2$  and  $s3$  (Figure 6.1 top); upon receiving this request, the MoSCA framework creates a Token Entity and passes it to the first service in the composition, in this case  $s1$ . As a result of its execution,  $s1$  updates the Token Entity with temporary results and session information; the same Token is then passed as input to the next service  $s2$  to be executed, and so on until the last service in the composition has been completed and the result of the composition is extracted from the Token and returned to the entity that initiated the service request. At any time, a service can only be executed if it has control of a Token; depending on the actual composition semantics, a Token Entity may thus have to be split and later consolidated, for example, when executing parallel branches (Figure 6.1 bottom).

The MoSCA instance that initiates a request always maintains an updated local copy of the Token entity to enable rollback from component services' failures.

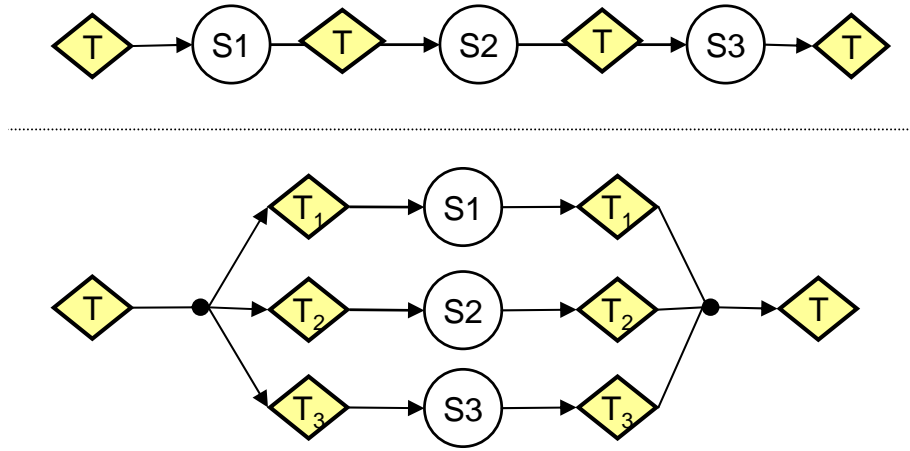


Figure 6.1: Token Passing Examples

MoSCA constructs upon these three entities – Service, Binding and Token entities to realise a framework that seamlessly carries out reliable and appealing compositions of services in mobile environments. An overview of the MoSCA Architecture is provided in Figure 6.2.

MoSCA consists of four main modules: the *Service Manager*, the *Service Analyser*, the *Service Discoverer* and the *Service Coordinator* that exchange objects of two types, *Service* and *Token*. As previously discussed, the former is used as a service descriptor, while the

latter carries information related to an on-going service request.

MoSCA handles two types of requests: *target service requests*, where the invoking component requests to execute a specific service; and *discovery service requests*, where the client requests to execute any service of a given type and matching a specific context (e.g., user profile).

The Service Manager acts as a façade and exposes two main end-points (for the same interface *requestService*): one for the *target service requests* and one for *discovery service requests*.

Upon receiving a **target service request** for Service  $s$ , the Service Manager invokes the *Service Analyser*, whose goal is to ‘understand’ the request: more precisely, the analyser decomposes  $s$  into component services  $s_1, s_2, \dots, s_n$ , and returns a composition semantic for them (e.g.,  $s_1 \text{ seq } s_2 \text{ seq } \dots \text{ seq } s_n$ ).

The Service Manager then passes the annotated decomposition to the *Service Discoverer* component, whose main goal is to choose providers  $p_1, \dots, p_m$ , among those available in the current environment, that will satisfy the request (i.e., that will be able to deliver services  $s_1, \dots, s_n$ ) and that will maximise the chances of successful service completion. Finally, the Service Manager passes the service decomposition and the selected providers to the *Service Coordinator*, which is then in charge of executing the request.

Upon receiving a **discovery service request** for a service (or more), the Service Manager first checks of what type (e.g., news, radio) the service(s) to be discovered need to be. It then invokes the *Service Analyser*, whose goal this time is to ‘identify’ services of the type required. The Service Analyser thus returns a set of services of such type. The Service Manager then interacts with the Service Discoverer to both select services and choose their providers. Finally - as for target service requests – the Service Manager passes the service decomposition and the selected providers to the *Service Coordinator* that is then in charge of executing the request.

In the following, we describe each of these core components (Figure 6.3), their interfaces and their interactions more in detail.

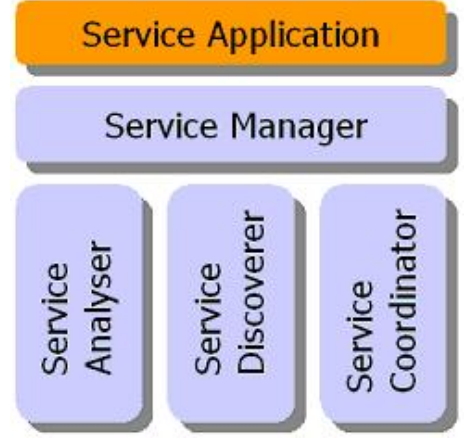


Figure 6.2: MoSCA Overview

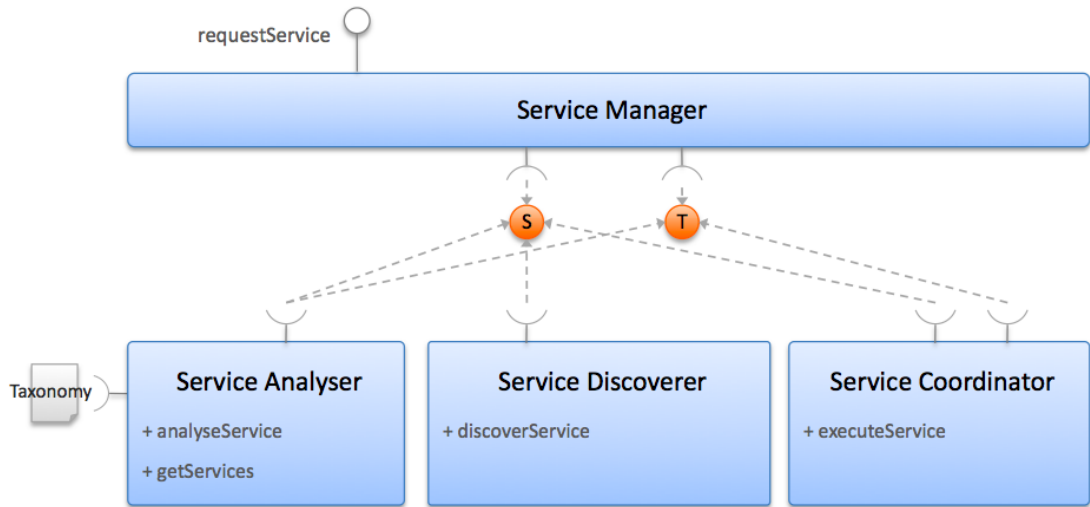


Figure 6.3: MoSCA Components

### 6.1.1 The Service Manager

The Service Manager component is the access point to the composition framework. It provides two main end points: one to execute a specific service (for the target service requests), and one to discover and execute services (for the discovery service requests). Overall, upon receiving a target service request for a service  $s$ , the Service Manager creates a Service object and passes it to (i) the Service Analyser to decompose it, (ii) the Service Discoverer to bind it to service providers and (iii) the Service Coordinator to execute it. Similarly, upon receiving a discovery service request, the Service Manager (i) interacts with the Service Analyser to restrict the eligible services based on their type; (ii) invokes the Service Discoverer to elect services as well as their providers and (iii) calls the Service Coordinator to execute the service(s) chosen. The main difference between the two end points is what is being passed to the Service Manager.

In the first case, the application/developer invokes the `requestService` method identifying a specific service. This is done either by passing a unique taxonomy identifier (e.g., a URI) and, within it, the service identifier, or by passing a Service Object. In the second case the application/developer calls the `requestService` method indicating a Taxonomy Filter and a Context. The Context entity captures anything that could help the Service Discoverer to best elect the services to run. In our specific implementation the Context consists of a User Profile. This is because our specific implementation of the framework selects/recommends services (and content) based on the end user's taste.

In both cases, the Service Manager creates a Token object – if not passed in the request – and orchestrates all interactions with the Service Analyser, the Service Discoverer and the

Service Coordinator to carry out the request, leaving the invoking application unaware of whether the requested services are single or compound, and whether they reside on the same device or it is accessible from other devices in the proximity (in particular, at a single hop distance).

### 6.1.2 The Service Analyser

The Service Analyser component reasons about the information collected in service taxonomy and/or ontology files and provides two key functionalities: the ability to decompose a service according to its taxonomy, and the ability to filter Services within a taxonomy given a Taxonomy Filter and possibly an input Token (so to get a set of Services that match the Taxonomy Filter and are such that in the Token object all required input are passed). The interfaces for these two functionalities are *analyseService* and *getServices*, respectively.

Upon receiving an *analyseService* request, and with it a Service object  $s$  from the Service Manager, the Service Analyser decomposes  $s$  into component services  $s_1, s_2, \dots, s_n$ , and returns the same Service object back to the Service manager, now enriched with the service (de)composition semantics (e.g.,  $s = s_1 \text{ seq } s_2 \text{ seq } \dots \text{ seq } s_n$ ). Different decompositions are possible; the analyser favours those that rely on the minimum number of component services (i.e., services that appear higher up in the taxonomy), as this implicitly minimises the number of providers required to deliver a service. As soon as the first decomposition has been analysed, the updated Service entity is returned to the Service Manager, ready to be processed by the Service Discoverer. The Service Analyser keeps looking for alternative (more fine-grained) decompositions (in case there are not providers in the environment capable of delivering the more coarse grained ones) and it updates the shared service object.

Let us revisit the example discussed in Section 4.3.2 (see Figure 4.3 in page 126) of a service  $s$  that can be decomposed as  $s = s_1 \text{ seq } (s_2 \text{ parallel } s_3)$ , with  $s_3$  that can either exist alone or be further decomposed as  $s_3 = s_4 \text{ seq } s_5 \text{ seq } s_6$ . Depending on the providers available in the surrounding (and on their predicted reliability), the executed composition may vary:  $s$  may be delivered as a single service by a single provider, if one exists; if not, the Service Analyser would look for fine-grained services  $s_1$ ,  $s_2$ , and  $s_3$ ; should a provider of  $s_3$  not be available, the Service Analyser would further decompose  $s_3$  into  $s_4$ ,  $s_5$ , and  $s_6$ , thus informing the Service Discoverer to look for providers of these fine grained services. As previously said, we assume that a pre-defined taxonomy (or ontology) exists to map a requested service  $s$  to a decomposition  $s_1, \dots, s_n$  with associated semantics (e.g., [Mokhtar et al., 2006]); this taxonomy can be either universal (and built, for example, on OWL-S [Martin et al., 2004]) or can be specific to a domain. The only requirement that MoSCA imposes on the taxonomy used is that a universal ID can identify it and that a local identifier can identify services within the taxonomy.

Upon receiving a *getService* request (and the Taxonomy Filter and Token within it), the Service Analyser traverses the tree of the service ontology in use to find all the services matching the Taxonomy Filter and the input Token (if any). Hence, it creates and returns a ServiceSet of eligible services.

### 6.1.3 The Service Discoverer

The Service Discover provides two main end points: one to discover the providers for a Service (for the target service requests), and one to discover services as well as their providers (for the discovery service requests). In the first case we talk about “provider selection”, in the second case we talk about “service selection”.

In the case of provider selection, the Services Manager invokes the *discoverService* method and passes as *input the Service object whose providers need to be elected*. This is after the Service Analyser has annotated the Service object with decomposition information. Here, the goal for the Service Discoverer is to choose providers  $p_1, \dots, p_m$ , among those available in the current environment, that will be able to deliver services  $s_1, \dots, s_n$  and that will maximise the chances of successful service completion.

With reference to the example in Figure 4.5 in page 128, the Service Discoverer component will thus first look for a provider of service  $s$  (i.e., top-level service type being found in the taxonomy); should this fail, it would then try to find providers of  $s_1$ ,  $s_2$ , and  $s_3$  (note that  $s_2 \& s_3$  does not exist in the taxonomy, and thus a single provider of such service is not looked for). Should this fail too, the most fine-grained decomposition would be attempted, with the Discoverer looking for providers of  $s_1$ ,  $s_2$ ,  $s_4$ ,  $s_5$ , and  $s_6$ .

As a service provider is elected, the Service Discover is then responsible to create and maintain the service bindings to such provider. Note that *MoSCA* aims to deliver only those services that are deemed reliable, that is, services which have high probability to complete successfully, before any of the providers involved moves out of range of the requester. It is the responsibility of the Discoverer to quantify the reliability of the currently analysed decomposition, pondering alternative bindings should more than one provider of the same service be available. If a set of providers is found for which the overall probability of successful completion is above a given threshold, the Service Discoverer instructs the Service Manager to carry out the composition.

In the case the Service Discoverer is used to both discover services and their providers, the Service Manager still invokes the *discoverService* method. This time, though, it does not pass a Service object; instead, it *inputs a Service Set and a Context* (e.g., in our specific case a User profile). The goal of the Service Discoverer is to select, filter and rank the services available in the surroundings, within the Service Set, so to match the input Context and to then pick their providers as discussed for the first case. This results in the Service Discover creating a Service object, which is returned to the Service Manager,

once the service-to-provider bindings have been set.

#### 6.1.4 The Service Coordinator

The Service Coordinator component is in charge of executing the service request. It provides just one interface: *executeService*.

It receives, from the Service Manager, both an annotated Service object, containing information about the (de)composition semantics and the selected providers (via Binding entities), and a newly created Token object storing (if applicable) the input parameters to the service request. The Service Coordinator will then carry out the request, updating the data stored in the Token object, splitting and consolidating it, if necessary, at every stage of the composition as the request is being processed. The bindings used during service execution are dynamically re-assessed, in order to react to unforeseen departures of providers; more precisely, each service decomposition is annotated with ‘aspects’, that is, entry gates in the execution flow where a re-assessment of the current environment should take place (with potential re-binding of services) before the service execution proceeds. For example, a possible aspect within the Smart Media Player compound service described in Section 4.2 is the entry to the loop: before the execution of a new iteration, the Service Coordinator notifies the Service Manager, which in turn asks the Service Discoverer to re-assess the reliability of the composition, with new bindings potentially being formed. If one or more providers become no longer available between re-assessment entry points, the Service Coordinator notifies the Service Manager that, depending on the annotated decomposition, determines if the overall service can still be carried out (e.g., with roll-back to the previous entry point and rebinding) or if a failure must be reported.

In the following section, we describe in more details the main contribution of this work, that is, the service discoverer component, whose goal is, on one side, to elect the services more likely to match an end user’s tastes and, on the other side, to select the providers that maximise the reliability of a composition, reacting to unforeseen changes if necessary.

## 6.2 The Service Discoverer and the Discovery Pipeline

We model Service Discovery as a filtering/ranking process implemented via a Discovery Pipeline (i.e., a sequence of Filters run in turn) as depicted in Figure 6.4. As a discovery request is issued, we input to the Discovery Pipeline the set of all the services (and their providers) available in the surroundings, hence we run each filter in the pipeline one after the other, and we obtain a ranked list of service instances (i.e., services and their providers).

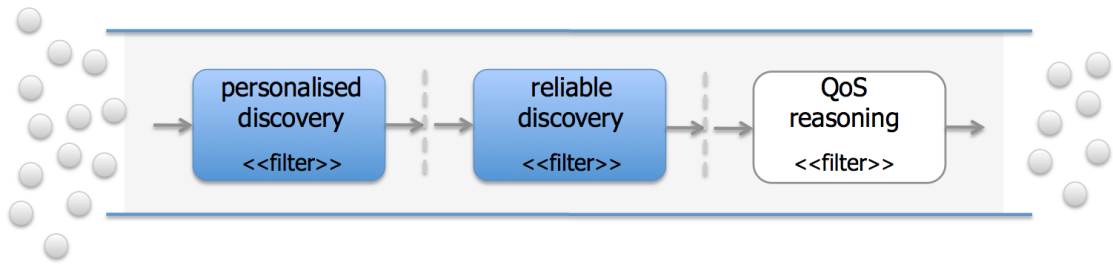


Figure 6.4: Discovery as Filtering/Ranking

### 6.2.1 Service Filters and Provider Filters

Filters in the Discovery Pipeline can be of different types. They can either act upon Services (*Service Filters*), Services Providers (*Provider Filters*) or both (*Service-Provider Filters*).

Filters of the first type – i.e., Service Filters – operate on Services and determine the set of eligible services based on user’s preferences (or more in general a context profile), regardless of their actual providers. For instance, the Filter built on the Personalised Discovery method presented in Chapter 2 is a Service Filter.

Filters of the second type – i.e., Provider Filters – control service providers based solely on the qualities of the provider (regardless of the service they provide). In doing so, they dictate the bindings for the eligible services and – as a side effect – they could further restrain the set of selectable services (this would be the case if a suitable provider is not found for a service). For instance, the Filter built on the Reliable Discovery method presented in Chapter 4 is a Provider Filter.

Filters of the third type – i.e., Filters that are both Service Filters and Provider Filters – are filters that select service instances based both on the qualities of the service and their actual providers. Filters of this type are useful for providers’ properties that depend on the actual service considered (e.g., latency, RTT, etc.).

### 6.2.2 Service Discovery

The actual selection of a service and its provider is determined by (i) the ordering of the Filters in the Discovery Pipeline, (ii) the filtering policy of each Filter and (iii) the overall policy for the Discovery Pipeline. In fact, each Filter in the Discovery Pipeline is executed logically one after the other and the output set (of services and providers) of a Filter is input to the next Filter in the pipeline.

Each Filter – regardless of its type – sets a score to the entities it controls. Service Filters grade services; Provider Filters grade providers; Service-Provider Filters grade service-provider pairs. Also, Filters define a maximum output set size  $m$  and a threshold value  $th$ . These two values determine the output set of the Filter, as a Filter purges all entities whose score value is lower than the threshold  $th$  or that are not in the set of the  $m$  best entities.

This allows to tailor the overall Service Discovery method still re-using existing filters. For instance, if we set the Discovery Pipeline to have a Personalised Discovery Filter first, then only services matching an active user’s preferences will be selected. If we set the Discovery Pipeline to have a Reliable Discovery Filter first instead, then only the services whose providers are reliable (from a co-location perspective) will be elected. The actual order of Filters within the Discovery Pipeline has impact on the overall resource overhead of the discovery method; this is because, in general, the smaller is the set of services or providers input to a Filter, the lower the processing and memory resources required. For example, in an highly dynamic environment, we could chain first a Reliable Discovery Filter and then a Personalised Discovery one to minimise the overall resource overhead and provide greater discovery processing speed. In fact, most likely, the Reliable Discovery Filter would discard a great number of providers (and their services), and so, just a very restricted set of services would need to be considered by the Personalised Discovery Filter. On the other hand, in an highly stable environment, the Reliable Discovery Filter would not purge many providers and services; hence, a Personalised Discovery Filter could be more effective and could be set first within the Discovery Pipeline. Ideally, Filters could be re-ordered at run-time, depending on the surrounding environment and on QoS of the system as a whole.

At run-time, the Discovery component maintains an association between services and providers in a data structure that we call Binding Map, depicted in Figure 6.5.

The Binding Map data structure consists of a Service list and a Service-to-Providers Hash Tree. The Service list contains a list of services relevant to a service composition request, ranked by their score. The Service-to-Providers Hash Tree persists, for each service, its providers in the surroundings, together with their provider scores and, where relevant, the actual binding and service-provider score.

Filters update the Binding Map data structure as they are run – logically – one after the other. Filters that act on the selection of Services modify the Service list, whilst Filters that impact the set of eligible service Providers alter the Service-to-Providers Hash Tree.

At discovery time, the Service Discoverer accesses to the Binding Map data as follows. Whenever a *provider discovery request* is issued (for a target service request), the Service Discoverer traverses the Service-to-Providers Hash Tree to find the best binding (i.e. best provider). As a *service discovery request* is issued instead, the Service Discoverer retrieves

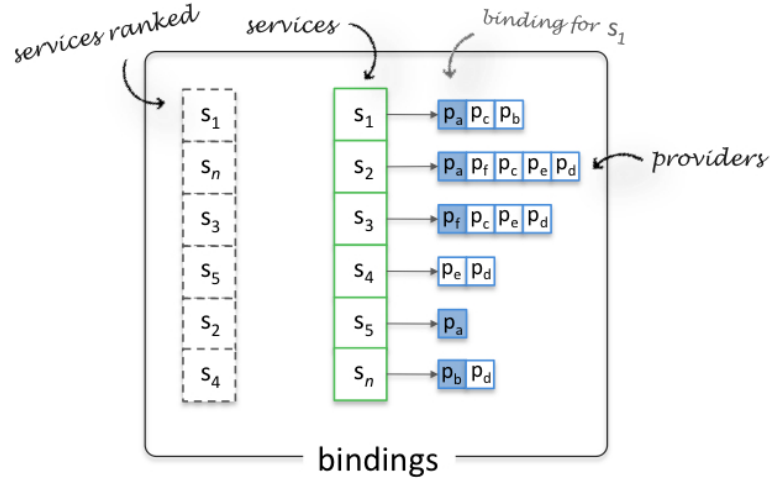


Figure 6.5: Bindings Conceptual Model

the set of eligible services from the Service list and then accesses to the Service-to-Providers Hash Tree to get their providers.

### 6.3 Implementation

We have implemented the full *MoSCA* Framework in Java 2 Micro Edition (Connected Device Configuration, Personal Profile 1.1 and Mobile Information Device Profile 2.0) and where possible in POJO (Plain Old Java Objects) building on the core Java classes common to JavaME, JavaSE and JavaEE. This has allowed us to implement a middleware suited to mobile environments and, at the same time, apt to run as part of any Java framework. A package has been developed for each main component. A Factory pattern has been used for the retrieval of a Service Analyser, a Service Discoverer and a Service Coordinator; in so doing, it is possible to configure *MoSCA* to use various implementations of these interfaces, and to dynamically discover and instantiate (by reflection) newly available implementations. As previously mentioned, the Service entity has been realised using the Composite pattern, thus intuitively representing the recursive nature of a composition. Moreover, Service objects are designed intentionally not to have a direct reference to their providers. Provider instances can only be retrieved via the Bindings singleton class; *MoSCA* can thus keep monitoring and updating the best providers of the various services in an asynchronous manner. As previously anticipated, we use Cache Management to cache users' similarities, preference predictions and surrounding providers' co-location times. Finally, the Observer pattern is used both to monitor the environment and find available services, as well as handling the exchange of user profiles. In total, the implementation occupies 51 kB (compressed), making it suitable for modern mobile devices. We have also

implemented the full framework in Java 2 Standard Edition, for deployment on resource rich devices, as discussed below.

## 6.4 Deployment

*MoSCA* can be deployed in different ways, depending on the resource capabilities of the device under consideration. It can be deployed as:

- a complete and stand-alone *Library*, for modern mobile devices (e.g., iPhone, Android phones) that are resource rich (both in terms of computational capabilities and memory), so to be able to run the full framework;
- a *Stub*, for devices that need to run a minimal configuration instead, thus relying on an external device running the full *MoSCA* framework to handle their service requests;
- a *Server*, typically on devices embedded in buildings (e.g., around the university campus or at the airport), exposing *MoSCA* functionalities to applications and services running on other devices.

In the reminder of this section, we describe each of these deployment approaches in turn.

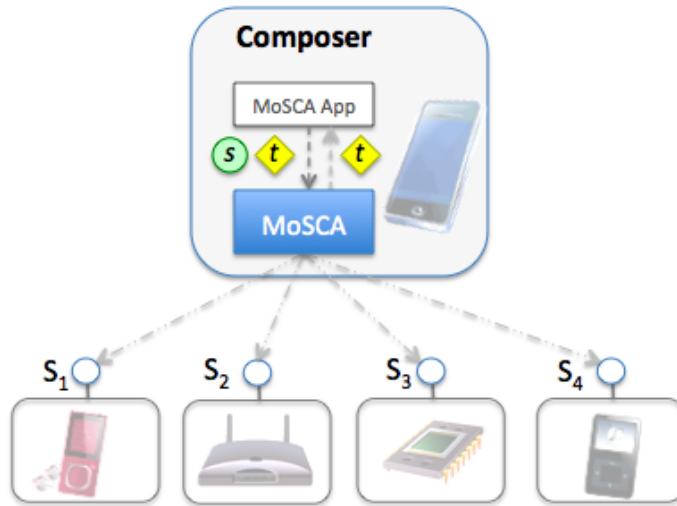


Figure 6.6: *MoSCA* deployed as a Library

For modern mobile devices, *MoSCA* will be most commonly deployed as a fully-functional library: as shown in Figure 6.6, *MoSCA* applications running on the device will issue

service requests to the underlying *MoSCA* framework, by passing a target or discovery service request (*S* in the picture) and, if applicable, a Token entity (*T* in the picture). The local *MoSCA* framework then analyses the requested service, it looks for providers in the surroundings, it co-ordinates the execution of the composed service, and it returns the final result to the originator, by means of the Token entity originally passed as input.

On less capable devices, *MoSCA* can run in its minimal configuration, that is, as a *MoSCA* stub delegating service requests to *MoSCA* Servers in reach. Applications running on the same device as the *MoSCA* stub interact with it as they would do in the case of a *MoSCA* Library deployment (Figure 6.7). The *MoSCA* stub then acts as a proxy and it passes the request to a *MoSCA* Server in the surroundings; the selection of the Server can be done using different strategies, from simple random selection, to longest predicted co-location time between the stub host and the *MoSCA* Server (as estimated by the latter). Upon completing a service request, *MoSCA* returns a Token to the client application, by using a mechanism very similar to HTTP X-Forwarded-For to identify the originating client (stub) itself.

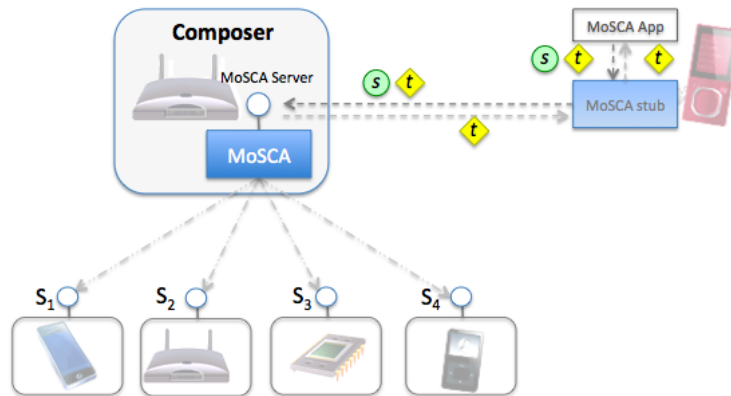


Figure 6.7: *MoSCA* deployed as a Server and as a Stub

On devices embedded in the infrastructure, *MoSCA* is typically deployed as a Server (Figure 6.7); in this case, *MoSCA* serves applications and services running on external devices, in addition to those executing on the server host itself. As before, requests are issued by passing a service request and a Token entity, with the main difference being that the originator now corresponds to a service or application running on a different device. *MoSCA* Servers are easily identifiable as they advertise and expose a *MoSCA* service interface, with reserved values for the taxonomy ID and service ID. Note that more complex deployments can be exploited too, for example, with a *MoSCA* Library or a *MoSCA* Server acting as stubs, in case the requesting device is running out of resources, or is not capable of answering the request with its local view and knowledge of the surroundings. The requesting *MoSCA* Library or Server may thus proxy a request to an external composer.

## 6.5 Programmatic Complexity

The evaluation results reported in Chapter 3 and Chapter 5 have demonstrated that the algorithms and data structures are light and effective. In this section, we move our attention to show how application engineers would use MoSCA in practice, to gather a sense of the programmatic complexity of our framework. To do so, we consider the lines of code required to:

- request the execution of a specific composite service (i.e., *target service request*);
- request the execution of a service appealing to an active user (i.e., *discovery service request*);
- *advertise* a service to the MoSCA framework.

### 6.5.1 Target Service Request

Requesting a specific service in MoSCA requires three steps: (1) the creation of a Composite Service object, specifying the taxonomy and service identifiers; (2) the creation of a Token object, containing input parameters in the form of attribute-value pairs, and (3) the invocation of the service itself, via the Service Manager. For example, in order to request a service *s*, with unique type *serviceID* within the taxonomy *taxonomyID*, and to pass as input a string variable of name *postcode* set to *WC1E6BT*, simply requires:

```
CompositeService s = new CompositeService(taxonomyID, serviceID);
Token t = new Token("postcode","WC1E 6BT");

ServiceManager sm = new ServiceManager();
sm.requestService(s,t);
```

Let us consider the case now of a composite service that is not associated to any taxonomy (hence its type is unknown); this service can however be described as the sequential composition of a service of type *serviceID1* and another service of type *serviceID2*, both according to the taxonomy identified by *taxonomyID*.

In this case, the composition has to be programmatically defined prior to invocation, as the composite service has not been pre-encoded in the taxonomy in use:

```
CompositeService s1 = new CompositeService(taxonomyID, serviceID1);
CompositeService s2 = new CompositeService(taxonomyID, serviceID2);
```

```

CompositeService s= new CompositeService(Service.SEQUENCE);
s.add(s1);
s.add(s2);

Token t = new Token("postcode","W1B 2EL");

ServiceManager sm = new ServiceManager();
sm.requestService(s,t);

```

Note that it is transparent to the application programmer whether component services (e.g., *s1* and *s2*) are themselves composite or basic, as all services are requested as Composite Service objects.

Rather than using MoSCA programmatic interface, specific services can be requested by passing a Uniform Resource Identifier (URI) to the Service Manager instead.

Given the URI definition: <scheme name>:<hierarchical part>[?<query>][#<fragment>], MoSCA expects to find both the taxonomy identifier and the service identifier within the hierarchical part of a URI, following the patterns: */tid-<taxonomy ID>/* for the taxonomy, and */sid-<taxonomy ID>/* for the service. Service parameters are passed as part of the URI itself, or separately using a POST method if the input data is too large to fit the <query> part of an URI. For example, the following fragment of code illustrates how to request a service of type 0034 according to the taxonomy identified by 01 using a Bluetooth schema:

```

String uri=
"btspp://0012F3008606:1;name=service/tid-01/sid-0034"

ServiceManager sm= new ServiceManager();
sm.requestService(uri);

```

The following fragment of code requests a service of type 0034 according to the taxonomy identified by 01 using an HTTP schema instead:

```

String uri=
"http://84.9.114.203/service/tid-01/sid-0034/"

ServiceManager sm= new ServiceManager();
sm.requestService(uri);

```

In case parameters have to be passed as well, we can use the URI query part as follow:

```
String uri=
    "http://84.9.114.203/service/tid-01/sid-0034/?postcode=W1B\%202EL";
ServiceManager sm= new ServiceManager();
sm.requestService(uri);
```

In case the service has not been associated to any taxonomy, but can be defined as the sequential composition of a service of type 0001 and of a service of type 0002 according to the taxonomy identified by 01, then the URI is enriched with the pattern */c-<composition type>/* as follow:

```
String uri=
    "http://84.9.114.203/service/tid-01/c-SEQ+01*0001+01*0002/";
ServiceManager sm= new ServiceManager();
sm.requestService(uri);
```

Note that, in all examples above, the identifiers for both service types and taxonomies have been trivialized to simplify presentation and indeed may not be numerical and that short (also, for clarity of presentation, we have not encoded all symbols in the URI as they should have been). However, the programmatic complexity involved in a MoSCA composite service request is minimal. In fact, for any new request, application engineers need only to create a Composite Service object and request its execution. They do not see its *mobility* and *service composition*; they do not know, neither they need to know, whether the service is simple or composite, and/or delivered by static or mobile providers. Overall, hence, MoSCA achieves an high level of transparency.

### 6.5.2 Discovery Service Request

Requesting a personalised service may be more or less complicated depending on the kind of constraint we want to impose on MoSCA discovery intelligence.

The simplest constraint would only dictate the type of service to look for. To do so, the developer/application must create a taxonomy filter by passing one (or more) *taxonomy ID-services IDs* pair(s) (or, for more flexible filters, an XPath expression [Clark and DeRose, 1999] [Berglund et al., 2010]). In either case, the developer needs to know the Taxonomy in use. For example, in order to request a service, within the taxonomy *taxonomyID* of unique type *sID1*, and to pass as input a string variable of name *postcode* set to *WC1E6BT*, requires:

```
Token t = new Token("postcode","WC1E 6BT");
TaxonomyFilter tf= new TaxonomyFilter(
```

```

        taxonomyID,
        sID1
    );

/*
 * or we could pass instead the XPath expression
 * taxonomy[@id=taxonomyID]//service[@id=sID1]
 */
PersonalProfile pp= new PersonalProfile(userID);
ServiceManager sm = new ServiceManager();
sm.requestService(tf,pp,t);

```

Note that the service request gets, as parameter, the user's profile, so to elect a service appealing to the user whose user identifier is *userID*

As in the case of target service request, the programmatic complexity is very low, as application engineers are only required to instantiate a Taxonomy Filter and a Personal Profile before invoking the service request.

### 6.5.3 Advertising a service

Service *advertising* follows a very similar mechanism to URI-based targeted service requests, with the exception that the first part of the <hierarchical part> must uniquely identify the device in the surroundings providing that service. For example, a device providing a service accessible via its Bluetooth interface could advertise:

```
btsp://0012F3008606:1;name=service/tid-01/sid-000034
```

while a device providing a service via its WiFi interface could advertise:

```
http://84.9.114.203/service/tid-01/sid-000034/
```

For what concerns the service registry, we use a lightweight distributed approach that builds on the Web robots.txt [rob, 2001] and Sitemap specifications [sit, 2006]. For instance the services exposed by a device via its Bluetooth interface would be enlisted at:

```
btsp://0012F3008606:1;name=service.txt
```

Similarly the services exposed by a device via its WiFi interface would be enlisted at:

`http://84.9.114.203/service.txt`

Indeed, though, any mobile service advertising protocol could be used for this purpose, even a simple broadcast.

## 6.6 Summary

In this Chapter we have presented a realisation of the discovery methods presented in Chapter 2 and Chapter 4.

First, we have contextualised the discovery methods proposed within the broader problem of service composition. We did so by means of a new architecture framework – MoSCA – that provides developers with abstractions to invoke and run the services discovered. We have then looked at the Service Discovery component in particular, and proposed to realise a new discovery paradigm as a chain of discovery filters.

We have presented how the framework can be deployed on devices with different resources. Finally, we have discussed the programmatic complexity that MoSCA offers application developers, both to formulate service requests and advertisements. We believe that the high level of transparency achieved sets MoSCA as a viable middleware for the development of pervasive services in mobile environments.

## Chapter 7

# Conclusions and Future Work

The research conducted in this thesis targeted the problem of service discovery in pervasive environments. We have tackled this issue in two ways: on one hand, we have eased the personalised discovery of services appealing to end-users by means of a mobile recommender system. On the other hand, we have reasoned on the composite nature of pervasive services and the mobility of their component providers to enable composite services that are reliable as if delivered by a single and traditional service provider. We have extensively evaluated all the algorithms and data structures we have proposed. Finally, we have discussed a practical realisation of these techniques, to guide application engineers on how to deploy and use them in practice. Overall, we have provided a discovery method that enables ‘better’ pervasive services, where by ‘better’ we mean both ‘more interesting’ to the user and ‘more reliable’.

In this last chapter, we summarise the main contributions of this thesis, we provide a critical evaluation of the goals attained, and, finally, we discuss some open issues that we leave for future research.

### 7.1 Contributions

The contributions that this thesis makes to the field of mobile service discovery are summarised below.

#### (1) Personalised Discovery

Our work contributes a lightweight and efficient method to recommend items that are mobile and of local interest. More precisely we provide a lightweight and fully distributed

mobile recommender system grounded on the characteristics of mobile environments and devices, whose precision and coverage are comparable with the ones of powerful recommender systems in use today. Instead of relying on global knowledge about users' profiles, we exploit the wisdom of local communities to compute recommendations, and we craft a virtual view of the local community's preferences. Users maintain their own rating profile locally on their mobile device and exchange them, using radio technology, between devices during periods of co-location. Each user thus gradually builds a virtual view of the local community's preferences. When a recommendation has to be computed, we assess dynamically whether the user is mass-like minded or individual: in the former case, no sophisticated recommendation algorithm is required, and the mean rating for any given item is returned; in the latter case instead, a novel lightweight algorithm is used to efficiently compute personalised recommendations. As part of the mobile recommender system proposed, our work also contributes a novel and very compact data structure called Rating Hash Tree that optimises, at the same time, storage and processing, grouping users' ratings by item and rating mark.

To mitigate the item cold-start problem for composite items, and to allow recommending novel composite items, we model composite items as teams, and we adopt team performance management both to compute the performance of a composition, based on the collective qualities of its components, and to infer individual qualities, based on the performance of the encompassing compositions.

This work has resulted in the following publication:

- L. Del Prete and L. Capra. “diffeRS: a Mobile Recommender Service”. In 11th International Conference on Mobile Data Management (MDM). Kansas City, Missouri USA. May 2010;
- L. Del Prete and L. Capra. “moRe: a Lightweight Recommender Service for Pervasive Computing”. Journal: IEEE Transactions on Mobile Computing (TMC). Pending.

## (2) Reliable Discovery

Our work contributes a mobility-driven filtering method that allows (i) to initiate only compositions likely to be successfully completed, (ii) to elect providers that optimise the reliability of the overall composition and (iii) to monitor and react to changes in the environments so to maximise the number of compositions completed.

More in details, our work contributes a lightweight method to predict the co-location times of devices in the surroundings. We build on the high degree of regularity that people show in their activities and we observe that the availability of pervasive services is naturally correlated to users' habits. In fact, the relative mobility between service providers and

consumers relates to people mobility patterns, either because those services are made available through people personal devices, or because they are embedded within building and structures people traverse. Hence, we record the relative mobility patterns of pervasive services and we learn the providers' co-location patterns over time, to then predict co-location times between component services.

We thus propose a method to decide if a composition should be attempted. First, we analyse the composition semantic to deduce the co-location requirements on component services that would ensure composite services to be executed successfully. Consequently, we ponder the co-location estimates against the co-location requirements inferred to then (i) decide whether a composition should be attempted or not and (ii) to select the providers' in the surroundings that maximise the reliability of a composition.

In addition, our work contributes a method to predict and react to changes in the environments. The speed of variation of signal attenuation for the elected service providers is continuously observed and then used to predict a "departure time", computed as the time it takes to reach a minimum accepted signal strength from the current signal strength at the recorded variation speed.

This work has been published in:

- L. Del Prete and L. Capra. "Reliable Discovery and Selection of Composite Services in Mobile Environments". In 12th IEEE International Enterprise Computing Conference (EDOC). Munich, Germany. September 2008.

### **(3) Architecture Framework**

Finally, our work contributes a flexible discovery framework that eases the reliable and personalised discovery of composite services in mobile environments.

We model Service Discovery as a filtering/ranking process that we implement through a Discovery Pipeline (i.e., a sequence of Filters run in turn). As a discovery request is issued, we input to the Discovery Pipeline the set of all the services (and their providers) available in the surroundings, hence we run each filter in the pipeline one after the other and obtain a ranked list of service instances (i.e., services and their providers).

Our work also contributes a middleware architecture that integrates and coordinates the selection, composition and execution of pervasive services in mobile environments, thus abstracting this complexity away from the developer. We decouple the main functionalities required for a service compositions and create a framework built around four main components: the Service Manager, the Service Analyser, the Service Discoverer and the Service Coordinator. The Service Manager component is the access point to the composition framework. It provides an interface to request services, leaving the invoking application unaware of whether the requested service is single or compound, and whether it resides

on the same device or it is accessible from other devices in the proximity (in particular, at a single hop distance).

This work has been published in:

- L. Del Prete and L. Capra. “MoSCA: Seamless Execution of Mobile Composite Services”. In 7th ACM Workshop on Adaptive and Reflective Middleware (ARM). Leuven, Belgium, December 2008;
- L. Del Prete and L. Capra. “MoSCA: Service Composition in Mobile environments”. In Middleware 2008, ACM/IFIP/USENIX 9th International Middleware. Leuven, Belgium. December 2008.

## 7.2 Critical Evaluation and Future Work

As we have pointed out in Chapter 1, the future we envision dictates new requirements to mobile computing. On one side, users will have access to a countless multitude of services and content sources impossible to know in its entirety. Therefore they will be not able, by themselves, to know which services are available and among them which ones they may like the most. They will need assistance. On the other side, users will consume composite services whose component providers will be relatively mobile and will be available just when co-located to their consumers. Still, users will expect a smooth user experience. Also, in order to boost the flourishing of applications built on pervasive services, the complexities of dealing with composite services in mobile environments should be abstracted away from developers.

Three major aims can thus be highlighted:

- (1) **Service and Content Abundance.** Promote services and content most appealing to users.
- (2) **Smooth experience.** Users should perceive pervasive services as if served by an always-available and unique service provider.
- (3) **Lightweight Middleware.** Provide a lightweight middleware to ease the development of applications built on pervasive services.

In this section we evaluate our work and suggest future work with respect to these criteria.

### (1) Service and Content Abundance

The Mobile Recommender and Composite Recommender we have developed provide recommendations on pervasive services with a precision and coverage comparable to that of centralised recommender systems in use today. We have thus provided a solution for Service and Content Abundance in mobile environments that is effective as much as the solutions adopted nowadays for traditional settings.

We argue, though, that still a number of issues exist when it comes to advise users.

First and foremost, we observe that the interests of a same user vary depending on her current context (e.g., at work, at home, commuting to work, commuting from work, etc.). Whilst this is true for mobile and desktop users alike, we remark that mobile devices assist their users along the entire day and almost wherever they go: not just when they are at work, not just when they are at home, but also when at a restaurant, in a coffee shop, commuting, at a party, at the airport, while reading or listening to music, etc. Also, mobile users are naturally multi-tasking and they seem to show a greater sense of urgency and need for action<sup>1</sup>. Hence, mobile users operate in a much greater variety and dynamicity of contexts and there seems to be a bigger opportunity for contextual information to be relevant and above all valuable. For these reasons, we believe that one of the most critical improvements for a Mobile Recommender is the ability to advice users based on their current (and/or future) context and underpinning intents.

Another interesting aspect of Mobile Recommenders is the data sparsity. Whilst this is a problem affecting both traditional and mobile recommenders, this acquires new connotations in the context of pervasive services. On one side, as observed in Chapter 2, mobile recommender systems will have access to just a portion of the global rating data. On the other side, we argue users will be consuming pervasive services while also attaining concurrent tasks, and so it will be unlikely for them to rate all the items experienced, thus causing even sparser rating matrices. A challenge then raises on understanding what a “non-vote” means and inferring ratings from users behaviours.

Finally, on a more pragmatic note, we observe that the Mobile Recommender proposed builds on two clusters: mass-like minded users and atypical users. As a future work, it would be interesting to investigate the impact on precision, coverage and overhead of clustering in general.

### (2) Smooth experience

The Mobility Predictor, the Semantic Reasoner and the Adaptive Binder provide together an effective and efficient method to optimise the ratio of compositions started and success-

---

<sup>1</sup>Source: The Mobile Movement Study, Google/Ipsos OTX MediaCT , Apr 2011

fully completed (e.g., above 90% success rate even for compositions that lapse for longer than 20 minutes). Hence, overall they provide an experience that in most cases users should perceive as if served by an always-available and unique service provider.

Although our framework allows to add QoS reasoning to the selection of service providers and we argue that, while important, QoS reasoning must come after co-location reasoning, we also remark that QoS will be fundamental in practice. The challenge is then to understand in what order to apply QoS filters and with what consequences on the user experience.

Also, we observe that QoS will acquire new meanings in mobile environments as it will be dependent on qualities that will vary as the runtime context, the consumer-provider pair, the consumer and provider relative mobility vary. For these reasons, as future work we suggest augmenting the methods proposed with QoS reasoning and to do so looking at relative and/or contextualised QoS (e.g., as recorded by different consumers in different contexts).

### **(3) Lightweight Middleware**

The composition framework we have delivered – MoSCA – is at the same time lightweight, flexible and easy to use. In fact, as demonstrated in Chapter 3 and in Chapter 5, all methods proposed are lightweight in terms of processing and primary memory allocation, and MoSCA footprint is about 51 kB (compressed). Also, as discussed in Section 6.1 and Section 6.2, MoSCA allows to plug-in new components and discovery methods leaving the architecture unchanged. Finally, as shown in Section 6.5, MoSCA abstracts away from developers the complexity of service compositions by providing easy-to-use interfaces. Overall, we have provided a lightweight framework that eases the development and use of composite services in mobile environments.

On the other hand, the actual implementation of the framework builds on very simple implementations of the Service Analyser and the Service Coordinator. So to be used in real scenarios, MoSCA would need to be integrated into a rich and comprehensive Service Analyser. Consequently we propose, as future work, to enrich MoSCA with a rich Service Analyser and investigate methods that cater for users' preferences and the services available at the time of the analysis. Also, we argue that, although we have analysed the resource overhead of the methods and data structure proposed, we have not carried out a real analysis of MoSCA suitability to mobile users. For these reasons, as future work we suggest an evaluation of MoSCA based on actual deployments to assess actual resource consumption (e.g., battery) and overhead, as well as the actual user experience.



# Bibliography

- [eic, 1997] (1997). EICC Homepage. <http://www.eicc.co.uk/>.
- [rob, 2001] (2001). The Web Robots Pages. <http://www.robotstxt.org/>.
- [eic, 2003] (2003). Terry Farrell and Partners, Projects, Edinburgh. <http://www.terryfarrell.co.uk/>.
- [mit, 2005] (2005). The Reality Mining Project. <http://reality.media.mit.edu/>.
- [net, 2006] (2006). Netflix Prize. <http://www.netflixprize.com/>.
- [sit, 2006] (2006). sitemaps.org. <http://www.sitemaps.org/>.
- [S3, 2007] (2007). S3 Contest. <http://www-ags.dfki.uni-sb.de/klusuch/s3/index.html>.
- [SWS, 2008] (2008). Semantic Web Services Challenge. <http://sws-challenge.org/>.
- [mah, 2010] (2010). Apache Mahout. <http://lucene.apache.org/mahout/>.
- [Abowd et al., 1999] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. (1999). Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK. Springer-Verlag.
- [Adomavicius and Tuzhilin, 2005] Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749.
- [Akkiraju et al., 2005] Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.-T., Sheth, A., and Verma, K. (2005). Web Service Semantics - WSDL-S. <http://www.w3.org/Submission/WSDL-S/>.
- [Alliance, 2001] Alliance, O. M. (2001). Wireless Application Protocol. <http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html>.
- [Allsopp, 2007] Allsopp, J. (2007). *Microformats - Empowering Your Markup for Web 2.0*. Springer.

- [Arkin et al., 2002] Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacs-Nagy, P., Trickovic, I., and Zimek, S. (2002). Web Service Choreography Interface (WSCI) Version 1.1. <http://www.w3.org/TR/WSCI/>.
- [Ashton, 2009] Ashton, K. (2009). That 'Internet of Things' Thing. *RFID Journal*.
- [ASL, 2011] ASL, W. (2011). Wac 2.0 specification. WAC 2.0 proposed release, WAC ASL. <http://www.wacapps.net/web/portal/wac-2.0-spec>.
- [Baccigalupo and Plaza, 2006] Baccigalupo, C. and Plaza, E. (2006). Case-based sequential ordering of songs for playlist recommendation. In *In Proc. of the ECCBR 06 Conference*.
- [Balabanovic and Shoham, 1997] Balabanovic, M. and Shoham, Y. (1997). Fab: Content-based, collaborative recommendation. In *Communications of the ACM*.
- [Bardram, 2004] Bardram, J. E. (2004). Applications of context-aware computing in hospital work: examples and design principles. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 1574–1579, New York, NY, USA. ACM.
- [Basu et al., 2003] Basu, P., Ke, W., and Little, T. D. C. (2003). Dynamic task-based anycasting in mobile ad hoc networks. *Mobile Networks and Applications*, 8:593–612.
- [Basu Roy et al., 2010] Basu Roy, S., Amer-Yahia, S., Chawla, A., Das, G., and Yu, C. (2010). Constructing and exploring composite items. In *Proceedings of the 2010 international conference on Management of data, SIGMOD '10*, pages 843–854, New York, NY, USA. ACM.
- [Battle and Benson, 2008] Battle, R. and Benson, E. (2008). Bridging the semantic web and web 2.0 with representational state transfer (REST). *Journal of Web Semantics*, 6(1):61–69.
- [Bennett, 2006] Bennett, J. (2006). Independent Study Report: A survey of SOM and Recommender techniques. In *A Research Report, Golisano College of Computing and Information Science Rochester Institute of Technology*, pages 553.
- [Berglund et al., 2010] Berglund, A., Boag, S., Chamberlin, D., Fernandez, M. F., Kay, M., Robie, J., and Simon, J. (2010). XML Path Language (XPath) 2.0. <http://www.w3.org/TR/xpath20/>.
- [Berners-Lee, 2007] Berners-Lee, T. (2007). Giant Global Graph — Decentralized Information Group (DIG) Breadcrumbs. <http://dig.csail.mit.edu/breadcrumbs/node/215>.
- [Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.

- [Birbeck and Adida, 2008] Birbeck, M. and Adida, B. (2008). RDFa primer. W3C note, W3C. <http://www.w3.org/TR/2008/NOTE-xhtml-rdfa-primer-20081014/>.
- [Box et al., 2000] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. (2000). SOAP: Simple Object Access Protocol. <http://www.w3.org/TR/SOAP/>.
- [Boxall and Purcell, 2003] Boxall, P. and Purcell, J. (2003). *Strategy and human resource management*. Management, work and organisations. Palgrave Macmillan, Basingstoke [u.a.].
- [Bray et al., 2008] Bray, D. A., Laubacher, R., and Malone, T. W. (2008). Collective Intelligence: Promoting Diversity, Crowd Performance Algorithms, and Better Decision Outcomes. *Social Science Research Network Working Paper Series*.
- [Bray et al., 1998] Bray, T., Paoli, J., and Sperberg-McQueen, C. (1998). Extensible markup language (XML) 1.0. Technical report, W3C.
- [Breese et al., 1998] Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52. Morgan Kaufmann.
- [Brogi and Popescu, 2006] Brogi, A. and Popescu, R. (2006). Contract-based Service Aggregation. Technical Report TR-06-12, Dept. of Computer Science, University of Pisa.
- [Bronstead et al., 2007] Bronstead, J., Hansen, K. M., and Ingstrup, M. (2007). A survey of service composition mechanisms in ubiquitous computing. In *Second Workshop on Requirements and Solutions for Pervasive Software Infrastructures, UbiComp 2007 Workshops Proceedings*.
- [Burke, 2002] Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370.
- [Candillier et al., 2008a] Candillier, L., Meyer, F., and Fessant, F. (2008a). Designing specific weighted similarity measures to improve collaborative filtering systems. In *8th Industrial Conference on Data Mining (ICDM'2008)*, LNCS. Springer Verlag.
- [Candillier et al., 2008b] Candillier, L., Meyer, F., and Fessant, F. (2008b). Designing specific weighted similarity measures to improve collaborative filtering systems. *Advances in Data Mining. Medical Applications, E-Commerce, Marketing, and Theoretical Aspects*, pages 242–255.
- [Capra, 2003] Capra, L. (2003). Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Trans. Softw. Eng.*, 29(10):929–945. Member-Emmerich, Wolfgang and Member-Mascolo, Cecilia.

- [Capra et al., 2005] Capra, L., Zachariadis, S., and Mascolo, C. (2005). Q-CAD: QoS and Context Aware Discovery Framework for Adaptive Mobile Systems. In *IEEE International Conference on Pervasive Services*, Santorini, Greece.
- [Castagnos and Boyer, 2007] Castagnos, S. and Boyer, A. (2007). Modeling preferences in a distributed recommender system. In *UM '07: Proceedings of the 11th international conference on User Modeling*, pages 400–404, Berlin, Heidelberg. Springer-Verlag.
- [Chakraborty et al., 2005] Chakraborty, D., Joshi, A., Finin, T., and Yesha, Y. (2005). Service Composition for Mobile Environments. *Journal on Mobile Networking and Applications, Special Issue on Mobile Services*, 10(4):435–451.
- [Chakraborty et al., 2006] Chakraborty, D., Joshi, A., and Yesha, Y. (2006). Integrating Service Discovery with Routing and Session Management for Ad-Hoc Networks. *Elsevier Journal of Ad-Hoc Networks*, 4(2):204–224.
- [Chakraborty et al., 2004] Chakraborty, D., Joshi, A., Yesha, Y., and Finin, T. (2004). Service Composition for Mobile Environments. *Journal on Mobile Networking and Applications, Special issue on Mobile Services*.
- [Christensen et al., 2001] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web Services Description Language (WSDL) 1.1. <http://www.w3.org/TR/wsdl>.
- [Cibran and Verheecke, 2005] Cibran, M. and Verheecke, B. (2005). Dynamic Business Rules for Web Service Composition. In *Proceedings of the Second Dynamic Aspects Workshop*.
- [Clark and DeRose, 1999] Clark, J. and DeRose, S. (1999). XML Path Language (XPath). <http://www.w3.org/TR/xpath/>.
- [Clinton, 2009] Clinton, D. (2009). Specification OpenSearch 1.1 Draft 4. <http://www.opensearch.org/Specifications/OpenSearch/1.1>.
- [Coalition, 2003] Coalition, B. (2003). Business process execution language for web services (BPEL4WS) version 1.1. <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf>.
- [Coalition, 2000] Coalition, U. (2000). The UDDI technical white paper. <http://www.uddi.org/>.
- [Cooper and Craig, 2009] Cooper, M. and Craig, J. (2009). Accessible Rich Internet Applications (WAI-ARIA) 1.0. W3C working draft, W3C. <http://www.w3.org/TR/2009/WD-wai-aria-20091215/>.

- [Coster and Svensson, 2005] Coster, R. and Svensson, M. (2005). Incremental collaborative filtering for mobile devices. In *Proceedings of Proceedings of the 2005 ACM symposium on Applied computing*, page 5, Santa Fe, New Mexico, USA.
- [Crockford, 2006] Crockford, D. (2006). RFC4627: JavaScript Object Notation. <http://www.ietf.org/rfc/rfc4627.txt>.
- [Dahlem et al., 2004] Dahlem, D., Bychkov, Y., Kawasme, L., and Jahnke, J. H. (2004). Towards context oriented web services for smart personal object technologies (cowspots). In *OOPSLA Workshop on Reference Architectures and Patterns for Pervasive Computing*.
- [de Spindler et al., 2006] de Spindler, A., Spindler, R. D., Norrie, M. C., Grossniklaus, M., and Signer, B. (2006). Spatio-Temporal Proximity as a Basis for Collaborative Filtering in Mobile Environments. In *UMICS*.
- [Del-Prete and Capra, 2008a] Del-Prete, L. and Capra, L. (2008a). Mosca: seamless execution of mobile composite services. In *ARM'08*, pages 5–10.
- [Del-Prete and Capra, 2008b] Del-Prete, L. and Capra, L. (2008b). Mosca: service composition in mobile environments. In *Middleware (Companion)'08*, pages 87–89.
- [Del-Prete and Capra, 2008c] Del-Prete, L. and Capra, L. (2008c). Reliable discovery and selection of composite services in mobile environments. In *EDOC'08*, pages 171–180.
- [Del-Prete and Capra, 2010] Del-Prete, L. and Capra, L. (2010). differs: A mobile recommender service. In *Mobile Data Management'10*, pages 21–26.
- [Dogac et al., 2003] Dogac, A., Kabak, Y., and Laleci, G. (2003). A semantic-based web service composition facility for ebxml registries. In *9th International Conference of Concurrent Enterprising*.
- [Driscoll and Mensch, 2009] Driscoll and Mensch (2009). OASIS Devices Profile for Web Services (DPWS) Version 1.1. <http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01>.
- [Eagle and Pentland, 2006] Eagle, N. and Pentland, A. (2006). Reality Mining: Sensing Complex Social Systems. *Personal and Ubiquitous Computing*, 10(4).
- [Eagle and Pentland, 2007] Eagle, N. and Pentland, A. (2007). Eigenbehaviors: Identifying Structure in Routine. *Behavioral Ecology and Sociobiology*.
- [Fielding et al., 1999] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1.
- [Fielding, 2000] Fielding, R. T. (2000). *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine.

- [Friederike Klan, 2008] Friederike Klan, Birgitta König-Ries, B. (2008). A personalized approach to experience-aware service ranking and selection. In *Proceedings of the 2nd international conference on Scalable Uncertainty Management (SUM2008)*.
- [Gao et al., 2007] Gao, Z., Liu, S., Ji, M., Zhao, J., and Liang, L. (2007). Ftscp: An efficient distributed fault-tolerant service composition protocol for manets. In *HPCC*, pages 797–808.
- [Garlan et al., 2002] Garlan, D., Siewiorek, D. P., and Steenkiste, P. (2002). Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1:22–31.
- [Gelernter, 1985] Gelernter, D. (1985). Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7:80–112.
- [Gershenfeld et al., 2004] Gershenfeld, N., Krikorian, R., and Cohen, D. (2004). The Internet of things. *Scientific American*, 291(4):76–81.
- [Ghazanfar and Prgel-Bennett, 2010] Ghazanfar, M. A. and Prgel-Bennett, A. (2010). Novel significance weighting schemes for collaborative filtering: Generating improved recommendations in sparse environments. In Stahlbock, R., Crone, S. F., Abou-Nasr, M., Arabnia, H. R., Kourentzes, N., Lenca, P., Lippe, W.-M., and Weiss, G. M., editors, *DMIN*, pages 334–342. CSREA Press.
- [Goldberg et al., 1992] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35:61–70.
- [Gratz and Andronache, 2008] Gratz, P. and Andronache, A. (2008). Optimising distributed collaborative filtering in mobile networks. In *ECMS’08*.
- [Gratz et al., 2008] Gratz, P., Andronache, A., and Rothkugel, S. (2008). Ad hoc collaborative filtering for mobile networks. In *SUTC ’08: Proceedings of the 2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008)*, pages 355–360, Washington, DC, USA. IEEE Computer Society.
- [Hadley, 2009] Hadley, M. (2009). Web Application Description Language (WADL). <http://wadl.dev.java.net/wadl20061109.pdf>.
- [Han et al., 2004] Han, P., Xie, B., Yang, F., and Sheng, R. (2004). A scalable p2p recommender system based on distributed collaborative filtering. *Expert systems with applications*.
- [Handorean et al., 2006] Handorean, R., Sen, R., Hackmann, G., and Roman, G.-C. (2006). Supporting Predictable Service Provision in MANETs via Context-Aware Session Management. *International Journal of Web Services Research*, 3(3):1–26.

- [Hansen and Golbeck, 2009] Hansen, D. L. and Golbeck, J. (2009). Mixing it up: recommending collections of items. In Jr., D. R. O., Arthur, R. B., Hinckley, K., Morris, M. R., Hudson, S. E., and Greenberg, S., editors, *CHI*, pages 1217–1226. ACM.
- [Hayes and Cunningham, 2000] Hayes, C. and Cunningham, P. (2000). Smart radio - building music radio on the fly. In *Expert Systems 2000*, pages 2–6. ACM Press.
- [Herlocker et al., 1999] Herlocker, J., Konstan, J., Borchers, A., and Riedl, J. (1999). An Algorithmic Framework for Performing Collaborative Filtering. In *ACM SIGIR*, Berkley, CA, USA. ACM Press.
- [Herlocker et al., 2004] Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53.
- [Hickson, 2011] Hickson, I. (2011). A vocabulary and associated APIs for HTML and XHTML. W3C note, W3C. <http://www.w3.org/TR/html5/>.
- [Hong et al., 2008] Hong, S., Rhee, I., Kim, S. J., Lee, K., and Chong, S. (2008). Routing performance analysis of human-driven delay tolerant networks using the truncated levy walk model. In *Proc. of the 1st ACM Workshop on Mobility Models*, pages 25–32.
- [Huttner, 2009] Huttner, J. (2009). From Tapestry to SVD. A Survey of the Algorithms That Power Recommender Systems. In *Haverford Computer Science Department*.
- [International, 1999] International, E. C. M. A. (1999). *ECMA-262: ECMAScript Language Specification*. ECMA (European Association for Standardizing Information and Communication Systems), Geneva, Switzerland, third edition.
- [Ippolito, 2005] Ippolito, B. (2005). Remote JSON – SONP. <http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/>.
- [Jameson and Smyth, 2007] Jameson, A. and Smyth, B. (2007). Recommendation to groups. In *The Adaptive Web: Methods and Strategies of Web Personalization*, chapter 20, pages 596–627. Springer.
- [Julien, 2006] Julien, C. (2006). Adaptive Preference Specification for Application Sessions. In *Proceedings of the 4th International of Service-Oriented Computing*, pages 78–89, Chicago, IL, USA.
- [Julien and Roman, 2002] Julien, C. and Roman, G.-C. (2002). Egocentric context-aware programming in ad hoc mobile environments. In *SIGSOFT '02/FSE-10: Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering*, pages 21–30, New York, NY, USA. ACM.
- [Julien and Stovall, 2006] Julien, C. and Stovall, D. (2006). Enabling Ubiquitous Coordination Using Application Sessions. In *Proceedings of the 8th International Conference on Coordination Models and Languages*, pages 130–144, Bologna, Italy.

- [Khabbaz et al., 2011] Khabbaz, M., Xie, M., and Lakshmanan, L. V. S. (2011). Toprecs: Pushing the envelope on recommender systems. *IEEE Data Eng. Bull.*, pages 61–68.
- [Klusch et al., 2006] Klusch, M., Fries, B., and Sycara, K. (2006). Automated semantic web service discovery with owls-mx. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, New York, NY, USA. ACM.
- [Kotz and Henderson, 2005] Kotz, D. and Henderson, T. (2005). Community Resource for Archiving Wireless Data At Dartmouth. <http://crawdad.cs.dartmouth.edu/>.
- [Krogh and Vedelsby, 1995] Krogh, A. and Vedelsby, J. (1995). Neural Network Ensembles, Cross Validation, and Active Learning. In *Advances in Neural Information Processing Systems*, pages 231–238. MIT Press.
- [Kropp et al., 2003] Kropp, A., Leue, C., and Thompson, R. (2003). Web Services for Remote Portlets Specification (WSRP). <http://www.oasis-open.org>.
- [Küster et al., 2007] Küster, U., Lausen, H., and König-Ries, B. (2007). Evaluation of semantic service discovery - a survey and directions for future research. In *Post-Proceedings of the 2nd Workshop on Emerging Web Services Technology (WEWST07) in conjunction with the 5th IEEE European Conference on Web Services (ECOWS07)*.
- [Lassila and Swick, 1999] Lassila, O. and Swick, R. R. (1999). Resource Description Framework (RDF). Model and Syntax Specification. Technical report, W3C.
- [Leamer, 1978] Leamer, E. (1978). *Specification Searches*. Wiley series in probability and mathematical statistics. Wiley, New York, NY.
- [Leino and Räihä, 2007] Leino, J. and Räihä, K.-J. (2007). Case amazon: ratings and reviews as part of recommendations. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 137–140, New York, NY, USA. ACM.
- [Libby, 1999] Libby, D. (1999). RSS 0.91 Spec, revision 3. <http://web.archive.org/web/20001204093600/my.netscape.com/publish/formats/rss-spec-0.91.html>.
- [Linden et al., 2003] Linden, G., Smith, B., and York, J. (2003). Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80.
- [Liu and Issarny, 2004] Liu, J. and Issarny, V. (2004). QoS-Aware Service Location in Mobile Ad-Hoc Networks. In *Proceedings of the IEEE International Conference on Mobile Data Management*, pages 224–235.
- [Manouselis and Costopoulou, 2007] Manouselis, N. and Costopoulou, C. (2007). Analysis and classification of multi-criteria recommender systems. *World Wide Web*, 10(4):415–441.

- [Martin et al., 2004] Martin, D., Burstein, M., Hobbs, E., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. (2004). OWL-S: Semantic Markup for Web Services. Technical report, W3C.
- [Martin and Domingue, 2007a] Martin, D. and Domingue, J. (2007a). Semantic web services, part 1. *Intelligent Systems*, 22(5):12–17.
- [Martin and Domingue, 2007b] Martin, D. and Domingue, J. (2007b). Semantic web services, part 2. *Intelligent Systems*, 22(6):8–15.
- [Mettala, 1999] Mettala, R. (1999). Bluetooth protocol architecture. *Bluetooth Special Interest Group SIG Version*, pages 1–20.
- [Miller et al., 2003] Miller, B. N., Albert, I., Lam, S. K., Konstan, J. A., and Riedl, J. (2003). Movielens unplugged: Experiences with an occasionally connected recommender system. In *InProceedings of ACM 2003 International Conference on Intelligent User Interfaces (IUI'03) (Accepted Poster)*, pages 263–266. ACM Press.
- [Miller et al., 2004] Miller, B. N., Konstan, J. A., and Riedl, J. (2004). Pocketlens: Toward a personal recommender system. *ACM Transactions on Information Systems (TOIS)*, 22(3):437–476.
- [Mokhtar et al., 2006] Mokhtar, S. B., Kaul, A., Georgantas, N., and Issarny, V. (2006). Efficient Semantic Service Discovery in Pervasive Computing Environments. In *Proceedings of the ACM/IFIP/USENIX 7th International Middleware Conference*, Melbourne.
- [Moloney, 2005] Moloney, S. (2005). Simulation of a distributed recommendation system for pervasive networks. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 1577–1581.
- [Murphy et al., 2001] Murphy, A. L., Picco, G. P., and Roman, G.-C. (2001). Lime: A middleware for physical and logical mobility. *Distributed Computing Systems, International Conference on*, 0:0524.
- [Nottingham and Sayre, 2005] Nottingham, M. and Sayre, R. (2005). The Atom Syndication Format. <http://www.ietf.org/rfc/rfc4287>.
- [Orchestra, 2007] Orchestra, T. C. (2007). To build a home. In *Ma Fleur*. Domino US.
- [Pakkala et al., 2004] Pakkala, D., Sihvonen, M., and Latvakoski, J. (2004). Towards a distributed service platform for extending enterprise applications to the mobile computing domain. In *International Conference on Internet Computing*, pages 416–422.
- [Paradiso and Starner, 2005] Paradiso, J. A. and Starner, T. (2005). Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Computing*, 4(1):18–27.

- [Patil et al., 2004a] Patil, A., Oundhakar, S., Sheth, A., and Verma, K. (2004a). METEOR-S Web service Annotation Framework. In *Proceedings of the Thirteenth International World Wide Web Conference*.
- [Patil et al., 2004b] Patil, A. A., Oundhakar, S. A., Sheth, A. P., and Verma, K. (2004b). Meteor-s web service annotation framework. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 553–562.
- [Paulos and Goodman, 2004] Paulos, E. and Goodman, E. (2004). The familiar stranger: anxiety, comfort, and play in public places. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 223–230. ACM.
- [Penn and Zalesne, 2007] Penn, M. and Zalesne, K. (2007). *Microtrends: the small forces behind tomorrow's big changes*. Twelve.
- [Picco et al., 1999] Picco, G. P., Murphy, A. L., and Roman, G.-C. (1999). Lime: Linda meets mobility. In *Proceedings of the 21st international conference on Software engineering*, ICSE '99, pages 368–377, New York, NY, USA. ACM.
- [Rao and Su, 2005] Rao, J. and Su, X. (2005). A survey of automated web service composition methods. In *LNCS*, volume 3387/2005, pages 43–54. Springer.
- [Resnick et al., 1994] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In Furuta, R. and Neuwirth, C. M., editors, *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM Press.
- [Resnick and Varian, 1997] Resnick, P. and Varian, H. R. (1997). Recommender Systems. *Communications of the ACM*, 40(3):56–58.
- [Ricci and Nguyen, 2005] Ricci, F. and Nguyen, Q. N. (2005). Critique-based mobile recommender systems. *OEGAI Journal*, 24(4).
- [Román et al., 2002] Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., and Nahrstedt, K. (2002). A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83.
- [Ruffo and Schifanella, 2009] Ruffo, G. and Schifanella, R. (2009). A peer-to-peer recommender system based on spontaneous affinities. *ACM Transactions on Internet Technology (TOIT)*, 9(1):1–34.
- [Ruffo et al., 2006] Ruffo, G., Schifanella, R., and Ghiringhello, E. (2006). A decentralized recommendation system based on self-organizing partnerships. In *Networking*, pages 618–629.
- [Rukzio et al., 2004] Rukzio, E., Schmidt, A., and Hussmann, H. (2004). Privacy-enhanced intelligent automatic form filling for context-aware services on mobile devices.

- In *Workshop Artificial Intelligence in Mobile Systems 2004 (AIMS 2004) in conjunction with UbiComp 2004*.
- [Sailhan and Issarny, 2005] Sailhan, F. and Issarny, V. (2005). Scalable Service Discovery in MANET. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications*, pages 235–244, Hawaii, USA.
- [Sarwar et al., 2001] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *10th International World Wide Web Conference*. Association of Computing Machinery, Association of Computing Machinery.
- [Sarwar et al., 2000] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. T. (2000). Application of dimensionality reduction in recommender system - a case study. In *ACM WebKDD Workshop*, pages 285–295.
- [Sarwar et al., 1998] Sarwar, B. M., Konstan, J. A., Borchers, A., Herlocker, J., Miller, B., and Riedl, J. (1998). Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work, CSCW '98*, pages 345–354. ACM.
- [Schifanella et al., 2008] Schifanella, R., Panisson, A., Gena, C., and Ruffo, G. (2008). Mobhinter: epidemic collaborative filtering and self-organization in mobile ad-hoc networks. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 27–34, New York, NY, USA. ACM.
- [Schilit and Theimer, 1994] Schilit, B. and Theimer, M. (1994). Disseminating active map information to mobile hosts. *IEEE Network*, 8:22–32.
- [Schreiber and Dean, 2004] Schreiber, G. and Dean, M. (2004). OWL web ontology language reference. W3C recommendation, W3C. <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>.
- [Sean, 2005] Sean, O. (2005). Taste. Collaborative Filtering for Java. <http://taste.sourceforge.net/>.
- [Shardanand and Maes, 1995] Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating ‘word of mouth’. In *In Proc. of the Conf. on Human Factors in Computing Systems*.
- [Svensson Fors et al., 2009] Svensson Fors, D., Magnusson, B., Gestegård Robertz, S., Hedin, G., and Nilsson-Nyman, E. (2009). Ad-hoc composition of pervasive services in the palcom architecture. In *ICPS '09: Proceedings of the 2009 international conference on Pervasive services*, pages 83–92, New York, NY, USA. ACM.

- [Tecnologica et al., 2001] Tecnologica, S. E., Aguzzoli, S., Avesani, P., Massa, P., Aguzzoli, S., Avesani, P., and Massa, P. (2001). Compositional cbr via collaborative filtering. In *ICCBR '01 Workshop on CBR in Electronic Commerce*.
- [Terveen and Hill, 2001] Terveen, L. and Hill, W. (2001). Beyond recommender systems: Helping people help each other. In *HCI in the New Millennium*, pages 487–509. Addison-Wesley.
- [Thomas et al., 2004] Thomas, E., Zhang, Y., Wright, J., Mckenzie, C., and Sleeman, P. D. (2004). Microsoft Corporation: Universal Plug-and-Play (UPnP). <http://www.upnp.org>.
- [Tveit, 2001] Tveit, A. (2001). Peer-to-peer based recommendations for mobile commerce. In *WMC '01: Proceedings of the 1st international workshop on Mobile commerce*, pages 26–29, New York, NY, USA. ACM Press.
- [Urbietia et al., 2008] Urbietia, A., Barrutieta, G., Parra, J., and Uribarren, A. (2008). A survey of dynamic service composition approaches for ambient systems. In *SOMITAS '08: Proceedings of the 2008 Ambi-Sys workshop on Software Organisation and MonIToring of Ambient Systems*, pages 1–8, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [van der Aalst et al., 2000] van der Aalst, W., Barros, A., ter Hofstede, A., Kiepuszewski, B., and B (2000). Advanced workflow patterns. In *7th International Conference on Cooperative Information Systems (CoopIS 2000)*.
- [van der Aalst and ter Hofstede, 2005] van der Aalst, W. M. P. and ter Hofstede, A. H. M. (2005). YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275.
- [van Setten et al., 2004] van Setten, M., Pokraev, S., and Koolwaaij, J. (2004). Context-aware recommendations in the mobile tourist application compass. *Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 235–244.
- [Vukovic et al., 2007] Vukovic, M., Kotsovinos, E., and Robinson, P. (2007). Application development powered by rapid, on-demand service composition. In *SOCA '07*, pages 88–98.
- [Waldo, 1999] Waldo, J. (1999). The Jini Architecture for Network-Centric Computing. *Commun. ACM*, 42(7):76–82.
- [Wang et al., 2009] Wang, J., Peng, J., and Cao, X. (2009). A Distributed Collaborative Filtering Recommendation Model for P2P Networks. In *Collaborative Computing: Networking, Applications and Worksharing*, pages 1–10.

- [Wang et al., 2006] Wang, J., Pouwelse, J., Lagendijk, R. L., and Reinders, M. J. T. (2006). Distributed collaborative filtering for peer-to-peer file sharing systems. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1026–1030, New York, NY, USA. ACM Press.
- [Wang et al., 2003] Wang, Q., Yuan, Y., Zhou, J., and Zhou, A. (2003). *Advances in Web-Age Information Management*. Springer.
- [WAP Forum, 1999] WAP Forum (1999). WAP WML. <http://www.wapforum.org/what/technical/SPEC-WML-19990616.pdf>.
- [Weiss and Edwards, 2005] Weiss, D. J. and Edwards, W. (2005). Extracting individual contributions to a team’s performance. In *Teorie e Modelli*.
- [Williams et al., 2005] Williams, A., Padmanabhan, A., and Blake, M. (2005). Experimentation with Local Consensus Ontologies with Implications for Automated Service Composition. *IEEE Transactions on Knowledge and Data Engineering*, 17(7):969–981.
- [Williams et al., 2003] Williams, A., Padmanabhan, A., and Blake, M. B. (2003). Local consensus ontologies for B2B oriented service composition. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, New York, NY, USA.
- [WSCDL Coalition, 2005] WSCDL Coalition (2005). Web Services Choreography Description Language Version 1.0. <http://www.w3.org/TR/ws-cdl-10/>.
- [Xie et al., 2010] Xie, M., Lakshmanan, L. V., and Wood, P. T. (2010). Breaking out of the box of recommendations: from items to packages. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys ’10, pages 151–158, New York, NY, USA. ACM.
- [Ziegler, 2005] Ziegler, C. N. (2005). *Towards Decentralized Recommender Systems*. PhD thesis, Albert-Ludwigs-Universität Freiburg, Germany, Freiburg i.Br., Germany.
- [Zobel and Moffat, 2006] Zobel, J. and Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys*, 38:2006.