Creating & Enabling the Useful Service Discovery Experience

The Perfect Recommendation Does Not Exist

by

Magnus Ingmarsson

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE

Linköping University

INSTITUTE OF TECHNOLOGY

Human Centered Systems
Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden

Linköping 2013
Creating & Enabling the Useful Service Discovery Experience

The Perfect Recommendation Does Not Exist

©Magnus Ingmarsson, 2013

Cover concept: Magnus Ingmarsson
Picture: Charlene Sprong
Illustration/Design: Per Lagman

Printed by LiU-Tryck, Linköping, Sweden, 2013

ISSN 0345-7524
URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-88435
To the person who seeks knowledge
Abstract

We are rapidly entering a world with an immense amount of services and devices available to humans and machines. This is a promising future, however there are at least two major challenges for using these services and devices: (1) they have to be found and (2) after being found, they have to be selected amongst. A significant difficulty lies in not only finding most available services, but presenting the most useful ones. In most cases, there may be too many found services and devices to select from.

Service discovery needs to become more aimed towards humans and less towards machines. The service discovery challenge is especially prevalent in ubiquitous computing. In particular, service and device flux, human overloading, and service relevance are crucial. This thesis addresses the quality of use of services and devices, by introducing a sophisticated discovery model through the use of new layers in service discovery. This model allows use of services and devices when current automated service discovery and selection would be impractical by providing service suggestions based on user activities, domain knowledge, and world knowledge. To explore what happens when such a system is in place, a wizard of oz study was conducted in a command and control setting.

To address service discovery in ubiquitous computing new layers and a test platform were developed together with a method for developing and evaluating service discovery systems. The first layer, which we call the Enhanced Traditional Layer (ETL), was studied by developing the ODEN system and including the ETL within it. ODEN extends the traditional, technical service discovery layer by introducing ontology-based semantics and reasoning engines. The second layer, the Relevant Service Discovery Layer, was explored by incorporating it into the MAGUBI system. MAGUBI addresses the human aspects in the challenge of relevant service discovery by employing common-sense models of user activities, domain knowledge, and world knowledge in combination with rule engines.

The RESPONSORIA system provides a web-based evaluation platform with a desktop look and feel. This system explores service discovery in a service-oriented architecture setting. RESPONSORIA addresses a command and control scenario for rescue services where multiple actors and organizations work together at a municipal level. RESPONSORIA was the basis for the wizard of oz evaluation employing rescue services professionals. The result highlighted the importance of service naming and presentation to the user. Furthermore, there is disagreement among users regarding the optimal service recommendation, but the results indicated that good recommendations are valuable and the system can be seen as a partner.

This work was supported by Vinnova under project number 2002-00907, the Swedish Research Council under project number 621-2003-2991, and also by grants from the Swedish Emergency Management Agency (KBM), which is gratefully acknowledged.

URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-88435
Populärvetenskaplig sammanfattning

Vi rör oss snabbt in i en värld med en enorm mängd tjänster och enheter som finns tillgängliga för människor och maskiner. Detta är en lovande framtid, men det finns åtminstone två stora utmaningar för att använda dessa tjänster och enheter: (1) de måste hittas och (2) rätt tjänst/enhet måste väljas. En betydande svårighet ligger i att, inte bara finna de mest lättillgängliga tjänsterna och enheterna, men också att presentera de mest användbara sådana. I de flesta fall kan det vara för många tjänster och enheter som hittas för att kunna välja mellan.

Upptäckten av tjänster och enheter behöver bli mer anpassad till människor och mindre till maskiner. Denna utmaning är särskilt framträdande i desktopmetaforens efterföljare Ubiquitous Computing (Det vill säga en form av interaktion med datorer som blivit integrerad i aktiviteter och objekt i omgivningen.) Framförallt tjänster och enheters uppdykande och försvinnande, mänsklig överbelastning och tjänstens relevans är avgörande utmaningar. Denna avhandling behandlar kvaliteten på användningen av tjänster och enheter, genom att införa en sofistikerad upptäcktsmodell med hjälp av nya lager i tjänsteupptäcktsprocessen. Denna modell tillåter användning av tjänster och enheter när nuvarande upptäcktsprocess och urval av dessa skulle vara opraktiskt, genom att ge förslag baserat på användarnas aktiviteter, domänkunskap och omvärldskunskap. För att utforska vad som händer när ett sådant system är på plats, gjordes ett så kallat Wizard of Oz experiment i ledningscentralen på en brandstation. (Ett Wizard Of Oz experiment är ett experiment där användaren tror att de interagerar med en dator, men i själva verket är det en människa som agerar dator.)

För att hantera tjänste- och enhetsupptäckt i Ubiquitous Computing utvecklades nya lager och en testplattform tillsammans med en metod för att utveckla och utvärdera system för tjänste- och enhetsupptäckt. Det första lagret, som vi kallar Förbättrat Traditionellt Lager (FTL), studerades genom att utveckla ODEN och inkludera FTL i den. ODEN utökar det traditionella, datororienterade tjänste- och enhetsupptäcktslagret genom att införa en ontologibaserad semantik och en logisk reglering. Det andra skiktet, som vi kallar Relevant Tjänst Lager, undersöks genom att införliva det i systemet MAGUBI. MAGUBI tar sig an de mänskliga aspekterna i den utmaning som vi benämner relevant tjänste- och enhetsupptäckt, genom att använda modeller av användarnas aktiviteter, domänkunskap och kunskap om världen i kombination med regelmotorer.

Acknowledgements

The following people have contributed to this thesis in significant ways. My first thought of gratitude and thanks goes to Sture Hägglund, who got me started and who has never stopped supporting me since. I am also grateful to my advisor Henrik Eriksson, who is the kindest advisor a Ph.D. Candidate could have. Someone else who was also there from the beginning and deserves a bow is Vivian Vimarlund, who has acted as secondary advisor on this thesis. My closest colleagues and good friends Anders Fröberg, Ola Leifler, David Dinka, Magnus Bång, and Erik Berglund have all contributed to this thesis by providing stimulating discussions. The rest of my current and past colleagues, who created a warm social and academic life are not forgotten, even though the space here is not large enough to name them all. Ulf Kronqvist and Christer Carlsson deserves a special mention for their assistance in the experiments at the Lambohov fire station. To conclude the work related section of these acknowledgements, I sincerely thank the administrative and technical personnel, for all their help in sorting out the details of the Ph.D. candidate work life.

A big smile to my mates Morgan Eklöf, Mats Ekberg, and Håkan Törnroth for all the things we did when I was not writing this book. A special thanks to Eva Linde for brightening the day. My family and friends have been a never ending source of cheerful support and encouragement. Thank you mom and dad!

Finally, I am grateful to all the scientists before me on whose shoulders I stand, and to a society which encourages and rewards curiosity of the brain, be it alone or in groups.

This work was supported by Vinnova under project number 2002-00907, the Swedish Research Council under project number 621-2003-2991, and also by grants from the Swedish Emergency Management Agency (KBM) through Niklas Hallberg and Sofie Pilemalm, which is gratefully acknowledged.

Magnus Ingmarsson
Linköping, February 2013
# Contents

1 **INTRODUCTION & PROBLEM STATEMENT**  
1.1 The setting: Ubiquitous Computing .......................... 1  
1.1.1 Service discovery and Ubiquitous Computing ........... 2  
1.1.2 Creating useful service discovery for Ubicomp .......... 4  
1.2 How nature and humans do Service Discovery ............. 8  
1.3 Thesis Goals and Delimitations ............................. 10  
1.3.1 Goals ............................................. 10  
1.3.2 Delimitations ....................................... 11  
1.4 Contributions ........................................... 11  
1.5 Thesis overview ........................................... 12

2 **BACKGROUND**  
2.1 What a service is .......................................... 15  
2.1.1 Textual uses of the term Service ........................ 15  
2.1.2 Technical uses of the term Service ....................... 15  
2.1.3 Services seen from the perspective of Service Designers  
2.1.4 Our own perspective on services ........................ 17  
2.1.5 Reflection on Adam Smith’s view of service .......... 19  
2.1.6 Reflections on the service designer perspective of  
    services .............................................. 20  
2.2 Discoverability ............................................ 20  
2.2.1 Discoverability in SOA ................................ 22  
2.3 Service Oriented Architectures .............................. 23  
2.3.1 SOA for Command and Control systems (C²) .......... 25  
2.3.2 Service Discovery in and across Organizations ......... 25  
2.3.3 Design and Runtime Discovery ........................ 25  
2.3.4 Discoverability Meta Information ........................ 26  
2.3.5 Measuring Service Discoverability the Traditional way  
2.3.6 Quality of Experience in service discovery .......... 29  
2.4 Anatomy of choice ......................................... 30
CONTENTS

2.4.1 What is possible? ........................................... 32
2.5 Boundaries of the service-discovery system ..................... 33
   2.5.1 Terminology ........................................... 33
   2.5.2 Human discovery in the world .......................... 35
   2.5.3 Boundaries for service discovery ......................... 36

3 UBICOMP & CURRENT SYSTEMS ........................................... 39
   3.1 Ubicomp and its challenges .................................. 39
      3.1.1 Ubicomp ........................................... 39
      3.1.2 Challenges in Ubicomp ................................ 41
      3.1.3 Ubiquitous Computing and service-discovery systems 43
      3.1.4 DUB .................................................. 44
   3.2 Service Discovery in Ubicomp .................................. 45
      3.2.1 Comments on service discovery in Ubicomp .......... 47
      3.2.2 Service Discovery and Cloud Computing .............. 48
   3.3 Current Service-Discovery Technologies: General approaches ........................................... 48
      3.3.1 Bluetooth Service Discovery Protocol ................. 48
      3.3.2 Universal Plug and Play (UPnP) ...................... 49
      3.3.3 WS-Discovery ...................................... 49
      3.3.4 Zeroconf ........................................... 50
      3.3.5 Service Location Protocol (SLP) .................... 50
      3.3.6 Jini ................................................. 50
      3.3.7 Critique of current world descriptions .............. 51
      3.3.8 Critique of current service-discovery protocols .... 51
      3.3.9 Critique of current service descriptions .......... 52
   3.4 P2P in Ubiquitous Computing ................................ 54
      3.4.1 Edutella ........................................... 54
      3.4.2 Other efforts ....................................... 55

4 METHOD ................................................................. 57
   4.1 Methodology .................................................. 57
      4.1.1 Model and Systems development method .................. 57
      4.1.2 Evaluation method / Wizard Of Oz study of RESPONSEORIA ........................................... 58
   4.2 Outline of the research process ................................ 60
      4.2.1 Literature study and problem specification ............ 60
      4.2.2 Implementation of the Enhanced Traditional Layer in a system called ODEN, and evaluation of current common-sense research/systems .......... 61
      4.2.3 Specification of new revised demands, problems and scenarios ........................................... 61
      4.2.4 Implementation of the second Layer .................... 61
      4.2.5 Evaluation and analysis of implemented layers ....... 62

iv
5 **AN ENHANCED MODEL OF SERVICE DISCOVERY**

5.1 The service discovery system development process & its challenges ........................................ 65
   5.1.1 Challenges in the development process .......................... 66
   5.1.2 Development in accordance with Hallberg et al. .............. 67
   5.1.3 Additional four steps to enhance the development process ........................................ 69
   5.1.4 Logs and maintenance ................................................ 70
5.2 The layers and the motivation behind them ............................................................... 71
   5.2.1 Motivation for two layers ............................................. 71
5.3 The Enhanced Traditional Service Discovery Layer ......................................................... 71
5.4 The Relevant Service Discovery Layer .............................................................................. 72
   5.4.1 Ontology in The Relevant Service Discovery Layer .............. 73
5.5 User & World Model ......................................................................................................... 73
   5.5.1 User Model & its general rules ......................................... 74
   5.5.2 World Model & its general rules ....................................... 76
   5.5.3 Reasoning in the two layers ............................................. 77
   5.5.4 Alternatives to the ontologies and rules ............................. 78
   5.5.5 Rules and how they should be interpreted ........................ 80
5.6 Evaluation Platform ......................................................................................................... 80

6 **EXPLORING THE SERVICE-DISCOVERY LAYERS**

6.1 Exploring the Enhanced Traditional Layer using ODEN .................................................... 82
   6.1.1 OWL ............................................................................ 82
   6.1.2 ODEN ........................................................................... 83
   6.1.3 ODEN running .............................................................. 88
   6.1.4 Discussion of ODEN ...................................................... 90
   6.1.5 Observations regarding ODEN ......................................... 92
6.2 Observations regarding the use of CYC-like systems in developing service discovery ......................................................... 93
   6.2.1 Decision: Building our own ontology for our domain ........ 93
6.3 Exploring The Relevant Service Discovery Layer using MAG-UBI ........................................ 94
   6.3.1 Scenarios and use cases .................................................. 95
   6.3.2 Lunch scenario up close .................................................. 96
   6.3.3 MAGUBI system architecture ......................................... 98
   6.3.4 Control architecture of MAGUBI ..................................... 99
   6.3.5 Running MAGUBI ........................................................ 103
   6.3.6 Lessons learned .......................................................... 103
CONTENTS

7 EVALUATING RESPONSORIA 107

7.1 RESPONSORIA & Integration of MAGUBI in it ............... 107
7.1.1 Introduction ........................................... 108
7.1.2 Proposed solution ...................................... 111
7.1.3 Multimedia and crisis management ...................... 111
7.1.4 Multimedia technologies and their integration into ...
    RESPONSORIA ............................................. 112
7.1.5 Method .................................................. 115
7.1.6 Scenario ................................................ 116
7.1.7 The RESPONSORIA model and implementation ......... 117
7.1.8 MAGUBI service discovery and RESPONSORIA ......... 120
7.1.9 A brief introduction to the prototype user interface .... 121
7.1.10 Integration ............................................. 125
7.1.11 Observations .......................................... 126
7.1.12 Knowledge acquisition, Ontology construction, and ...
    Rule creation ............................................. 130
7.2 Reasons for a Wizard Of Oz Study ......................... 131
7.3 Employed method for evaluating MAGUBI ................. 132
7.4 Experimental and analysis setup .......................... 133
7.4.1 Participants ............................................ 133
7.4.2 Physical setup .......................................... 133
7.4.3 Ethical concerns ....................................... 134
7.4.4 Experimental Procedure ............................... 134
7.5 Analysis of experiment ...................................... 136
7.5.1 Results from the analysis of RESPONSORIA .......... 137
7.5.2 Suggested model ....................................... 140
7.5.3 The dimensions ......................................... 141
7.6 Summary of findings ....................................... 145

8 DISCUSSION 147

8.1 The Nature of Services ..................................... 147
8.1.1 Service Properties .................................... 147
8.1.2 Resources .............................................. 148
8.1.3 Checklists as a way to select the right services ...... 150
8.1.4 Implications for designing ontologies for service dis...
   covery in ubiquitous computing .......................... 151
8.2 Service Presentation and Development .................... 152
8.2.1 Service discovery versus user discovery ............... 152
8.2.2 Support for developers and service creators ......... 153
8.2.3 The smartphone as interface to the world and the ...
    services in it ............................................. 154
8.2.4 How to present appropriate services to the users ... 155
8.3 Other Issues ............................................. 156
8.3.1 Security ................................................. 156
<table>
<thead>
<tr>
<th>8.3.2</th>
<th>Privacy/ethical issues</th>
<th>157</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3.3</td>
<td>Measuring success for service discovery from a user and organizational perspective</td>
<td>158</td>
</tr>
<tr>
<td>8.3.4</td>
<td>Observations regarding complex systems</td>
<td>159</td>
</tr>
<tr>
<td>8.3.5</td>
<td>Service descriptions</td>
<td>161</td>
</tr>
<tr>
<td>8.3.6</td>
<td>Wiki interfaces</td>
<td>162</td>
</tr>
<tr>
<td>8.3.7</td>
<td>Formal rules vs. evolving rules</td>
<td>162</td>
</tr>
<tr>
<td>8.3.8</td>
<td>A choice between a transparent decision process or not</td>
<td>163</td>
</tr>
<tr>
<td>8.3.9</td>
<td>Intelligent Systems vs. Task Guidance Systems</td>
<td>163</td>
</tr>
</tbody>
</table>

**9 SUMMARY & CONCLUSIONS**

<table>
<thead>
<tr>
<th>9.1</th>
<th>Summary</th>
<th>165</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2</td>
<td>Conclusions</td>
<td>166</td>
</tr>
</tbody>
</table>
### List of Figures

1.1 An illustration of the interrelationships between the different parts of the system. The GUI allows the user to interact with The Relevant Service Discovery Layer and the Enhanced Traditional Service Discovery Layer. The Relevant Service Discovery Layer uses the Enhanced Traditional Service Discovery Layer and Jess. The Enhanced Traditional Service Discovery Layer makes extensive use of a Peer-to-Peer system and a Reasoner. The systems rest on a Java subframe which makes for easier portability and greater compatibility. Due to the modular construction, components can be switched out if the need arises. .......................... 7

1.2 Potato Cod being cleaned by two Striped Cleaner Wrasses. Ling [97] .................................................. 7

1.3 Sabre-toothed Blenny Fake cleaner fish. Will eat part of victim. Huang [71] ................................. 7

2.1 Relationships when gathering information before creating a service. .......................................... 18

2.2 Relationships while creating a service. ........... 18

2.3 Relationships when Using a service. ............. 19

2.4 Classical SOA. ........................................... 22

2.5 Example of Stovepiping within an organization. Note that there is no lateral transfer of knowledge and/or services. Service discovery may aid in promoting reuse of services as well as transfer of knowledge. Adapted from Martin [106] . 26

3.1 A modular view of Ubicomp, hierarchically arranged, showing major components and their subparts [74]. ........... 40
LIST OF FIGURES


4.2 Method: Literature study and problem specification . . . . 60
4.3 Method: Implementation of the Enhanced Traditional Layer in a system called ODEN, and evaluation of current commonsense research/systems. ................................. 61

4.4 Method: Specification of new revised demands, problems and scenarios .................................................. 61
4.5 Method: Implementation of the Second Layer, (MAGUBI) . 61
4.6 Method: Analysis of the first and second Layers (ODEN and MAGUBI) ................................................. 62

4.7 Method: Implementation of the third system RESPONSORIA (Scenario & requirements from Hallberg et al. [62]) .................................................................................. 62

4.8 Method: Qualitative analysis of RESPONSORIA ....... 63

5.1 Our two new layers are shown with back ground in the Figure. ................................................................. 66

5.2 (a) Adapted method from Hallberg et al. [62]. (b) Our additional steps. ....................................................... 68

5.3 Traditional Layer (a) vs. Enhanced Traditional Layer with its sub- systems (b). ................................. 72

5.4 The Relevant Service Discovery Layer with its subsystems. 73
5.5 Example of an ontology used in the Relevant Service Discovery Layer. ..................................................... 74

6.1 Our two new layers are shown with back ground in the Figure. ................................................................. 81

6.2 ODEN system hierarchy. .............................................. 83

6.3 The one-to-one mapping in the ODEN service discovery system between (a) the P2P (in this case JXTA) Name hierarchy and (b) the OWL-S Profile hierarchy. ......................... 84

6.4 A service advertisement in ODEN. The name places the device or service in a hierarchy; the WDSL field provides a programming interface to the service; and the OWL-S tags point to detailed semantic descriptions, retrievable from the service-providing peer. ............................ 86
6.5  The PDA-client starting in OdE.N. It starts by loading OWL-files into its JTP knowledgebase, starts the PDA peer for P2P-networking and finally creates and joins the Office peer group. ................................. 88
6.6  Simulated PDA-client GUI after start. Note that there is no device constraint selected in the left panel. The middle panel shows results when a search for services is done. The buttons to the right shows the corresponding information about the selected service. ........................................... 89
6.7  The search for Printer panel in OdE.N. To the left (A) can be seen the different properties that can be selected for the current device (in this case a printer), and also a column of numbers to the immediate right (B), where the user can set the priorities for the different properties. To the furthest right (C) are a number of buttons which when pressed bring up the corresponding information. ................................. 89
6.8  Printer1 log. Upon a service search, the different devices/services send the profile, process, service, and grounding information. .......................................................... 90
6.9  Results from OdE.N when we are prioritizing color. A higher number to the immediate right of the property indicates a higher priority for that property. ............................... 91
6.10 The location of MAGUBI in the system hierarchy. MAGUBI acts on-top of OdE.N filtering results and generating queries based on world knowledge. ................................. 95
6.11 Howard going to lunch. His Bluetooth cellphone provides his identity. OdE.N and MAGUBI work in concert to provide a suitable recommendation knowing his preferences and that he has access to a car. ................................. 97
6.12 One of the touch-screen entrance information terminals at the Computer and Information Science department at Linköping University. (Far right of the picture.) The terminal provides basic agenda information as well as contact and location information to persons, events and locations. ...... 97
6.13 Diagram over the MAGUBI lunch-ontology, containing information regarding places, transportation, appointments, and agents. .......................................................... 100
6.14 Diagram of the MAGUBI control flow, showing the reasoning process in the rule-based expert system, based on either results from OdE.N or the common-sense knowledge ontology. .......................................................... 102
6.15 Running MAGUBI. Showing logging outputs as well as the recommendation from the system. ................. 104
LIST OF FIGURES

7.1 The OODA loop (in Brehmer [23]). The OODA loop stands for Observe, Orient, Decide, and Act. Normally, it refers to a single person doing this cycle. However, the OODA loop can also be used when referring to organizations. Ultimately, the OODA loop describes how an individual or organization reacts to an event. ........................................... 111

7.2 Chart generated from the following URL: http://chart.apis.google.com/chart?chxt=x,y&chtt=Emergency%20room%20load%20and%20waiting%20time&chco=990066,0,0,0,30,1

The emergency room load is shown for the different locations as none, medium, or high. The size of the X indicates relative waiting time. .................................................. 114

7.3 Complan [90] showing different tasks to be performed in a crisis scenario and when to do them. ........................................... 115

7.4 Architecture of the RESPONSORIA system. (A) Web-based user-interface client. (B) Server cloud consisting of a collection of implemented web services running on application servers. ........................................... 118

7.5 The MAGUBI model. .................................................. 121

7.6 RESPONSORIA’s main interface. (A) Service selection panel, with pre-configured service groups for different scenarios. (B) Service panel showing the Mobile phone positioning service. In this case showing the a trace of a mobile phone for the last five hours. (C) Status panel, showing progress for different requests as well as tasks and activities. (D) Log. Showing all activity in the system as well as provider of a note taking function. ........................................... 123

7.7 RESPONSORIA at startup. Diagonally shaded area added to emphasize and denote area where service recommendations appear. ........................................... 134

7.8 Physical setup of the WoZ study: The experimenter/wizard watches as the user goes through the use case. The wizard also acts out the scenario as well as responds to the user’s input. ........................................... 135

7.9 The interface used by the wizard to control which services are displayed to the user in the service recommendation area of RESPONSORIA. ........................................... 136
7.10 The continuing service-discovery process. **Step 1:** Needs are identified. **Step 2:** Service is identified for/by user. **Step 3:** The service interface is discovered. **Step 4:** The result of the use is discovered and interpreted. **Step 5:** User 1 evaluates the whole experience and may communicate this experience to User 2, as well as using the evaluation in subsequent interactions with the system. **Step 6:** User 2 uses the information from User 1 to form an opinion of the service before using it.

7.11 Identified dimensions and properties in the service-discovery situation, and how they affect the ability to choose a service. Note that even with perfect suggestions/recommendations from a wizard, perfection is not experienced by the user.

8.1 Relationship between managerial actions applied to aviation safety. The figure depicts the requirements-solutions-decisions-implementation originating from known precedent. From: Miller [110, p.54]

8.2 Google Latitude showing the Author’s location Google Inc.

8.3 Left: User interaction with Intelligent System. From: Ockerman et al. [123]. Right: User interaction using a task guidance system. Figure adapted from: Ockerman et al. [123] Original part inside dotted line.
List of Tables

1.1 Summary of current service-discovery technologies and their abilities.................................. 8
3.1 Interaction paradigms........................................ 40
5.1 Service listing example.................................... 69
INTRODUCTION & PROBLEM STATEMENT

This thesis deals with one challenge:

How to find an approximation to ideal service discovery in terms of achieving an optimized balance between searching for, and presenting the user with the very best service alternative (according to given criteria), and the usefulness of a good-enough support service as experienced by a user.

The thesis has one overarching goal; to show it is possible to create human usable service discovery even when dealing with a potentially large amount of services. To reach this goal we first created a more advanced model of service discovery that incorporates a human dimension. Second, we verified this new model through implementing it in two systems, called MAGUBI and RESPONSORIA. Finally, using real users, we tested our two systems and the effect that ideal service discovery has on the perception of a service discovery system.

1.1 The setting: Ubiquitous Computing

Before discussing service discovery in detail, we will provide a brief introduction to ubiquitous computing.
1. INTRODUCTION & PROBLEM STATEMENT

“Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives.”

–Mark Weiser[166]

As Ubiquitous Computing [165] comes of age, it becomes clear that in order to reach Weiser’s vision from the 1990s, it is not sufficient to connect every service and device with every other service and device. Instead, it is important to connect the appropriate services and devices with each other, and foremost with the user. In this context appropriate means relevant and suitable services and devices for the user.

1.1.1 Service discovery and Ubiquitous Computing

Current service-discovery protocols are aimed towards technology and not towards people. That is to say, there has been a focus on solving the most pressing issue which is to get the machines to be able to connect to each other. Currently, this challenge is being met by several protocols [33] and technologies [53], and it is possible to envision a world where everything is potentially interconnected with everything else.

This development gives rise to a new challenge: Appropriate service discovery (ASD). ASD is a challenge because as the number of potentially available services grows, so also does the strain on the users. This strain is apparent if we consider the work the users have to do to sift through the mass of interconnected services. The strain is even further emphasized if we consider the possibility of aggregating or combining services. Not only will users have to sift through the services offered, but they will also have to determine how to combine the different services.

Aggregating and selecting among a plethora of services is rapidly becoming a common issue as the following example shows; The Swedish Armed Forces like most modern defense forces, are reinventing itself in a networked and service based direction [127]. To this end, the Swedish Armed Forces have started out by first enabling a greater interconnectivity and communication between the different units. When this interconnectivity was achieved, and all the units and their capabilities (services) were available to operators planning different missions, the operators encountered the challenge of ASD. The Swedish Armed Forces constructed a service demonstrator to show that it was indeed possible to utilize the interconnectivity between the different units to compose missions. While

---

1Mark Weiser is considered the founder of Ubicomp.
1.1. The setting: Ubiquitous Computing

this method was possible, it became apparent that a long list, (over 400 items), of available units is not a viable way of putting together missions. ASD is needed to make the list manageable. This realization has led to the Swedish Armed Forces creating profiles for typical missions, where the operator selects a type of mission and then automatically gets a pre-selected list of services that are appropriate for that type of mission. There are also civilian applications, such as rescue operations and web-services.

This need, to only see appropriate resources, is becoming increasingly apparent in a multitude of settings. Another example is crisis management. In crisis management, it is often the case that resources are collected and allocated on the fly when a crisis occurs. This on-the-fly allocation is due to the expense associated with having resources standing by at all times, since by its very nature a crisis is a fairly infrequent occurrence. An example of such an infrequently used service (in crisis management) is transportation. During a crisis, transportation must be allocated on-the-fly as need arises.

A common way of finding transportation services is to use the Yellowpages [178] and to manually call Taxi, and other Transportation companies. Depending on how the contact between the staff at the crisis center and the Taxi operators is carried out, some companies may be overloaded with requests while others have only a few. Service Oriented Architectures (SOAs) have been identified as one possible solution to this challenge. Ingmarsson et al. [78] By defining interfaces based on common web-service standards, the ordering, use, and exchange of services becomes possible. However, even though Enterprise, SOA, and web-based architectures are being used, service discovery is seldom built into these architectures. If these facilities exist, they are usually rudimentary in their functionality.

A further research challenge is when a system meets the criteria and the user actually gets what would be defined as ‘perfect’ service discovery. What are the follow-up issues? It is safe to assume that in dealing with human beings, there is no such thing as ‘perfect’. It is therefore reasonable to assume that there will be a host of new challenges that arise from a system that is working.

Service discovery in Ubiquitous Computing is a difficult challenge. It is not realistic to assume that the service-discovery challenge can be solved for the whole area of Ubiquitous Computing, such as any setting at any time. It is, however realistic to assume that it can be solved for a particular setting and a particular time. In this work, we have aimed to examine specifically the challenge of crisis management and ubicomp. This area can be well defined and provide a number of interesting and realistic scenarios. At the same time, it provides a bulwark against too extreme uncertainty in that the overall goal of the operation is known. Not knowing the overall goal for a person or organization makes performing advanced service discovery much more difficult.
1.1.2 Creating useful service discovery for Ubicomp

As mentioned earlier in Section 1.1.1, there is a difference between total service discovery and useful service discovery. Total service discovery may be seen as the Yellow Pages. In this book, we know that nearly every possible service is available. The problem is that although they are in a sense discovered, meaning that they are in the Yellow Pages, they are also in effect still unavailable for the user for a multitude of reasons. Continuing the analogy with the Yellow Pages, the fact that there is such a large number of listings makes it difficult to select the service desired. Also, since there are so many listings, it is impossible to read them all when searching. Hence, the Yellow Pages rely on users knowing the conventions (such as keywords), and also must incorporate a set of internal references pointing to the correct service in case the user of the Yellow Pages chooses the wrong word when searching. Furthermore, in the Yellow Pages, there are some services that are more prominent than others by virtue of how large the entry is. Comparing this to the real world, we see that advertisements that are physically bigger get more attention.

As mentioned in the beginning of Section 1.1, service discovery is a crucial part of Ubicomp. At a rudimentary level, there already exist solutions for service discovery, such as hardware able to self interconnect through Universal Plug and Play (UPnP) [53]. It would appear that the challenge is met. Users do not need to do any manual configuration. However, as the number of devices increases, so does the load on both man and machine.

Subsequently, the issue is not really how to find things and connect them together. The issue is how to find and connect the correct things in order not to overload the machinery and the human user. There is a need for a service-discovery approach designed for the ubiquitous domain, which is also able to perform some form of reasoning about what to connect. For an introduction to current service-discovery techniques, we recommend reading Section 3.3.

One of the oldest research questions in Computer Science is; “What can be efficiently automated?” [35, page 41]. This question can be related to the service discovery problem. Is it possible to automate this in a Ubicomp setting, and if so, how can it be done? The other issue is the one of efficiency. Here, efficiency may be measured by the sheer ability to do what the human is incapable of doing, namely sifting through a high number of services and presenting the ones relevant for the user. Looking at service

\footnote{There is also a secondary issue of presenting all this information to the user in a ubiquitous computing environment, but GUI aspects are not within the scope of this thesis. It will suffice to say that if we have done our job, the GUI designer should be in a better situation to start with. This is also inline with the research directions in everyday computing identified by Abowd and Mynatt [1].}
1.1. The setting: Ubiquitous Computing

discovery from a Ubicomp perspective, we have identified four properties that we deem important for a system to possess:

1. Network independence. By placing the service-discovery mechanism at a higher level of abstraction than peers, peer groups, etc.

2. Peer-to-peer (P2P) discovery. We argue that service discovery should be unmediated if it is to be suitable for ubiquitous computing and ad-hoc networks.

3. Expressiveness. Service descriptions must be expressive and flexible, scaling to future device types and providing support for reasoning about services.

4. Common sense or world knowledge. Reasoning about services is not sufficient. The system must be able to act in a proactive fashion and take the benefits and restrictions from the surrounding situation as well as the innate wishes of the users into consideration. This property can be related to the why mentioned by Abowd and Mynatt [1].

The preceding four properties all contribute to finding relevant services, but also to reducing the workload of the users. With the increased number of devices at the users’ disposal, it is imperative that a service-discovery system can filter out services that are of limited interest to the users. Otherwise, a cognitive overload can take place, and potential useful services can be drowned in the noise from services that are currently irrelevant for the users.

After identifying these properties, it is possible to analyze current service-discovery protocols based on these. First, we note that some of the existing protocols depend on specific network transport layers. To ensure this network independence, service discovery in Ubicomp should be placed at a high level of abstraction. A second issue is the use of devices that serve as service registrars, so called brokers, and the suitability of the service-discovery technologies for P2P networks. Requiring the use of proxies or mediator services may mean that many peers lose their service-discovery capability if the central mediator goes down, or if the network connectivity to the mediator is faulty. This centralization could also mean that users have to configure manually which mediator service to use, or at least have to know which mediator services exist. Furthermore, in an environment where devices are highly mobile, and the state of devices is rapidly changing, having a mediator could entail an excessive amount of work, as the mediator service must be updated to reflect all such changes. It is fundamental to P2P systems that all peers are able to connect directly to each other, and this is one of the reasons why we feel that a P2P approach is suitable for our ubiquitous computing research.
A third issue to discuss concerns the expressiveness and flexibility of device descriptions and discovery requests. While using appropriate network technologies can solve the first two issues we have mentioned above, those of network independence, and direct peer-to-peer discovery, this third issue concerns the expressiveness and flexibility of device descriptions and discovery is more difficult, and requires some novel thinking. None of the standard service-discovery architectures we have examined scale well in this respect; without expressive device descriptions, service-seeking peers cannot reason about devices in an intelligent way.

The fourth and final issue to consider is the ability to use common sense in the service discovery process. Notoriously difficult to define, perhaps the most important thing about common sense is what it enables us to do as human beings.

“Commonsense knowledge is knowledge about the structure of the external world that is acquired and applied without concentrated effort by any normal human that allows him or her to meet the everyday demands of the physical, spatial, temporal and social environment with a reasonable degree of success.”

—Benjamin Kuipers [159]

An example of this would be to know that although it is possible to print to a printer, there is no way to physically gain access to it and pick up the paper. None of the standard systems are equipped to handle this requirement.

For a comparison of systems with respect to these properties see Table 1.1. We choose to see issues three and four as two additional layers in a service discovery system. In addition to conceptualizing these layers, we have also made proof of concept implementations of them as well and call them ODEN and MAGUBI respectively.

Indeed, it could be argued that world modeling is also beneficial for the system as a whole, since it can help prohibit spurious service discovery results as well as initiations, thus reducing the load on the system and network. One classic example might be the basic situation where a user would like to print a document. With world models, it becomes possible to see that it might be impossible to collect a printout behind a locked door, even though the first stage of service discovery results in a large number of potentially usable printers through the network.
1.2. How nature and humans do Service Discovery

Figure 1.1: An illustration of the interrelationships between the different parts of the system. The GUI allows the user to interact with the Relevant Service Discovery Layer and the Enhanced Traditional Service Discovery Layer. The Relevant Service Discovery Layer uses the Enhanced Traditional Service Discovery Layer and Jess. The Enhanced Traditional Service Discovery Layer makes extensive use of a Peer-to-Peer system and a Reasoner. The systems rest on a Java subframe which makes for easier portability and greater compatibility. Due to the modular construction, components can be switched out if the need arises.

Figure 1.2: Potato Cod being cleaned by two Striped Cleaner Wrasses. Ling [97]

Figure 1.3: Sabre-toothed Blenny Fake cleaner fish. Will eat part of victim. Huang [71]
1. INTRODUCTION & PROBLEM STATEMENT

Table 1.1: Summary of current service-discovery technologies and their abilities.

<table>
<thead>
<tr>
<th>Type</th>
<th>Peer 2 Peer discovery</th>
<th>Network independence</th>
<th>Expressiveness</th>
<th>Common Sense</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jini</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bluetooth</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPnP</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zeroconf</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TL\textsuperscript{a}</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>RL\textsuperscript{b}</td>
<td></td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

\textsuperscript{a}The Enhanced Traditional Service Discovery Layer
\textsuperscript{b}The Relevant Service Discovery Layer

1.2 How nature and humans do Service Discovery

Although service discovery in the computerized world is new, humans have been engaged in this activity since they first appeared. When we as humans walk about in the world, we are constantly looking for services. On the savannah, we looked for resources in terms of food, water, and tools (or items that could be constructed into tools). In today’s world, producers of services as well as of items, use knowledge to sell their products to us. However, we as humans, have a unique capability of filtering out irrelevant service information. For instance, we only perceive toilet signs when we are actively searching for them.

Services are not unique to the human animal however. Figure 1.2 shows the Striped Cleaner Wrasse cleaning a Potato Cod from parasites. The question is how the Cod does the service discovery. The Cod service discovery actually has two parts. The first part is the location. Cleaner fish have cleaning stations which they inhabit. The Cod and other bigger fish know about these locations and go to them. In fact, there is sometimes even a line of fish waiting their turn to be cleaned. The second part of service discovery for the cod is to look at the dance that the cleaner fish performs. This process makes sure that the bigger fish does not eat the cleaner. Notably, the cleaner fish dances more vigorously for hungry clients, even though at present the scientific community does not know how they know this. Grutter [58] The Cod allows the Cleaner fish to nibble on parasites that are embedded in the skin, gills and so on. Wounds are also allowed to be cleaned. Sometimes the service discovery process goes wrong however.

\textsuperscript{7}As: The Enhanced Traditional Service Discovery Layer & The Relevant Service Discovery Layer
1.2. How nature and humans do Service Discovery

In Figure 1.3 a Sabre-toothed Blenny can be seen. It mimics the dance of the authentic cleaner fish and when the victim exposes itself, instead of cleaning it, the Blenny eats part of the victim and quickly swims away.

There is an interesting analogy between the Cleaner fish and service discovery. This analogy leads to a number of insights that we may use when approaching the computerized version. First, we may observe that location is important in a physical service. The cleaner fish have cleaning stations that other fish frequent and queue at. A service may be useful only in a certain physical location, or it may be useful in any physical location.

Second, we can observe that there is a temporal aspect to a service. The cleaning takes a certain amount of time per fish. Notable is also that the service completion time may vary. A big fish takes a longer time to clean than a small fish. That is to say that the task entrusted to the service, may take different amounts of time depending on the client as well as on the task. What is also clear is that a task may be partially completed and still be satisfactory to the client. For instance, the fish that is cleaned may elect to swim away after only being partially cleaned; since it may value satisfying hunger more that being cleaned.

Third, the fish will build a history with each other. A history leads to a construction of a reputation or rapport. This bonding can only be done when the identity of another fish is known. When using a service, identity is crucial, if for instance financial transactions are to transpire. Another way in which a history is useful is knowing the requirements of a client in the case of the provider, and knowing the capabilities of a service provider in the case of the client.

Fourth, usefulness is the key to a service. There must be demand for the service. A service without demand is as useless as demand without a service. ‘Useful’ means that someone experiences a gain when using the service or when providing it.

Fifth, security is also important when dealing with services. In the example above, the Sabre-toothed Blenny will mimic a cleaner fish, but eat part of the victim instead. The conclusion that can be drawn from this mimicry is that in real life there is a need for identification as well as safe transactions if security is to be maintained. Furthermore, when there is too little security of any kind, usage and usefulness will suffer.

All this being said, there is a tradeoff between most of these properties. For instance, the user of a service may elect to give up information willingly if the experience is increased. Similarly, a user may have higher demands on a service which is payed for than a service that is free, and so on.

---

*Service provider* will in this thesis be used as a designation for the party offering a service and not as a name for a telecom operator.
1.3 Thesis Goals and Delimitations

Naturally, a thesis cannot address all the possible issues that arise within a subject. Below we define our goals with this thesis as well as delimitations for what we do not cover within the scope of this thesis.

1.3.1 Goals

The primary scientific goals for this thesis are listed below:

- To gain an increased knowledge into the challenge of how, from a large set of services, to discover and recommend a selection that is fairly optimized towards the users needs and situation.
- To add to the knowledge about services by creating more in depth service descriptions.
- To attempt to establish what types of world and user knowledge that are needed to create more focused service recommendations.
- To provide a deeper understanding of the process used in developing service discovery systems centered on the user.
- To identify common properties that describe services.
- To add to the body of knowledge in human machine interaction by investigating what the user experiences when near perfect service discovery occurs.
- To explore the ethical as well as privacy concerns of the user in a ubicomp service discovery setting.
- To establish if the approach described in this thesis is viable in multi-actor rescue operations setting.

Specifically, the practical goal for this thesis is to create a more suitable, usable, and relevant service-discovery design for the user, and if possible for the developer and service designer as well. This goal is approached by addressing the challenge of finding for the user, suitable services from the large amount available in a ubicomp setting. The primary concerns are the users and their experience.

Another goal is to evaluate our design in accordance with the previous goals. To achieve this, we set the following goals. For the user we aim to identify user requirements, create an experience which does not overwhelm the user, has an uncomplicated/automated search and which produces relevant results that are also personalized and context dependent. For the developer and service provider, we aim to identify requirements as well as gain an insight into what are the required standardized descriptions for joining a service pool.
1.3.2 Delimitations

Service discovery in itself is a large area. However, service discovery is not unique to the computer field as discussed in Section 1.2 in this thesis. Therefore it is necessary to delimit the scope of what we intend to convey or refer to when we say service discovery in ubiquitous computing: Service discovery in the scope of this thesis connotes the process with which users become aware of a service, device, or person in which the particular user is interested.

Furthermore, it does not include the type of discovery which does not involve some sort of computer assistance. This means that service discovery in terms of ordinary human service discovery, such as walking down the street looking for a specific resource, or accidentally coming across a resource and realizing that one has a use for it, is not included in this thesis. These issues are however interesting in themselves, and this thesis may contribute to starting points and directions in which to take such research. Moreover, it does not entail the type of service discovery performed with computer assistance which addresses connectivity issues, such as connecting a printer to a computer. Connectivity issues such as these are already addressed by a multitude of protocols and discovery systems. Instead this thesis focuses on the issue of what happens after the device has been connected.

1.4 Contributions

The main contribution to human–machine interaction and ubiquitous computing of this thesis is the formulation, implementation, and evaluation of a novel service-discovery architecture, through the introduction of new layers that assist the users (see chapter 5).

The Enhanced Traditional Layer, which is realized in ODEN, contributes to computer science and service discovery through an ontological model for rich service descriptions (see Section 6.1). The Relevant Service Discovery Layer, which is realized in MAGUBI, contributes to human–machine interaction and ubiquitous computing by employing user and world knowledge to create a proactive and autonomous service-discovery experience (see Section 6.3). The proof-of-concept system RESPONSORIA contributes to the verification of our model by serving as a platform for our Wizard-of-Oz study (see Section 7.1.7). In addition, RESPONSORIA contributes to command-and-control systems by illustrating the development of light-weight, web-based systems employing services and service-discovery.
1. INTRODUCTION & PROBLEM STATEMENT

The development of the model, together with the implementation of the systems, resulted in additional practical contributions in the form of the following:

- The implementation of OdEn further contributes to the know-how of rule-set and ontology creation (see Section 6.3.3 and, 8.1.4) for service discovery in ubiquitous computing by the development of a test scenario.

- By conducting an exploration of the measure of what successful service discovery is, the Wizard-Of-Oz study contributes to service discovery through the proposal of some success metrics (see Section 8.3.3).

- The investigation into the nature of services contributes to the field of service design and ubiquitous computing, by providing common properties that are shared among different services (see Section 8.1.1).

- The Wizard-Of-Oz study contributes to human–machine interaction by providing better understanding of what having near-perfect service discovery brings with it (see Section 7.5).

- A comparison of our systems in comparison to regular service-discovery systems, through an ethical/privacy perspective, contribute to human–machine interaction and ubiquitous computing, by creating a better understanding of the users’ privacy concerns (see Section 8.3.2).

- The Wizard-Of-Oz study specifically contributes to the knowledge about these systems within command and control with multiple actors for rescue operations (and indirectly to human–machine interaction, and ubiquitous computing), by showing that RESPONSORIA-like systems are a viable solution for this type of service-discovery challenge (see chapter 7).

1.5 Thesis overview

The remainder of this thesis is organized as follows: In this first chapter, the challenges are defined and outlined. Chapter 2 and 3 presents the background to the area as well as related research, in order to lay a foundation for examining the state of the art. Chapter 4 presents the method used in this work. Chapter 5 presents the enhanced model for service discovery. Chapter 6 presents implementations and run throughs of our two new layers, The Enhanced Traditional Layer as well as The Relevant Service Discovery Layer. Chapter 7 reports the results of the
1.5. Thesis overview

Wizard-Of-Oz study. Chapter 8 discusses the issues in and around this thesis. Finally, Chapter 9 summarizes this thesis and presents conclusions.
Chapter 2

BACKGROUND

Here we give a background to the area in terms of what a service is. This is done mostly from a technical standpoint but also includes the perspective of service designers. Furthermore, we also explore what discoverability is from a computer-science perspective and we examine the anatomy of a choice. Finally, we investigate the boundaries of the service-discovery system.

2.1 What a service is

Here we explore the textual, technical, and service designer views of the concept of a service.

2.1.1 Textual uses of the term Service

In attempting to define what a service is, we may look at the definitions available. There is a large number of modern modern definitions of this term. The word may be used as both a noun, verb and an adjective, as well as in idiomatic expressions. The normal use however, is as an act of helpful activity; help; aid: to do someone a service. From a human standpoint a service is in a way a ritual, and as such is governed by certain assumed conventions.

2.1.2 Technical uses of the term Service

The technical uses of the term service is derived from many of the above concepts, and as in everyday language, in computing the word service also have a number of different meanings. One of the earlier and more common ones is a background service, or daemon, most notably in UNIX-systems. The daemon is not normally controlled by the user directly, as a program is,
but rather provides useful service to the user, such as ssh-login ability, ftp-service, http-server, and so on. The word daemon may sound sinister, but actually derives from a Greek concept of a benevolent supernatural being situated between man and gods (Rexine [134]).

Service may also be viewed from a systems architecture point of view. In this case, it normally refers to a set of software functions that go together, as well as to the restrictions or policies that govern the use of these functions. The following characteristics are also notable. A description of the service, constructed in such a way that it is clear what the service does, and that this description may be read by a machine, but simultaneously a description not so specific that it becomes an implementation of the service rather than a description of it. A way for a machine to verify different aspects of performance. How long does it take to complete the service? What is the access time? When should the service be available? What is the frequency of the usage of the service? Another important aspect of describing a service is to know what the service is owner of in terms of information and resources, as well as what it receives as third party information from other actors and services. All of these properties are important in that they characterize the service and make it possible for an organization to integrate the service, as this is one of the main reasons for using the service concept in the first place.

2.1.3 Services seen from the perspective of Service Designers

Research into what a service consists of, is something that is done by researchers into service design. Among others one view is that of Adam Smith, who claims that products can be produced in advance and then stored before utilization or consumption, whereas a services cannot.

One common example that Smith uses is the military. They are not able to store “the capacity to defend the nation”, it simply has to be there. So, according to Smith, the military is unproductive, whereas a cobbler is productive.

A popular view is that there are four characteristics that are distinctive of services (Zeithaml et al. [179]). These are:

**Intangibility** Since a service is not a product in the traditional sense, it does not have a physical presence. In other words, it cannot be picked up, touched, smelled, and so on.

**Inseparability** This refers to the fact that it is impossible to separate the consumer of the service from the producer of the service. The actual production of the service is inseparable from the consumption of it. This inseparability leads up to another common viewpoint among service designers, which is that the consumer actually co-creates or is co-creator of the service (Vargo and Lusch [161]).
2.1. What a service is

Heterogeneity Since services are performed by different individuals at different times, and with different mindsets, it is not possible to foresee the end result of a service procedure. In contrast, when producing products it is easy to determine the end result. Additionally, when the consumer of the service is involved in the process, the same principles apply to him/her as to the provider of the service.

Perishability A service is not persistent over time. It cannot be stored for later use. As Adam Smith pointed out regarding the defense of a nation, it cannot be preserved, but rather is present in time. This perishability also highlights the need to have resources at hand at any given time. “If demand is low, unused capacity is wasted. If demand exceeds capacity, it goes unfulfilled and business may be lost” (Lovelock and Grumnesson [101]).

These four characteristics are known as IHIP (Intangibility, Inseparability, Heterogeneity, and Perishability) (Zeithaml et al. [179]).

2.1.4 Our own perspective on services

We take the perspective of all three preceding views (textual, technical, and service designer) and would like integrate them. We believe that it is of vital importance to view the services from all three perspectives. We take this position due to the fact that all three views contribute to the sense-making of services. Taking the narrow view of only one of these perspectives may lead to a distorted understanding of what services are, in particular for other actors coming from other areas or fields.

In the end, the perspective of the programmer, provider, or service designer has little value without the user. With the term user we refer to the heterogenous group of potential users of the system. Thus, it is more often than not insufficient to design/develop for one user-type, but instead many user-types need to be catered for. Indeed, it is the user who is the reason for the service in the first place. Without the user, there is no service. Indeed, to satisfy the user of the service is the most important thing of all, and the raison d’être for the service in the first place is to fulfill a demand from the user of the service.

Another aspect of a service is the relationships that they foster and create. Since a service most often is created to meet a demand, or in some cases to create a demand, there is a relationship between the designer of the service and the programmer/creator/implementer of the service. The user and the provider are also involved in creating the service, but more often than not, their involvement is only peripheral in the beginning of the process (see Figure 2.1 p. 18).

In the best of cases the users are consulted by the service designer and programmer, and in the worst of cases completely left out. Currently,
2. BACKGROUND

Information gathering

![Diagram of relationships during information gathering]

Figure 2.1: Relationships when gathering information before creating a service.

Creating a service

![Diagram of relationships during service creation]

Figure 2.2: Relationships while creating a service.
2.1. What a service is

![Diagram: Relationships when Using a Service](image)

Using a Service

Figure 2.3: Relationships when Using a service.

The attitude towards the users and provider in that more and more companies are realizing the importance of their involvement early on in the process (see Figure 2.2).

However, after the service has been established, the provider and the user are all that are left. However, when creating the service, the main work is done by the designer and the programmer. Hence, from our perspective there are three stages involved in the production of a service. The planning, creation, and the usage. In the last part, the usage, it is only the user and the provider that co-create, provide, and use the service (see Figure 2.3).

A further reflection regarding services is on what occasions can a service be denied. There are many such instances that may be envisioned, such as failure of payment, unsatisfied requirements to use the service, service overload etcetera. However, it is of utmost importance that the service does not spend more time than necessary in qualifying users, since the qualifying part of the service also uses resources.

2.1.5 Reflection on Adam Smith’s view of service

Adam Smith’s view that it is only the person who produces goods that actually contributes to the economic wealth of a nation, is understandable put in the context of its time.

“The labour of some of the most respectable orders in the society is... unproductive of any value, and does not fix or realize itself in any permanent subject, or vendible commodity, which endures after that labour is past, and for which an equal quantity of labour afterwards be procured. /.../ Their service, how honourable, how useful, or how necessary soever, produces nothing for which an equal quantity of service can afterwards be procured.”

–The Wealth of Nations [147, p. 356]
Indeed, it is not uncommon for discussion to arise regarding what contributes to a society’s wealth.

For instance, Adam Smith would most likely hold the view that a medical doctor does not produce a product and as such, does not contribute to wealth building in a society. Many of the economists of today would argue that the service the doctor performs actually produces wealth, even though it might not be direct, since the doctor is not directly involved in creating a product. It is also possible to take the view that a service does in itself create wealth since a company that produces services may make a profit.

2.1.6 Reflections on the service designer perspective of services

With respect to services and heterogeneity, there is a common belief among service designers that one cannot expect everyone that produces a service to have the same standard. While this is true in principle, it is reasonable to assume that certain minimum standards can be met. For instance, it is reasonable to assume that a taxi service moves a customer from point A to point B, even though the time it takes may vary, as well as numerous other factors such as level of comfort of the car, car size, exact route, pleasantness of the driver, and so on. However, a physical product is easily standardized by different measures, strengths and so on. (For instance a bolt.) The important aspect of a service is its outcome. If the outcome is as promised, then the service has been performed, otherwise it has not.

The view of service designers that a service with unused capacity is business lost is not only true for services. It is also true for products. If a factory producing cars does not sell all it can produce, the remainder is wasted. The analogy that services are different from products in that they are perishable, is also not true in all instances. Food products get spoilt for example.

2.2 Discoverability

When referring to Discoverability in the sense used by computer science, there are a multitude of properties that come with it. For instance, we must be able to communicate information about the available resources in a manner that can be decoded by potential users of the service. This information is commonly defined and documented by the actors that know the most about the service. Somehow, this information also has to be stored for and accessed by the discoverers.

Since new services may be made from aggregation and combination of old ones, one of the key points in discoverability is for the discoverer to understand if the service required can be satisfied by using an already
existing service, composed of old services, or if a totally new service is required. Another issue is fall-back. It is possible that a black and white printer may be used instead of a color printer. Normally, this type of fall-back handling is not included in protocols that are common in service-discovery.

Naturally, there is a risk that if the capabilities of an already existing service are not understood, a incorrect use may be attempted or no use at all may be attempted. A less dangerous development is if an existing service happens to overlap with a new service, since there is more capacity and redundancy this way. Indeed in a service based “economy” the purpose of multiple providers is to drive the price of the service down. A service-discovery system that collapses under heavy load or due to many services being in the system, is not Enterprise-ready. It is advisable to use an already existing Enterprise architecture, where this already has been solved as a base.

A service is sometimes referred to as a solution logic when it is viewed from an architectural perspective. When discussing discovery from a system developer or programmer perspective, it is important that the interpretability, (ability to discern the capabilities of the discovered service), is good enough for the programmer to decide if s/he can use the service or not. Essentially, the first step is to query and then filter out the services that are sufficiently good to warrant further scrutiny. When services are chosen discovery is often said to take place first, and is then followed by the interpretation process with assessment and selection. We would like to argue that the service discovery process never stops and that the interpretation process is simply an extension of the discovery process.

One of the main reasons for having interpretability and discoverability, is to get an ordered discovery and interpretation process. This is usually done by means of a service registry. Having a service registry gives an organization control over its discovery process by providing it with a defined and formal way of locating, retrieving, and evaluating/interpreting the service descriptions and meta data.

When a programmer or user searches for a service, the common procedure closely resembles the following [44, p. 366]:

1. The programmer/user looks in the registry for a service that fulfills the desired requirements.

2. The service registry returns information about the services as well as pointers to service contracts.

3. The programmer/user then interprets the results and identifies the potentially correct service to utilize.

4. The service contract corresponding to the selected service is retrieved and attempts to interpret it commence. If the service is deemed
appropriate, the discovery phase is finished and the utilization phase begins. However, if the service identified does not provide the necessary functionalities, but still provides a good basis for development, it may serve as a starting point for an extension of the service.

2.2.1 Discoverability in SOA

UDDI made its appearance as a part of the first-generation Web services platform [44]. Although it was not widely embraced, the major materialization of SOA soon followed suit. One might say that it was the very existence of the discovery mechanism that distinguished SOA from other architectures of distributed systems. The classical model of SOA shown in Figure 2.4 has been surpassed with more modern ones, but it still illustrates how important discoverability is. Thus, service

![Figure 2.4: Classical SOA.](image)

discoverability is then composed by discoverability and interpretability, but service discoverability in itself may be further profiled in the following way [43]:

**Short definition** Services are discoverable.

**Long definition** Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.

**Goals** Services are highly discoverable resources within an Enterprise. The purpose and capabilities of each service are described so that they can be interpreted by computer software as well as humans.
2.3. Service Oriented Architectures

Design characteristics  The service contracts are filled with correct metadata which will be reported when discovery inquiries are filed. The contracts also have extra data which is easily understandable by human beings. In the case of a service registry, its records are created in the same fashion as above. When no service registry exists, special service profile documents are created to serve as a foundation for when the registry is eventually created.

Implementation Requirements  Design standards control the metadata information that controls the discoverability and interpretability as well as information about how and when to supplement the metadata with extra annotations. Another requirement is design standards that define how to store the metadata information that is required outside the contract. This information may either be stored in documents outside the registry, so that it can be in the registry at a later date, or directly be stored in the registry. The lack of a service registry in these requirements is noticeable evident. However, the goal is not to implement design characteristics within the architecture, but within the service.

Web Service Region of Influence  Even though it is practical to have a service registry, it is naturally advantageous for a service to be discoverable by itself. As such, the focal point is on the service contracts and not on the logic in the service nor on the logic that handles communication.

2.3  Service Oriented Architectures

Service Oriented Architectures (SOA) are closely connected to Service-Discovery. Creating a system in the paradigm of SOA means that a distributed IT-system is organized in a structure of communicating services. This enables a multitude of desirable properties, such as Quality Of Service (QoS) assurance, encryption and so on, that may be obtained ‘for free’ by using already existing enterprise-grade systems. However, using SOAs also brings with it a need for some kind of service discovery in the system. For instance, when a service that is needed is not available, a replacement service will have to be discovered and brought into the SOA-system.

In a SOA architecture, there are services (or service providers), and service consumers. To produce a SOA, loose-coupling is required between the service itself, the service-consumer and the operating systems that the service, SOA and service consumers run under. This loose-coupling lends itself well to web-based services (web services). Web services use standardized internet protocols that fulfill the loose-coupling required.

The services commonly adhere to the following principles (Erl [43]):
2. **Background**

- abstraction
- autonomy
- composability
- discoverability
- formal contract
- loose coupling
- reusability
- statelessness

It should also be noted that even though SOA in combination with web services produces easy interoperability, there are some drawbacks in speed, most notably due to extensive layering and XML-parsing.

A service is stand-alone and may exist without the need for external systems. For instance, there might be a Temperature service that provides the service-consumer with a Temperature reading. This Temperature service may be replaced by any other Temperature service that provides the Temperature that the service-consumer wishes to obtain. A service may also be a so-called an aggregate-service. Being an aggregate service means that it is comprised of other services. The fact that it is comprised of other services may or may not be known to the service-consumer. It is also possible that a service works by coordinating other services. It is worth mentioning that a service in a SOA is well defined in terms of what it provides, as well as the protocol/interface for obtaining the service.

**SOA in systems**

The overarching characteristics of SOA may be summarized in the following four points:

**Business-Driven** When establishing the system, care is taken to make certain that it is in sync and alignment with current the business climate and culture. When developing the system further, this sync and alignment are carefully maintained.

**Composition-Centric** SOA accommodates aggregating multiple services. This aggregation makes the system agile since it is easy to accommodate change by simply using the currently available services.

**Enterprise-Centric** The SOA-architecture makes sense from a business perspective since it represents a meaningful business unit. Having this representation allows for composition and reuse across what are traditionally referred to as silos.
2.3. Service Oriented Architectures

**Vendor-Neutral** Since there is no proprietary dependence within SOA, this allows for constant recombination and replacement over time to maximize business requirements as they arise.

A commonly used standard for web-services is Business Process Execution Language (BPEL) [120]. BPEL is a standard that enables adaptable connections between different systems. However, service discovery is not a focus of BPEL. There has been research aimed at service discovery in conjunction with BPEL, such as that by Grigori et al. [56], Xiao et al. [176], and Wombacher et al. [169].

2.3.1 SOA for Command and Control systems (C²)

Note that in Command and Control systems is the extra emphasis on redundancy and uptime. SOA is especially suitable in this respect. The use of SOA in C²-systems is something that has been studied for example by The United States of America’s military forces. Their system ‘The Global Command and Control System’ (GCCS) consists of SOA-systems and applications.

2.3.2 Service Discovery in and across Organizations

A challenge both within and across organizations is to find the required services. This challenge has a number of reasons, but among them is stovepiping,\(^1\) (see Figure 2.5 p.26) a lack of dissemination of information (Smith and Clemente [148]). Much research has been done on how to disseminate information within an organization, and across multiple organizations (Barua et al. [13]). Cooperation across organizations provides an opportunity for information sharing. Indeed, it may be argued that cooperation is not possible without information sharing. However, there is a difference between sharing in which the different parties share out of necessity and sharing which it is done willingly. Software and systems that enable sharing of information automatically is valuable to a company (Echikson [40]).

2.3.3 Design and Runtime Discovery

When constructing systems and reusing components, the service discovery is conducted by humans, in the sense that it is the individuals that search through service directories during the construction process. This may be referred to as design-time discovery. Service records and contracts are reviewed during the process.

\(^1\)A structure of an organization, which allows for, and/or promotes hierarchical/vertical transfer of information, but not horizontal transfer of information.
At the other end of the spectrum is the situation when programs themselves can issue requests for services. This process is called runtime discovery. Automated runtime discovery is possible due to the programmatic interfaces that the programs have available to them. Even though this might suggest that automatic service discovery and consumption is easily available, this is not the case today and this is one of the key issues addressed in this thesis. What is easily available is meta data though WS-Metadata exchange (W3C [163]), but this is quite rudimentary.

2.3.4 Discoverability Meta Information

The classical view of Meta Information is that it is necessary for traditional service discovery and that it is one of the four different types: 1. Functional. 2. Technology 3. Programmatic. 4. Quality of Service (Erl [44]). Another assumption that is commonly made is that the format of the meta-information is standardized for the specific conglomerate in which it is to be used. The most basic aspect of the Functional meta information is the service name. If the service name is non-descriptive and the service
itself does not have a standardized contract, it may be debated whether the service is discoverable at all. Furthermore, if the functional Meta Information is all there is, manual interpretation and adaptation to its use are normally required.

With regard to Quality of Service Meta information, it is there to ensure a timely delivery of the service that the user requires. Here such information as time to completion, time from contact to contract, policies are defined. This is to ensure that runtime requirements are met and may allow for fairly elaborate searches with multiple criteria stipulated.

2.3.5 Measuring Service Discoverability the Traditional way

Erl [44] suggests the following checklist for measuring service Discoverability:

1. Has functional meta information been documented in plain English?

2. If yes, has functional meta information been clearly expressed as part of the service contract for discoverability purposes?

3. Has quality of service information been documented in non-technical English?

4. Has quality of service meta information been clearly expressed as part of the service contract of formal SLA?

5. Has a service profile document or (if a service registry exists) the corresponding service registry record been created?

6. Does the service profile/registry record contain all relevant functional meta information?

7. Does the service profile/registry record contain all relevant quality of service meta information?

8. Have business subject matter experts contributed to the definition of business centric discoverability meta information?

9. Has all discoverability meta documentation been subjected to standards and conventions to ensure consistency?

10. Has all discoverability meta documentation been reviewed and refined by a communications expert?

As Erl [44, p. 376] states:
2. BACKGROUND

What these simple levels communicate is only that the necessary discoverability meta information was put in place and that interpretability has not been taken into account. Because they do not communicate the quality of this information, these levels may not be an indication of a service’s actual discoverability.

It is also clear that this checklist focuses on technical issues and not on world information or contextual information about potential users. Nevertheless, it brings a formal way of assessing important technical information which is fundamental to the discoverability of a service. However, Erl [44] also mentions a Custom Rating System, where the ratings used need to reflect the context in which the service operates, what the service can do as well as including the discovery platform in use.

To gain this valuable Discoverability information, it is necessary to involve experts such as business analysts early in the design process together with the technology architect, and this is often neglected according to Erl [44]. When the service-oriented analysis process is complete, the technology architect remains and performs the service-oriented design process by formalizing the meta information, such as the behavior and business that was obtained during the first phase.

Service granularity does not need to be affected by its Discoverability. These entities are distinct from each other and a coarse service may have excellent Discoverability, just as that of a highly granular service may be poor. The service model itself should also not influence Discoverability. If granularity is not affected by Discoverability, and models should not influence it, Discoverability may be affected by Policy Assertions which use for instance WS-Policy features (e.g. wsp:optional, wsp:ignorable), policy alternatives and policy parameters. Even though these features are aimed at the usage of the service, their presence and values may transmit information of the underlying characteristics of the service and thus affect its Discoverability. The WS-Policy however, is somewhat further down in the system hierarchy and as such, is mostly aimed at developers or needs to be adapted in order to be useful to end-users.

Discoverability affects other issues such as contracts, abstraction, reusability, and composability. In case of contracts, Discoverability influences what is published in the contract, since the advertised features must match what is actually offered. With respect to abstraction, there is tension between reducing the amount of information to make the service more abstract and appealing to more consumers by publishing more information. If too much information is published, it becomes hard to maintain the system and there is a risk for over-documentation in the service contract. A highly Discoverable service is also highly reusable. Therefore it is in a service’s own interest to be highly Discoverable to attract as many of the correct users as possible. Again, clarity in the description
2.3. Service Oriented Architectures

is the key. When composing services from other services, the traditional view is that this is done at design-time. However, it is desirable to reach a situation where this is done automatically. By keeping an eye on redundant services within a business organization, increases in efficiency are reached, as well as an increased ability to compose new services at run-time.

Discoverability carries with it its own set of risks. Poorly implemented Discoverability may reveal itself at some time after the system has been put into operation. Additionally, an attempt to apply Discoverability after the implementation phase is complete may be hazardous, since freshness of the knowledge of what actually is and is not, is lost. This means that the quality of the information will be lower. This is especially true if reverse engineering is attempted, and even more so if the people involved are different from the people who originally designed the system. The documentation and/or design of the service contract is often done by the same professionals that perform the implementation. In other words, programmers and highly skilled technical people. These professionals may not be the best at communicating the information about the Discoverability to all the people that need to receive it, such as managers, analysts, and potential external entities.

2.3.6 Quality of Experience in service discovery

There is a close relationship between Quality of Experience (QoE) and appropriate service discovery. QoE is commonly defined as the quality perceived by a user of a system. The International Telecommunication Union defines it as:

“The overall acceptability of an application or service, as perceived subjectively by the end-user. NOTES 1 Quality of Experience includes the complete end-to-end system effects (client, terminal, network, services infrastructure, etc). 2 Overall acceptability may be influenced by user expectations and context.” –Dvorak [38]

One example of QoE in low-level service discovery is that of adding a resource reservation scheme to avoid frequent reconfiguration (Delphinanto et al. [34]). This scheme can also bee seen as rudimentary common sense. This allocation is different from the QoS mentioned above, since the QoS can be objectively quantified, whereas the QoE often can not. Another view is that although QoE is qualitative in its nature and not qualitatively measurable, it may be comprised of parts that in themselves are quantitatively measurable, such as transaction time.

For instance, in the case of finding an appropriate service, the user will be content with finding a service that corresponds to his/her needs at the time. The maxim is:
2. BACKGROUND

If you don’t discover all the services, you won’t know. But if you discover the wrong service, you will certainly know.

It is essential to understand the importance of this statement. The idea that in advertising its capacity for intelligent behavior a system, should understate rather than give an accurate estimation, has been mentioned before (Dinka and Ingmarsson [37]). This underestimation of the system’s performance is done to place the user in a position to be positively surprised, rather than negatively surprised when using the system. It also has the additional benefit of conditioning the user to willingly adapt his/her behavior to better suit the system\(^2\). In this way, using the system is seen as less cumbersome compared to a situation where the user perceives a forced change in his/her behavior.

2.4 Anatomy of choice

Having a choice means that a decision has to be made. With regard to the different types of decisions that can be made, four types are generally identified: 1. Command decisions, which can only be made by the actor him/herself. 2. Delegated decisions. 3. Avoided decisions. Decisions whose consequence may be catastrophic and unchangeable if the wrong decision is made. 4. “No-brainer” decisions, where the choice is so obvious that only one choice can reasonably be made (p.153 Tracy [157]).

There is also a philosophical aspect to making choices, since we define ourselves to a certain extent from the choices we make. As Nietzsche noted:

“Und wahrlich, hätte das Leben keinen Sinn und müsste ich
Unsinn wählen, so wäre auch mir dies der wählenswürdigste
Unsinn.”\(^3\) – p. 30 Nietzsche [115]

General decision theory can be said to originate with Nicolas de Condorcet [105], who introduced a three-staged method. An influential figure in modern decision theory is John Dewey [36] who talked about five stages. Herbert Simon turned these five stages into three [143]. Intelligence (as in gathering), design, and choice. This was then further refined by other researchers such as Mintzberg et al. [113] who created a similar model, but which was non-sequential.

When and actor chooses a service in a computerized setting, the process is similar to that of using other interactive software and may be viewed as follows (Salen and Zimmerman [137]):

\(^2\)This can be compared to the way in which persons change their behavior when they interact with children or animals such as dogs.

\(^3\)“And verily, if life had no sense, and I had to choose nonsense, this would be the desirablist nonsense for me also.” –Nietzsche [116]
1. What happened before the actor was given a choice? Here there are two aspects:
   - The internal representation of the external world and actor in the system that is presenting the choice.
   - The actual external world in which the actor and system act.

2. How is the possibility of a choice conveyed to the actor? The choice is made using either information available at the moment through the five senses, or information that is recalled. When choosing real world, visual impressions play a big part. The possibility of making a choice is also dependent of the world being in a special state that enables the actor to make the choice at all. For instance, there might be advertisements for certain restaurants along the street, but if they are closed or located in a different place it is impossible to choose them. In the first case, the actor may choose to wait until the service is unlocked/opened. In the second case, a choice to move to the correct location may still be made, but this would require the actor to be able to plan in several steps. This stepwise planning is not difficult for a human being, but may be more challenging for a computerized system. Currently, common service discovery systems, do not handle multiple step planning.

3. How do we know the actor made a choice? In the real world, the choice is made by actively utilizing the service, such as restaurant. In the virtual world, the result of a choice may be a physical utilization, but it may also be the case that the utilization of the service takes place in the virtual world, and the interaction is intermediated by a digital device.

4. What are the results/effects of the actors’ choice? In case of eating in a restaurant, the actor will probably have satisfied his/her hunger. Therefore it is important that the service discovery system also has some knowledge of, or means for, the effect or post conditions of the choice. Knowing the post conditions is important, since these influence and affect the actors’ next steps and choices. Once post conditions are satisfied, they will in turn become pre conditions, starting a new cycle of choice and service discovery.

5. How is the result of the choice conveyed to the actor? A result of a choice is commonly conveyed to a human actor by his/her becoming aware of a changed state in the world or within. As in point 4 when the actor is no longer hungry. In the case of a service discovery system, the issue becomes more complex. The service discovery system might employ dead reckoning, that is, assume that certain
2. BACKGROUND

actions that the actor performs in the environment will have intrinsic effects on the actor and/or the environment. Another way for the system to perceive the results is to observe the actor’s behavior. If after paying the bill in a restaurant, the actor indicates a preference for another type of activity by his/her actions, then the system may assume that the post condition is satisfied.

2.4.1 What is possible?

The most fundamental query that the user may put to a service-discovery system in theory is: What is possible? However, this is a question which in practice is almost never asked. In essence this question decides what a service discovery system can and cannot do. In the case of common service discovery systems, this is not a significant issue since the reasoning power of the system in itself is not great, and thus the system is severely restricted. However, when the reasoning power of the service discovery system increases, so does the risk of running into complexity issues such as NP-completeness. Therefore it is important to limit the service discovery system with clear boundaries beyond which it may not venture. This boundary making might seem simple, but recall that efforts such as CYC [91] have spent a lot of time and effort in precisely this area.

Part of the purpose of these boundaries is to introduce fundamentals into the service discovery system. What is important here is to determine what types of properties will help the service discovery and subsequent utilization to progress. However, as we introduce fundamentals in the form of rules into the service discovery system, we increase its complexity, since we have to introduce and define a concept in to be able to reason about it. We should therefor be careful when we introduce a concept or rule into the system. We should always ask ourselves: Is this absolutely necessary for a better service discovery? For example: The introduction of space and thus time, in the form of the time it takes to get from point A to point B, could be viewed as a great improvement in service discovery. On the other hand, time may be viewed simply as a property of a service without any greater implication for the service discovery process.

The challenge in detail

There are different levels to the challenge:

1. At first, the service discovery system needs to know what is going on. That is to say, what activity is taking place. This is done by analyzing all available sensor data as well as integrating it with knowledge

---

*A discovered service is only a service if it can be utilized.*
boundaries of the service-discovery system

about the world and user. An alternative is to ask the user what is currently happening, or simply let the user feed this into the system.

2. Once this is known, the system can decide what service to look for.

3. However, it should be noted that even though the system knows what service it wants, there is still a significant challenge in choosing which service to use, since there might be many services that fulfill the criteria. One way of solving this challenge is to use meta properties that may influence which service to choose.

2.5 Boundaries of the service-discovery system

In this section we focus on Terminology, Human discovery in the world, and conclude with Boundaries for service discovery.

2.5.1 Terminology

There are many concepts that need to be addressed in service discovery to properly understand its purpose. In many protocols and technologies, the responsibility of the service-discovery element is merely to discover the services and then hand over the interface information to the other parts of the system. This presents a number of challenges as previously discussed. Briefly, we can observe that there is a minimum amount of logic and intelligence that need to go into the service-discovery layer. Without this, service discovery cannot be performed. At its core, the service-discovery system needs to respond to a request to find a service.

The most simple request would be: `<Find All Available Services>`. This request returns a pool of services. As we can see, this places the burden solely on the requestor, since evaluation of which service to select has to be performed there. A somewhat more advanced service discovery system would have either an option for what type of service to discover, or a method for recognizing what services should be returned, on the basis of who is asking. In the case of the what type option, the burden is still to a certain extent shifted onto the requestor, since the service-discovery system in the general case will return more than one service.

It is already possible to see the two different paths ahead: 1. Insert the intelligence for making advanced service discovery into the consumer of the services. 2. Insert the intelligence for making advanced service discovery into a separate service discovery system.

Naturally, there are many aspects to be optimized for in service discovery for ubiquitous computing, Network bandwidth, Response time, User experience, Autonomy, — to mention just a few. Looking at service discovery from a historical perspective, the first challenge was be able to
2. BACKGROUND

do service discovery at all. It is also important to consider what challenge
service discovery was supposed to solve. Early in computing there was
a need to integrate hardware/components, get it/them to communicate,
and to be able to perform component/hardware upgrades. This prompted
the development of ways to configure components/hardware. At first
this configuration was done manually, then semi-automatically, and finally
automatically. It is clear that service discovery in its most primitive form
can be viewed as device/component/hardware discovery.

A similar development can be seen in external peripherals. Where,
there was no other possibility for automatic configuration than input
devices. Keyboards were easily connected and used. The more complex
devices such as printers, were often connected through a parallel port,
which like the serial port, had to be configured. Today these types of
devices use a Universal Serial Bus (USB). USB is hot pluggable as well
as auto configuring, and many types of peripherals such as scanners,
joysticks, cameras, storage units, and printers, today use the USB-system.
USB, may be said to host a primitive service discovery system, since it
allows for the identification of the connected units as well as prompting
the host operating system to load the appropriate drivers.

With regard to the configuration of networking between different
computers, again there was no easy auto-configuring solution. The user
had to specify a large number of parameters manually. Today, network
configuration is often carried out automatically through a Dynamic Host
Configuration Protocol (DHCP). DHCP works by the client first broadcast-
ing a DHCP discovery packet on the subnet that it is connected to. Two
things should be pointed out: 1. There is a special type of packet that is
used for DHCP 2. It is broadcasted on an entire subnet. From these two
statements, we can conclude that for automatic configuration to occur, there
must be a method for discovery, and it must be well defined.

However, certain items have been free of configuration and have been
plug-and-play since the beginning of computer history. These items
include screens and keyboards, both of which are necessary boot-strap-
devices/services needed to operate a computer at all. There are multiple
ways to configure wireless peripherals and devices, from pressing two
pairing buttons simultaneously on the devices to a system with somewhat
increased security which involves entering a code once on one device and
entering the same on the device it is supposed to be paired with, as is the
case with Bluetooth devices.

We may then conclude that in the case of computer hardware, there
was a real challenge to be addressed. This challenge was how to make it
easier for people to put together their own computer systems. Interestingly
enough, it is with the advent of reconfigurability within computers that this

\footnote{Printers were sometimes used as an alternative to screens.}
2.5. Boundaries of the service-discovery system

has become a problem. When computers were static the configuration was performed by experts. One might say that service discovery then became a mainstream challenge only when it involved enough people for whom configuring computers, networks and services were not a main interest in life, but what computers could be used for once configured.

The same is true for services. Society is now at a point where the saturation level of networked devices, awareness of their potential, and cyber-literacy are high enough for there to be a market as well as an interest for what can be achieved when leveraging services. As has been observed, simple services such as Twitter have become immensely popular, in 2011 it ranked number 9 according to Alexa.com [5] and had 50 000 000 tweets per day (Beaumont [15]). However, the next step, and which is currently emerging, is the combination of services like Twitter with location services for instance.

2.5.2 Human discovery in the world

Connecting back to Section 1.2 on page 8, human service discovery also needs conventions and configuration. This starts from when we are born and is highly culturally dependent. Encouragingly, as we have learned from the ability of human beings to filter out advertisements with almost zero perceived mental effort and only consciously notice those considered relevant, it is conceivable that the same principle may be applied to the general case of service discovery. The fact that we sometimes view computer-based interfaces as intrusive and frustrating indicates that there is a fundamental flaw in the design of the said systems. While the scope of this thesis does not allow for an in-depth analysis of the root causes behind this perceived intrusion and frustration, we may briefly consider how we as humans, interact with systems in the world that we perceive as non intrusive but yet efficient, and which able to provide us with the right information at the right time. There are two aspects to this phenomenon:

**when the user wants to find a service/device** It may be the case that the user is actively searching for a specific service. The user will then accept to do some work, such as actively consult other people, guidebooks, or similar resources. It also means that the user is primed to look for a certain word or symbol. It is important to recognize this priming since it has a direct impact on what the user will look for, as well as the person’s ability/performance to find the correct service. Thus, it may very well be the case that a Swedish man in Japan may inadvertently use the women’s door in a lavatory instead of the men’s, since he is unable to understand the correct symbol. Sometimes there is also a gender and/or age difference (Olinstead [124, pages: 313–320]). It is also clear that knowing the correct
symbol as well as the correct manner in which to advertise for the intended target, is imperative when designing services that are to be discovered.

**when the user accidentally finds a service/device** even though the user is not searching for a service but still registers an offering. This registration may be caused by an advertiser utilizing knowledge about how the human body and mind react to certain stimuli, or by similar offerings that may not be directly designed to do this registration but may have enough impact on the user’s peripheral perception to cause an escalation from unconscious registration to conscious registration. This escalation is especially true with regards to certain signs such as those for toilets, restaurants, or other things that may be of interest to the individual, but for which s/he is not actively searching for at the current time.

### 2.5.3 Boundaries for service discovery

As mentioned in section 3.1.3, a service-discovery system for Ubicomp needs to be able to handle a diverse array of situations and challenges. It is therefore necessary to define, what is and what is not a service-discovery system in this context. In essence, we need to define the boundaries of the service-discovery system, or in other words its responsibilities.

Looking at traditional/mainstream service-discovery systems, their responsibility ends as soon as they have discovered a service. In some cases there might be a continuous search for new services in the background. In other cases the user is required to actively initialize a search. It is clear that different service-discovery systems have different aims. For instance, Bluetooth [19] requires the user to manually initialize a search, although the reverse (being discoverable) can be automated. Bluetooth does not require any extra configuration once set up and produces a wireless alternative to cables. However, many implementations of the original Bluetooth design would sometimes lead to confusion in a scenario such as the following:

*One headset configured with two devices.*

In theory, this configuration works well. As long as the two devices that are paired to the same headset are far enough apart, everything works as planned. However, when the two devices that use the same headset are close together, the result is an undefined behavior. The reason for this undefined behavior has to do with how Bluetooth was envisioned originally. Bluetooth was often thought of as a replacement for a single cord.\(^6\)

---
\(^6\)Many times multipoint is not implemented in a device even though it has been available from v.1 of the specification.
2.5. Boundaries of the service-discovery system

Nonetheless, since there is no actual physical cord, the user may use the device in a way that was not intended, since the perceived affordances are different from the actual ones. In this particular case, it is possible for the user to imagine that a headset can be used in the same way as a corded headset, that is to say, use it with multiple devices, and as mentioned previously, this works well as long as the two devices using the headset are far enough apart. The affordance that was not conveyed in this particular case was that of a corded headset plugged into two devices at the same time. If that had been conveyed, the user would have instantaneously understood the possibility for error and would have been in a position to rectify the situation, either by turning off Bluetooth on one of the devices or unpairing one of the devices and the headset.

Another observation regarding this situation is how Bluetooth and its service discovery system deals with this conflict. In this particular case, the result was a confused state, where two devices tried to allocate the same resource/service. It is clear that there is a need for service discovery systems as well as services that are able to handle this type of situation.

The above shows a lesson to be learned when implementing services and service discovery: For services, it is paramount that they handle sessions or their equivalent. Clearly, there is a challenge to create systems that allow for flexible access to services while at the same time ensure a correct allocation of resources in time. For service discovery systems and protocols, competition for resources is detrimental if left unchecked.

An important issue is whether it should be the responsibility of the service-discovery system or the service itself to ensure a satisfactory performance when it comes to resource allocation. One argument could be put forward that it is in the self-interest of the service to ensure that its clients are satisfied and keep coming back.

As discussed in Ingmarsson [74], there is clearly a need for more advanced systems to handle the complex reality of service discovery. It is not enough to merely discover services. It is not even enough to connect to the right services. Connections have to be made in the right way at the right time, and the whole session has to be executed in a manner with which the human client ultimately has good Quality of Experience. As demonstrated, there are a host of challenges to address and part of this work is to clarify and point out these challenges.

A list of considerations/challenges for service discovery research:

- Descriptions of services
- Descriptions of world
- Descriptions of actor
- Right service at the right time
2. **Background**

- Service quality verification
- What can be viewed as "session" handling

As the issue of service discovery becomes more detailed and complex, so does the need for defining the borders or boundaries between service discovery, service recommendation, service suggestion, service auto-use, and so on. Discovery only makes the system perform a very limited task. However, it is valid to discuss whether a service-discovery system should do anything more than merely discover. It is our contention that useful service discovery cannot be decoupled from some form of evaluation or reasoning.
Chapter 3

UBICOMP & CURRENT SYSTEMS

To frame service discovery in general from a more human perspective, we need to look more into those elements of computer science that closely relate to humans and service discovery. One area where human-centeredness and service discovery intersect is ubiquitous computing (Ubicomp) and its challenges.

3.1 Ubicomp and its challenges

As computers have gone from being few and expensive to many and inexpensive, the user interaction has shifted from batch jobs and timesharing to personal computing. We are now expanding personal computing to ubiquitous computing, where users interact with many computerized devices in the environment. However, most of the current application-development methods and human-machine interaction paradigms are aimed at one computer–one user. With the increasing proliferation of computer-enabled devices, current software- and interaction-development methods are not able to harness the full capacity of the Ubicomp environments.

3.1.1 Ubicomp

We are currently transitioning into the Ubicomp stage. Users will have access to many computerized devices, which leads to new requirements for thinking, developing, modeling as well as interacting, Larsson and Berglund [85]. Interaction with these Ubicomp environments will be different from the computer interaction that we are used to today. One of the most fundamental differences will be the on the fly pooling of different interaction elements, and resources from different devices and services, in order to create custom interfaces that fit a particular situation as well as
3. Ubicomp & Current Systems

Figure 3.1: A modular view of Ubicomp, hierarchically arranged, showing major components and their subparts [74].

Figure 3.1 below presents our view of Ubicomp (Ingmarsson [74]). We believe that an intricate part of Ubicomp is the UbiOS, the equivalent of an operating system for ubiquitous computing, or at least an OS that is capable of coping with a ubiquitous environment. A significant amount of effort has been devoted to creating hardware infrastructure and more specifically components or “gadgets” (Borniello and Holmquist [21], Davies et al. [32], Alois Frescha [6]). Given the lack of infrastructure and relative young age of Ubicomp as a field of science, this bottom-up strategy is understandable.

Other efforts have been aimed towards finding ways of interacting with devices in a more ubiquitous manner, by using enhanced affordances already provided in the environment. For example, augmenting binders to support automatic document tracking while maintaining the traditional

<table>
<thead>
<tr>
<th></th>
<th>One Computer</th>
<th>Many Computers</th>
</tr>
</thead>
<tbody>
<tr>
<td>One User</td>
<td>Personal computing</td>
<td>Ubiquitous computing</td>
</tr>
<tr>
<td>Many Users</td>
<td>Time sharing</td>
<td>Walk-up computers</td>
</tr>
</tbody>
</table>
3.1. Ubicomp and its challenges

interaction, using traditional TV-remote controls to input text supported by predictive completion systems in the TV-set, and the idea of using robotic animals as interfaces to the home, in order to keep expectations low about the computer controlling the home (Ingmarsson et al. [77], Ingmarsson and Dinka [76], Ingmarsson et al. [79]). This experimentation has resulted in many insights into the infrastructural aspects of Ubicomp, and into a situation where all the infrastructural parts exist as islands separated from each other with little potential for synergetic effects. One explanation for the lack of common ground for hardware communication is that Ubicomp is a novel field.

The next step for Ubicomp is to move towards interaction among these infrastructural elements. It is time to start building a UbiOS, and attempts to do so have been made [10]. In Figure 3.1, there are a number of key components in the UbiOS that need to be addressed in order to realize the vision of ubiquitous computing. Distributed memory (DM) is required in order for the devices to operate on the same data. Distributed User Interfaces (DUI) [85, 86] are required to provide an optimal user experience. Finally, service discovery is imperative not only for the reasons mentioned above, but also for other parts of the UbiOS such as the DUI, to work correctly.

3.1.2 Challenges in Ubicomp

Abowd and Mynatt [1] identified three major challenges. The first, being a desire for natural interfaces, in other words interfaces that support common forms of human expression. The second is context awareness, to ensure correct behavior in different situations. The third challenge identified is automated capture of experiences in order to have access to these experiences later.

Breaking down these challenges, we find sub-challenges that need to be resolved first. In connection with the issue of natural interfaces, we can see that one of the biggest efforts in Ubicomp has been to create devices. However, many of these devices have focused on one-off solutions for particular problems and not as general components in a larger context. An example of this is wall-mounted screens that have often not been used as general display and interaction devices but rather as parts of a particular engineering problem or application.

With regard to context-awareness, Abowd and Mynatt first established what they believe to be a minimal set of necessary context: Identity in the form who; activity in the form of what; location in the form of where; time and passage of time in the form of when; and the reason why the users are doing what they are doing. Furthermore, Abowd and Mynatt stressed the need for a good representation of context in order to make progress with correct responses to that particular context.
With regard to issues arising from automated capture, Abowd and Mynatt saw two major challenges: Capture and Access. Capture is problematic in terms of having the right capture device at the right time and capturing the right thing. Access is problematic when we want to find specific sections of a recording. Abowd and Mynatt ended with a brief mention of privacy concerns regarding all the recorded material.

A recurring theme in Abowd and Mynatt is the notion of everyday computing. Everyday computing is defined as the result of scaling ubiquitous computing with respect to time. This definition is due to the nature of informal daily activities. The specifics listed as typical for these are: Activities rarely have a beginning or an end. Interruption is expected. Multiple activities operate concurrently. Time is an important discriminator. Associative models of information are needed.

Abowd and Mynatt pointed out the following challenges as main focal points for their research: designing a continuously present computer interface; presenting information at different levels of the periphery of human attention; connecting events in the physical and virtual worlds; modifying traditional HCI methods to support designing for informal, peripheral, and opportunistic behavior.

Evaluation and social implications are two additional aspects that Abowd and Mynatt emphasized by the authors. They argued that there has to be a real human need for the Ubicomp system when evaluating it, since it is often at the cutting edge of technology. Moreover, evaluating in the context of authentic use is crucial in order to obtain correct results. Another point that is accentuated is that task-centric evaluation techniques are inappropriate, since they evaluate how well a system fits a particular task at hand and not the system itself. Privacy aspects as well as potential security issues are also mentioned as important challenges.

Another perspective of the vision and challenges of Ubicomp comes from Satyanarayanan [138], who identified the following research avenues: The first is effective use of smart spaces which means that by embedding computing infrastructure into building infrastructure, they are brought together. The second is invisibility. The aim of Ubicomp is to be unobtrusive and not enter the users’ minds until they decide they need it. Satyanarayanan also stressed that a modicum of anticipation may be essential to avoiding a large unpleasant surprise later, much as pain alerts a person to a potentially serious future problem in a normally unnoticed body part. [138, page 11]

The third is localized scalability, which means scalability in its broadest sense. How do we handle multiple users who want the same resource?

---

1This is also touched upon in: Pederson [129]
3.1 Ubicomp and its challenges

The fourth and final challenge is masking uneven conditioning; that is, how to obtain a graceful degradation of performance, since not all parts of the environment are immersed in Ubicomp.

Satyanarayanan also listed some concerns regarding users' intent. Depending on what the users are trying to accomplish, the system has to respond by doing different things. According to Satyanarayanan Important questions here are: How does one represent user intent internally? How is the accuracy of the knowledge represented? Will the attempt to discern user intent impact the user, and if so how? Cyber foraging is also brought up in the article, and the issue of whether or not surrogates for the smaller devices that the users are carrying will be readily available in the surroundings. A number of questions follow: How are the surrogates discovered? How is security handled? What time-frames are necessary for the surrogates so that they can fulfill their role?

3.1.3 Ubiquitous Computing and service-discovery systems

Ubicomp, like most types of computing, places special requirements on its components. To understand these requirements we need to understand what Ubicomp is and what it is not.

Ubicomp is non-intrusive or calm, that is to say that it is a type of computing that is designed not to require human interaction unless the human chooses to interact. It is also possible to define Ubicomp as computing done by the users without realizing that they are doing computing. This required calmness, places a restriction on the service-discovery protocol. The protocol cannot require frequent interaction from the user to carry out its task. In fact, it is whether or not the service-discovery protocol should require user interaction at all, at least when it comes to the day-to-day running of the service-discovery system.

Ubicomp is also an abundant amount of devices working together. There is a view that by its nature Ubicomp, must include cheap devices, and while that is certainly a prerequisite to saturate one's life in Ubicomp, it is equally true that expensive devices may also be used. For the service-discovery protocol/system, this implies that it has to be able to handle inexpensive devices as well as expensive ones. Inexpensive often translates to devices with less capacity in terms of memory, CPU-power, network connectivity and capacity, interaction capabilities, and so on. Expensive can also be interpreted as being more complex, although this is not necessarily the case in every situation.

Ubicomp is by nature transient. Transient implies a number of properties.
3. Ubicomp & Current Systems

1. It cannot be assumed that all services are available at all times.
2. It cannot be assumed that services are stationary.
3. It cannot be assumed that services will be available for the whole time needed for a request to be completed.

A service-discovery system for Ubicomp, must then be adapted to this kind of environment. That is to say, the environment in which a service-discovery system for Ubicomp works is highly volatile since services cannot be assumed to be available for the duration of a service request to complete. The service-discovery system must handle these eventualities.

Ubicomp has different interaction venues than traditional computing. Traditionally we interact with computers through keyboards, mice, and screens. In Ubicomp, this type of interaction is not necessarily the case. Common types of interaction in Ubicomp include: gesture, natural language, haptic, and so on. Service discovery intended for Ubicomp environments must take this fact into consideration. For instance, it may not be assumed that there is a keyboard or mouse available at all times when using the service-discovery system.

Ubicomp is non-stationary. Whereas traditional computing is mainly static, Ubicomp is highly mobile. This does not necessarily imply that all the objects that are part of the Ubicomp experience are mobile. It does however, imply that it cannot be assumed from the outset that they are not.

3.1.4 DUIS

As an example, we can take the issue of distributed user interfaces (DUIs)\(^2\)[144]. DUIs make it possible to distribute the user interface (UI) among several devices. However, DUI-technology in general does not decide how to distribute the UI-components but rather, enables the possibility of distributing the components. If the user is to not only decide which services and devices to use and combine, but also how to distribute and combine the UI-elements for aforementioned services and devices, then Weiser’s vision is far off indeed.

\(^2\)By DUI we mean a User interface that is spanning over many devices, some of which might have other primary functions, for example a cell-phone. As an illustration of this, imagine a home where the home owner wishes to control the temperature setting. The service of temperature control and adjustment, might for instance be combined with DUI in the following way: The user has a number or generic displays and wishes to use part of one of them to show the set temperature as well as the current one. Furthermore the user wishes to control the setting by using his/her cell-phone. Using a DUI this is possible.
3.2 Service Discovery in Ubicomp

In the paper “Service Discovery in Pervasive Computing Environments”, Zhu et al. [180] noted that a number of service discovery protocols have been designed, however that although not specifically aimed towards the Ubicomp field, can be utilized. They also noted that Ubicomp environments are far more dynamic and heterogeneous than enterprise environments. The authors also pointed out that ambient services have limitations with regard their physical location, which web services do not have. This possibly makes the service discovery protocols based on LANs and single-hop wireless communication too coarse in a Ubicomp setting. Zhu et al. also argue for a common platform so that different service discovery protocols can interact with each other.

Furthermore, Zhu et al. identified ten components of existing service-discovery protocols and approaches:

1. **Service and attribute naming.** This can take the form of a template-based naming, or template-based and predefined naming.

2. **Initial communication method.** Three different methods to establish contact are listed; Unicast4, Multicast5, or Broadcast6.

3. **Discovery and registration.** This can be Query-based, meaning that the service/device user actively sends out requests for the services, or Announcement-based, i.e., the services/devices make themselves known.

4. **Service discovery infrastructure.** Whereas the Directory-based approach has a dedicated component that processes queries and announcements, the Nondirectory-based approach has no such component.

5. **Service information state.** In the case when the Soft state variant is used, announcements specify the service’s life span. However, with a Hard state approach it is necessary for the clients and possible directories to poll the services at periodic intervals.

6. **Discovery scope.** In the case of Network topology scope, the network itself is used to regulate how far away services are discovered. One hop is usually the case. The User role scope employs the user identity to establish the area in which the services are found. Finally, in Context driven scope, information such as temporal, spatial as well as other activity information is used to set the scope.

---

4Sending packets to a single destination.
5Sending packets to multiple destinations in the form of a multicast group.
6Usually this means sending packets to all within a single hop range in a network.
7. **Service selection.** Here the system can do a fully automatic selection for the users or allow them to make a manual selection for themselves.

8. **Service invocation.** There are three levels when invoking a service. A first level system only presents a pointer to the service location. If the system has support for a second level it will in addition reveal the communication mechanism for the service. Finally if it has support for three levels it will also provide application operations for the service.

9. **Service usage.** This describes when the subscription to the service ends. Here it is possible to have explicitly released, which means that the service user has to explicitly release the service. It is also possible to have a lease-based service usage, in which case the possibility to use the service automatically ends when the lease ends.

10. **Service status inquiry.** Here the choice is between getting direct notification from the service or having to poll the service for updates.

Zhu et al. also touch upon security and privacy issues in service discovery, and mention that many solutions fail simply due to service environment changes. They also advocate end-to-end encryption as well as the possibility for users to use the service-discovery mechanism without having any special skills. Finally Zhu et al. stress that service discovery in unfamiliar computing environments has not been addressed well, and it is in these kinds of environments Ubiquitous computing takes place. Taken as a whole, this means that there is a need for more intelligence in the underlying computing infrastructure.

Chakraborty et al. [25] note that service discovery is a well-recognized challenge in distributed environments. They also emphasize that there will be a need for a flexible service discovery architecture that is tailored for Ubicomp. An important distinction is made between the discovery architecture and the service-matching mechanism. The discovery architecture can be viewed as an analogy of how the network is constructed and how the different nodes find each other. The service-matching mechanism can be viewed as the identification of the correct service searched for. The also note that syntactic level matching is inefficient in Ubicomp environments, due to the autonomy of service providers which will result in heterogeneity of implementations and interfaces [25]. Chakraborty et al. also addressed the issue of service vicinity, by creating groups and including the possibility to limit access to the services based on distance in the semantic network, which is the network formed inside the ontological structure.
3.2.1 Comments on service discovery in Ubicomp

Zhu et al. [180] also point out that there is a need for more intelligence in the underlying computing infrastructure. As far as service discovery goes, we have chosen to focus on this as well as other aspects of service discovery for Ubicomp, see Table 1.1. To reiterate, we believe that P2P-discovery, Network independence, Expressiveness in the service descriptions, and Common sense are of paramount importance to further the proliferation of Ubicomp. Our layers, implemented as ODEN and MAGUI, contribute to more intelligence in the underlying computing infrastructure through the use of semantics and common sense, as well as to the issues identified.

Chakraborty et al. mention the need for a service discovery infrastructure for Ubicomp, and argue for the combination of discovery architecture and service-matching mechanisms. We share this view and want to take it further with the addition of common sense into the service discovery system.

With regard to the mention of policy-based/serendipitous advertising made by Chakraborty et al.’s [25, pp. 98, 101] it is our opinion that this is most likely a view that is consistent with Ubicomp. We believe that policy-based/serendipitous advertising will be advantageous in a future Ubicomp environment, where the users will have friends and antagonists. By friends, we mean other users and services that can be drawn upon for help, whether it be for accreditation purposes or information sharing, etc. By antagonists, we mean devices, services, and to a certain extent other users, that the users want to steer clear of, much in the same way that users in general want to steer clear of pop-up advertisements today on the web. Any research into service discovery in ubiquitous computing will have to take this into account in order to be able to operate in the real world. Semantics and common sense models of the world may help in this endeavor. Chakraborty et al. also focus on evaluating the network performance of the protocol, which is important since novel approaches often are presented without evaluating performance.

Furthermore, Chakraborty et al. [25, page 98] assumed that there will be heterogeneity in the Ubicomp environment as far as implementations and interfaces are concerned. We believe that this will be true in certain settings. However, we also believe that it will be in the interest of both service providers and service consumers to be compatible, to have as high degree of utilization as possible. In particular, we believe this to be true in market-driven situations. In fact, in these aforementioned market-driven situations, there might be a problem with too much compatibility or “false positives,” i.e., services or service consumers that pose as something they are not in order to gain advantages. We also note that Chakraborty et al. [25, page 98] shared our assessment for the need of higher level semantics in service discovery for Ubicomp.
3.2.2 Service Discovery and Cloud Computing

Service Discovery is essential to Cloud Computing. Mei et al. describe Cloud Computing as:

“Cloud computing is a paradigm that focuses on sharing data and computations over a scalable network of nodes. Examples of such nodes include end user computers, data centers, and Web Services. We term such a network of nodes as a cloud. An application based on such clouds is taken as a cloud application.”

–Mei et al. [107]

A key problem of the Cloud is how to discover services. Without service discovery Cloud Computing will not work, or is severely hampered.

3.3 Current Service-Discovery Technologies:

General approaches

Looking at the field of service discovery in general beyond ubiquitous computing, there are different solutions that cater to different needs. It is not until recently however, that the focus has started to come closer to the field of Ubicomp. To implement service discovery for Ubicomp, we first examine a selection of existing service-discovery protocols currently in wide use, to see if any of these can provide a solution or give valuable insights into the service-discovery problem. Several surveys of service discovery have been published. (For example: Golden [53], Helal [65], Zhu et al. [180].) Our examination will present a selection of the characteristics of the protocols. We discuss some shortcomings of these protocols, especially as related to service-discovery for P2P and ubiquitous-computing environments. In the following sections, we present some of the state-of-the-art in general service discovery.

3.3.1 Bluetooth Service Discovery Protocol

Bluetooth [19] is a short-range, low power wireless technology for cable replacement. In other words, it is intended to rid users of cable clutter and the need for frequent disconnection and reconnection. Bluetooth was invented by Ericsson, but is now controlled by the Bluetooth special interest group (SIG), today comprised of a large number of companies, including 3Com, NOKIA, IBM, Intel, Microsoft, and Motorola.

Bluetooth defines its own protocol stack, including a service-discovery protocol, SDP. This is based on unique identification numbers (UUIDs), with several predefined services such as Headset, Printing, Fax, and so on. After a device has been discovered, a Protocol Descriptor List in the
3.3. Current Service-Discovery Technologies:

General approaches

SDP service description is consulted to find out which protocols can be used to initiate contact with the device. Avancha et al. [7] show that the Bluetooth SDP can also be extended with semantic descriptions written in W3C’s Resource Description Framework (RDF) [173] or DAML+OIL [70] to support more expressive information about devices.

While not truly P2P in the sense of equality among peers (one is master the other is slave), Bluetooth does provide the ability to switch between these roles. The Bluetooth specification defines so-called piconets, which are groups of up to 255 devices. Only eight of these devices can be active at any given time, and the rest have to be in park mode. Several piconets can be connected into scatternets using a device acting as a bridge-slave. A device can be master in one piconet and slave in another.

3.3.2 Universal Plug and Play (UPnP)

Intended to make home networking easier, the UPnP [160] standard, geared towards both software services and physical devices, was developed by a forum led by Microsoft. UPnP builds on existing protocols and standards: DHCP and AutoIP for addressing; IP, UDP, TCP and HTTP for communication; and SOAP [170] for remote invocation. UPnP also defines additional XML-based protocols to support service-discovery: SSDP (Simple Service Discovery Protocol) and GENA (Generic Event Notification Protocol). Here we will focus on SSDP, as it has the most relevance for our purposes.

When entering a network, a device broadcasts a short advertisement containing its type, unique identifier, and a URL pointing to more information. These advertisements are stored by control points. Searching for a device is done by broadcasting a request for the desired type of device. This request is intercepted by all control points, and matching advertisements are sent back to the requester. Next, the service requester retrieves device descriptions and service descriptions of the devices found, using URLs embedded in the advertisements. These descriptions are based on templates defined by the UPnP forum. Only a few templates are defined however, among them Internet Gateway Device, Printer Device, and Lighting Controls. SSDP device descriptions may also have a URL to an HTML presentation page, which can be used to control the device or service and view information about it.

3.3.3 WS-Discovery

Web Services Dynamic Discovery is a multicast discovery protocol. The communication in the protocol is done via web services standards, of which SOAP-over-UDP is common.
A client sends a probe over the network for a matching service. If the client knows the address of the service, the client may choose to send the probe directly to the service. The service then contacts the client directly.

If there is a Discovery proxy that the client trusts, the client may enter into managed mode. However, if the client does not find a proxy that it can trust, the client will be in ad-hoc mode.

### 3.3.4 Zeroconf

Zeroconf is intended to create easy setup of devices and shared resources at home. It may for instance be file servers, scanners, or printers. No configuration should be required by the user.

Apple uses the Zeroconf approach to service discovery by using multicast DNS. Apple is also one of the major adopters of Zeroconf. When joining the mDNS multicast group on the local area network, every computer maintains its own list of DNS resource records. The DNS based service discovery is built on the Domain Name System, so the messages used are DNS-messages. In these messages there is information regarding what services are available, such as, service type, instance, domain name, as well as optional configuration parameters. A register of the different service types is centralized to DNS-SD.org.

### 3.3.5 Service Location Protocol (SLP)

In designing SLP [72], the Internet Engineering Task Force (IETF) aimed at IP-based networks, and this architecture relies heavily on TCP and UDP to determine the existence, location and settings of the services offered. There are three types of agents in SLP: User agents (UAs), Service agents (SAs) and Directory agents (DAs). UAs discover the locations and settings needed by the potential user of the service; SAs advertise the availability of services; and DAs act as brokers, caching information about services. The system can operate in two modes – with or without DAs. When operating without DAs, the UA will send a multicast request for services and will receive unicast replies. When DAs are present, SAs will attempt to register with a DA, and UAs will send all discovery requests to these brokers.

Service descriptions in SLP are very basic Service URLs that categorize service types. SLP strictly deals with discovery. What happens after that is not within the range of the SLP specification.

### 3.3.6 Jini

To ensure platform independence, Sun developed Jini [151] to run on Java. While this of course leads to a certain amount of platform independence, it unfortunately also creates the requirement that devices that want to use
3.3. Current Service-Discovery Technologies:

General approaches

Jini have to have a running Java implementation. When a service wishes to make its presence known on the network, it registers itself by uploading a proxy object to a lookup service. When searching for a service, the searcher sends out a multicast UDP request. After receiving the results for the search from the lookup services, the searcher can download the proxy objects and run them locally in order to establish contact with the desired service. Jini is one of the few service-discovery systems that rely on code mobility and serialization of (Java) objects. Jini also has other features such as a transaction server, but we will not describe these here.

3.3.7 Critique of current world descriptions

Currently, many of the popular service-discovery systems have very limited capacity for describing the world. In fact, many systems do not describe the world at all, but rather the services in it. This is most probably due to the fact that they are not programmed to take the world into account when performing service discovery, and that in itself is not surprising, since service discovery has been viewed as an activity that involves only the medium through which the services communicate, and the services themselves.

Herein lies a philosophical issue. Should a service discovery system take the world and users into account instead of only the service? There are arguments for positions of view. On the one hand, one might argue that service discovery by its nature only involves services and their properties. On the other hand, one might argue that the services exist in a context where the world, as well as the users, play a central role. It is therefore necessary to explore the border between services, users, and service discovery.

3.3.8 Critique of current service-discovery protocols

Universal Plug And Play (UPnP) [53] integrates a suite of such protocols, from Internet Protocol (IP) [31] and Transmission Control Protocol (TCP) for basic communications, up to Simple Object Access Protocol (SOAP) [170] and General Event Notification Architecture (GENA) [109] for service invocations and event notifications. Service Location Protocol (SLP) [72] is also IP-only, but uses its own formats instead of XML [172] for its various service-discovery messages. Jini [151] uses UDP [73]. Bluetooth SDP [19] only runs on Bluetooth networks. Other protocols have abstractions for the transport layer that allow the use of different infrastructures. Salutation [155], with its Transport Managers, is a case in point.

A service-discovery architecture for Ubicomp should be independent of network infrastructure, such as Bluetooth and TCP/IP, since there is no guarantee which infrastructure will be used by the different services.
and devices. Using something like UPnP or SLP would compromise this platform independence, something which we view as important in a Ubicomp setting. By separating the network infrastructure and the service discovery protocol, it is possible to achieve a high level of abstraction. Of the technologies we have examined, only Salutation meets this first goal without major modifications.

Bluetooth SDP does not require brokers, but connections have a master-slave setup, whereas peers in P2P networks are usually considered as fundamentally equal. UPnP makes use of control points, but these are usually directly connected to the device itself, instead of acting as proxies or brokers for many devices. Salutation’s Service Location Managers can be both local and remote, so both direct and mediated discovery is supported. SLP and Jini require the use of central repositories of service descriptions or interfaces.

Although this is done by for example JXTA Search [164], using brokers, or proxies, to mediate service-discovery requests is not suitable for ad-hoc and ubiquitous-computing networks. Of the examined standard technologies, Bluetooth, UPnP and Salutation meet this goal of P2P discovery.

Bluetooth SDP uses globally reserved unique identifiers for predefined service types. UPnP uses its own XML format, which allows more information about devices to be described; but these descriptions must be based on one of the (so far limited number of) existing templates agreed upon by the UPnP forum. Salutation has a similar approach with its Functional Units. SLP only allows a simple categorization of services based on its Service URLs. Jini describes services only from a programming perspective; its service descriptions are the Java interfaces to services.

3.3.9 Critique of current service descriptions

Most service descriptions today as well as current ontological approaches to the descriptions of services and devices, describe the service and device itself, but are lacking in richness. In previous work we have begun to address this issue [74] by creating systems that enable the service provider as well as the user, to specify a richer description of the service or device than what has been the case with traditional service discovery protocols.

Furthermore, a service does not exist in a vacuum. It exists in a setting, in a time and, in a place. The current service discovery protocols do not address this issue. Taking the setting and surrounding world into consideration is important because it influences which services we can and do use. Take the example of printing a document. Normally, the user is happy if s/he can find a printer that will accept the print request. However, the real world is different from the virtual world. If the service discovery system does not take into account that the user should be able to physically access the printer to retrieve his/her document upon printing, the ability
to use the printer is worth nothing. This might for instance be the case if the printer is in a locked room.

The evaluation of services from human perception of quality and security perspective is not part of current mainstream SD-protocols. However, some of them do address these issues from the perspective of the computer. Again, this is mainly due to the focus having been on ability to interconnect at all, which is understandable when an area is in its infancy, and when the said interconnectivity is lacking.

As the matter of interconnectivity is getting solved, it is becoming increasingly apparent that in order to increase the usefulness of the services, the systems need not only take into consideration the aspects of interconnectivity that are important to machines, but also the higher-level aspects of it that are important to humans. These aspects need to be incorporated into the SD-protocols. In earlier work we have also discussed user discovery [74], that is to say that users can be viewed as services from the service’s point of view. The service the user would provide, would be a utilization of a service, or consumption as it is often called.

In general, what high level aspects that are important to humans varies from situation to situation. However, there are some aspects of world knowledge that always are important:

- **time** Knowing the time is important since so much in human life is governed by time. For instance, normally work related tasks will not be performed before 08.00 in the morning, or after 17.00 in the evening. Furthermore it is usually not performed on holidays or weekends.

- **location** Where the user is located is also an important aspect. This will impact what the user deems as important services.

- **agenda** What comes next for the user is also of importance to the system. This is also true for what has happened already. From knowing what comes next, and what was done before, decisions can be made as to which services are important for the user.

- **activity** Discerning the activity of the user is of great importance to the service discovery system for it to have further information on what to use as basis for its work.

- **basic human facts** Knowledge about the person, such as their work and its implication is important. Furthermore, certain aspects of human life such as that people food, need to sleep and so on are also important. This should be contrasted to the more heavyweight approaches such as CYC [92] or MIT’s Concept Net [99].

---

*network speed, type of protocol, platform, and so on*
The common problem for much of what is described above is the so-called *grounding problem*. It is easy to become entangled in very heavy processing as a result of this. Instead, common-sense heuristics could be employed.

### 3.4 P2P in Ubiquitous Computing

Developing for a ubiquitous P2P environment requires new ways of approaching the development process. Mädche and Staab [103] present three dimensions that they claim will create a new paradigm of service-driven architectures: Information vs. activity; centralized vs. ad-hoc networks; and implicit vs. explicit semantic descriptions. The authors argue that ad-hoc configured, activity-oriented services with explicit semantics will offer new opportunities. On the downside, going from centralized to ad-hoc configurations and from implicit to explicit semantic descriptions, usually generates a higher degree of complexity in the system. However, this kind of migration is probably necessary if we are to support future dynamic and ubiquitous environments. Using JXTA to solve the ad-hoc issue, as we have done in our proof-of-concept implementation of our Enhanced Traditional Layer, is a good starting point. Several attempts have been made to marry the P2P and semantic web worlds.

#### 3.4.1 Edutella

This [114] is a metadata query infrastructure based on RDF, and layered on top of JXTA. Our system ODEn is also based on JXTA, so it is interesting to contrast ODEn with Edutella. The similarities do not run very deep, however. Edutella uses an approach similar to JXTA Search ([164]), that uses super-peers (called hubs in Edutella) to register query answering capabilities and to propagate queries. Querying takes place using a pre-defined query language and data model, and reasoning is not done on the service-seeking agent. As we have argued, these approaches are, not appropriate for service-discovery in ubiquitous computing networks. Edutella is not intended for discovering and using services, but for finding metadata about information providers. It has no explicit SOAP[170] / Web services Description Language (WSDL)\(^5\) [174] integration as ODEn

---

\(^1\)The problem of anchoring symbols to reality.

\(^2\)Resource Description Framework, a W3C Recommendation.

\(^3\)WSDL is a language for defining programming interfaces of objects and methods, in a way that is independent of the programming language, the objects and methods are written in. WSDL uses XML descriptions to achieve this language independence, and is being standardized by the W3C.
3.4. P2P in Ubiquitous Computing

does. Despite these difficulties, research [153] on using DAML-S[^10] service descriptions with Edutella. Still, Edutella is an impressive infrastructure for information integration in distributed systems, and its makers have shown how a multitude of different types of information resources can be aggregated and translated using its mediating wrapper peers.

3.4.2 Other efforts

Other developments focus more on the low-level details of P2P networks, such as routing of messages between peers. Schlosser et al. [139] present an ontology-based P2P Infrastructure for semantic web services. While JXTA relies on broadcasting to distribute queries and advertisements, and relies on rendezvous and relay peers to propagate these messages to other networks if necessary, the infrastructure presented by Schlosser et al. [139] partitions the P2P network into a so-called hypercube. All peers must implement specific algorithms for entering and leaving the cube, and for routing and broadcasting messages. Consequently, all peers share the responsibility for the integrity of this delicate topology, which may be a disadvantage. Furthermore, an outer hypercube partitions the network according to global ontologies, for example ontologies for service types. This approach can be seen as a more sophisticated version of the first step of our approach to service-discovery. That is, instead of using a hierarchy of JXTA names and the standard JXTA search mechanism, the authors use global service ontologies that partition a hypercube topology.

Schlosser et al. [139] claim superior scalability to very large networks for their approach, and it is possible that it is very useful for these situations. Indeed, it could be combined with the second stage of service discovery that we have suggested, that is, OWL and OWL-S service descriptions that are retrieved from the service providing peer and loaded into an inference engine on the client. However, their approach placed an extra burden on all peers, and the benefits must be weighed against this disadvantage. Many ubiquitous-computing applications that have been envisioned have a high degree of locality. Peers will mostly communicate with other peers that are physically nearby, and for these situations, a local broadcast may be sufficient. A similar approach has been used by Crespo and Garcia-Molina [29]. Their Semantic Overlay Network groups nodes according to semantic similarity.

[^10]: Darpa Agent Modelling Language has been carried out. However, the precursor to OWL-S.
3. Ubicomp & Current Systems
Chapter 4

METHOD

This chapter is divided into two major parts. Section 4.1 explores some of the possible methods for conducting the research, and addresses the methods employed. Section 4.2 describes the methods used using a step-by-step walkthrough of the research.

4.1 Methodology

Research in service discovery should be aimed at establishing how to better enable users to utilize the services available in the environment, and also to uncover the requirements of the creator/developer of the services. This concern is an important factor to emphasize, since much of the research effort so far has been on pure connectivity and system issues. Indeed, a lot of what researchers deem to be problems often differs from what practitioners in the field deem as relevant (Potts [131]), and vice versa.

4.1.1 Model and Systems development method

In the world of computing there are many scientific methods that can and need to be employed, such as psychological studies/experiments, methodology development/evaluation and tool development/evaluation. In previous research, we focused on a mixture of quantitative and qualitative methods (Ingmarsson et al. [77]). For the model and system development and construction part of this and current research, the method employed has been of the type explorative, iterative development [136], similar to Basili’s [14] building-experimentation-learning cycle. This method was selected mainly due to the fact that there are many uncertain factors when it comes to knowledge about current technology and their level of deployment, among users in particular but even among creators. In order to discover what can be done with today’s technology it makes sense to
4. Method

start by exploring current technologies, as well as the literature in the field. The next stage is to try to improve the most obvious drawbacks, add new solutions, arrive at a more in depth understanding of current capabilities and possible improvements, do a re-implementation, to be in a better position to go to the users.

In contrast to the above is the notion that in order to best capture the users’ needs, one way of starting would be to interview subjects as to what they would like from a solution. However, due to the current small saturation of general service-discovery systems in society, there are risks associated with going to the user too soon:

- We might be unclear as to what we are actually asking the potential users. This might very well be due to the fact that we ourselves and the users are uncertain as to what we want, and what we can achieve with the current system [27, 28, 33].

- The technology is quite new and unproven. Due to this, neither the developers nor the users know what to expect; that is to say they do not know the affordances [51] of the technology. This is another reason to explore the prototyping space in order to better understand the actual problems that face creators and users.

4.1.2 Evaluation method / Wizard Of Oz study of RESPONSORIA

After designing and constructing the model and the two more technically oriented systems, more user inclusion was needed. Therefore a final, third system was designed and constructed.

For the final system, RESPONSORIA, we chose the Wizard Of Oz evaluation method [81]. The Wizard of Oz evaluation method is a research method with which either a whole system, or a specific feature of a system may be tested without implementing the feature or system. Instead, the system or feature is simulated. The name Wizard Of Oz, comes from the fact that, unbeknownst to the test subject, the system or feature is acted out by an experimenter. Commonly, this is done by having an experimenter in one room acting as a wizard, and controlling all or part of what comes up on the screen where the research subject is situated.

In our experiment, we had the wizard and subject in the same room. The subject was informed that the experimenter was taking notes as well as pausing the simulation when necessary. We verified through control questions that the user did in fact believe the system to be the one constructing and presenting the services suggested¹.

¹The Wizard Of Oz study was cleared with the Ethics unit of Linköping University [75].
4. Method

4.2 Outline of the research process

This research has been carried out in seven major stages, each comprising several sub-stages. For an overview see Figure 4.1.

The seven major stages were:

1. Literature study and problem specification, as well as analysis of current service-discovery systems. See Sections 3.1.2 and 3.3.

2. Implementation of the Enhanced Traditional Layer in a system called OdE N. See Section 5.3 and 6.1.2.


4. Implementation of the Relevant Service Discovery Layer in a system called MAGUBI. See Section 5.4 and 6.3.

5. Evaluation and analysis of MAGUBI. See chapter 8 and OdE N.

6. Implementation and evaluation through the Wizard Of Oz study, of the third system RESPONSORIA. See Section 7.1 and chapter 7.


4.2.1 Literature study and problem specification

Stage one has four sub-stages:

1. An analysis of existing service discovery systems.

2. A literature study of the state-of-the-art in service discovery.

3. An analysis of Ubicomp in general, with an emphasis on the service-discovery perspective.

4. A synthesis of the preceding three items into a problem statement, as well as requirements on systems to reach a higher level of performance.

Figure 4.2: Method: Literature study and problem specification
4.2. Outline of the research process

4.2.2 Implementation of the Enhanced Traditional Layer in a system called ODEN, and evaluation of current common-sense research/systems

Implementation of the Enhanced Traditional Layer designated ODEN [41], based on demands from previous literature study and problem specification was carried out. The system was implemented using a combination of Java, JXTA, and Java Theorem Prover [47]. See Section 6.1.2 for details.

4.2.3 Specification of new revised demands, problems and scenarios

From Section 4.2.2 we get new requirements that lead to a proposal for and design of a new Layer, designed to compliment the first one, this layer also shifts focus closer to the users, whereas the first Layer had a greater focus on the devices. In order to more thoroughly penetrate the situation that the users are in, a number of scenarios were drawn up, analyzing service discovery through a combination of a mobile phone and an already present wall-mounted touch screen in the foyer of the Department of Computer and Information Science at Linköping University. Of the resulting scenarios one was selected for implementation.

4.2.4 Implementation of the second Layer

In stage four, the Relevant Service Discovery Layer, (MAGUBI) was implemented using a combination of Protégé, Jess, the scenario as well as the First Layer (ODEN). A more detailed scenario was drawn up, in which the system would help the user in a ubiquitous fashion when...
lunchtime approached, and make recommendations based on location, activity, and schedule, as well as external factors. Protégé was used to engineer an ontology that incorporated the necessary properties from the scenario. Jess was used to create a number of rules which acted as a reasoning system. The First Layer (ODEN) provides a rough list of available services that The Second Layer (MAGUBI) may act upon and further refine.

4.2.5 Evaluation and analysis of implemented layers

A critical analysis of both systems was performed. For a more in depth discussion of each system see Sections 6.1.4 (for The First Layer ODEN) and 6.3.6 (for The Second Layer MAGUBI). As a note on the method employed, it was beneficial to do a full cycle of redesign, which took into consideration the lessons learned from the first system while implementing the second, as well as addressing some of the issues discovered in the first one. The final analysis has led to future research agenda that utilizes the conclusions from the two first systems, while shifting focus away from the technical systems themselves and onto the users.

4.2.6 Implementation and testing through Wizard Of Oz study, of the Test Platform (RESPONSORIA)

The Test Platform (RESPONSORIA) was implemented as a web application. See Section 7.1 and chapter 7. For the backend, an enterprise grade server was selected. The following rationale were behind the selection: First, we wanted a system that was easy to maintain over a range of platforms, based on open standards and was also somewhat ‘future proof’. Second, by choosing tools that are aimed towards web development, we ensure that the backend of the system will run on a multitude of platforms. Third, by providing the system through a web server, we ensure that any device/system with
a browser capable of running Javascript will be able to run the system. RESPONSORIA is a system that is designed for Emergency response situations in which many different actors from different parts of the municipal government as well as private actors need to work together.

The scenario employed while constructing and evaluating RESPONSORIA was constructed by Pilemalm and Hallberg, together with professionals involved in rescue services [130]. This is also the case for a subset of the system requirements (Ibid). The remaining parts came from the results of implementing ODEN and MAGUBI as well as the analysis of those systems. To control the system and perform the Wizard of Oz study, a WoZ-control system was implemented. All of the above led up to the qualitative testing of RESPONSORIA. The testing was performed by running through the scenario provided by Pilemalm and Hallberg [130] while recording the spoken interaction of the session. Additionally, a semi-structured interview was conducted after the scenario was concluded.

4.2.7 Analysis of RESPONSORIA

RESPONSORIA was analysed qualitatively using Grounded Theory [30]: A scenario was run through and the sound of the whole session was recorded. In addition, a post experiment semi-structured interview was conducted, and the results coded and general themes identified. (See chapter 7 and Figure 7.11.)
4. Method
Chapter 5

AN ENHANCED MODEL OF SERVICE DISCOVERY

As our main contribution to the body of science is an enhanced model or method for service discovery comprising of two layers, and an evaluation of a complete system, we present the process for developing the layers, as well as the layers themselves in this chapter. We also discuss the motivation behind the evaluation platform. The evaluation and the evaluation platform are presented in Chapter 7. For an overview of how the different layers are connected and their current major components, see Figure 5.1.

After enhancing the service discovery system and process as we have done, it is conceivable to view a service discovery system as a expert system or as a decision support system. However, it does not fully mimic either of them. Instead, we would suggest using the term Decision Suggestion System (SUG), a modification similar to one suggested by Daniel J. Power [132], even though one can interpret Power as seeing the expert system as part of the decision support system. The reason for using the SUG acronym, is that our new type of more enhanced service-discovery system not only mimicks an expert to a certain extent by using expert system methods to try and select the correct service, but it is also mimicks a decision support system, by sometimes providing more than one service to choose from, and being part of a bigger system than only the service-discovery system itself.

5.1 The service discovery system development process & its challenges

One of the results from this thesis is a development process for service discovery systems. This process has six major parts: The first part involves
performing a large evaluation in line with Hallberg et al. [62]. The second part sketches the system on a conceptual level based on what has been learned in step one. The third part creates a mockup evaluation system. The fourth part performs an evaluation of the mockup to verify the conclusions from the second part. The fifth part involves performing a redesign based on the previous four parts. The sixth part implements the system using the knowledge gained from the previous parts.

5.1.1 Challenges in the development process

An important part of developing service discovery applications is knowing that the services discovered will be presented to the user in a non-linear fashion. Whereas certain applications have a clear cut flow, a service discovery system’s interface is probably non-linear in that it is likely that the developer will never know in which order the services will be presented to the user. It is reasonable to assume that a service discovery system will have a different workflow every time it is used, in the sense that it will present services in a different order.

A further point to be made is that there is a question regarding how the interfaces to the individual services are presented, and that in itself also affects the development process. There are three possible ways forward here:

1. Let the service offerers decide how the services are presented.
2. Force the services to be presented in a specific way.
Before we get to the third option, let us first consider that if the first option is selected, there is a danger that the interfaces presented are different in every instance. These differences which have both advantages and drawbacks, will not be addressed here. They are addressed in Chapter 8.

By using option 2 and forcing services to be presented in a specific way, limits are placed on what may be included in a specific service and what is allowed to be presented to a user. The advantage is naturally, that a greater recognition is ensured and hopefully will result in a greater speed in use as well as user satisfaction. A disadvantage is that there is a rigidity in what may be included in a service of a specific type and this in itself requires a high level of pre-approval and hands-on work to bring a service into the service discovery system.

3. Present standard features of services in a stipulated manner, but allow a free additional area of the interface to have non-standard settings and features.

Another challenge in the development process is knowing the granularity of the services that are handled by the system. In other words, how big are they? What do they cover? For instance, a service may be to move a vehicle 50 m south, or it may be to move it from one address to another. Knowing the granularity of a service has an impact on which service to choose. Yet a further challenge is to manually (or automatically) aggregate services of different granularity to create a complete service.

Together with the challenge of service discovery being a non-linear interaction, there is also a challenge in trying to prepare for services to appear and disappear. There is always the possibility that there will be new services that have unexpected properties, and it is important to design the solution in such a way that to the highest possible extent no assumptions are being made regarding what can and cannot be accommodated. For instance, having a model which assumes a maximum speed for roads on the basis that only gravel roads existed at the time of the construction of the system, would create difficulties when asphalt roads were invented, since parts of the system would be unable to handle or reason about the services offered by these roads.

5.1.2 Development in accordance with Hallberg et al.

In the beginning of the design process, we recommend a slightly adapted procedure closely related to the one followed by Hallberg et al. [62]. See Figure 5.2. It is comprised of the steps below:

A literature study To be prepared for the following steps, we recommend the designer to become familiar with the subject matter as well as the working procedures at the location in question. This includes
5. **AN ENHANCED MODEL OF SERVICE DISCOVERY**

![Diagram of the enhanced model of service discovery](image)

Figure 5.2: (a) Adapted method from Hallberg et al. [62]. (b) Our additional steps.

- **Learning specialized terminology.** This knowledge is essential to ask the right questions in the upcoming interviews and observations with the users.

  **Interviews with, and observations of the users**   After having learned about the subject, interviews are made with the parties involved. In these interviews it is important to establish:

  - How the procedures are currently done.
  - Who are involved in what procedures and as what type of agent, as well as in which role.
  - What are the tasks involved in the procedures.
  - What the different resources, that are involved in the procedures are. Based on these interviews and observations, services are listed in the form showed in **Table 5.1 on page 69**.

- **Design of a relevant scenario**  Based on previous knowledge and what services have been identified, a scenario is constructed together with the users/actors themselves, to get the highest possible ecological validity.

- **Design of a prototype**  Following the scenario design, a simple prototype may be constructed in a prototyping tool such as Adobe Flash Adobe Systems Inc. [3].

- **Light-weight scenario based evaluation**  After creating the prototype a light-weight scenario based evaluation is performed. In its most basic form the evaluation is run through by going through the scenario step-by-step and verifying that the services that are needed in each step can be ordered. The difference here, compared to a Wizard Of Oz evaluation is that the scenario-based evaluation has no wizard and uses the same
5.1. The service discovery system development process & its challenges

Table 5.1: Service listing example

<table>
<thead>
<tr>
<th>Definition of Service</th>
<th>Example of Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Hospital Transport</td>
</tr>
<tr>
<td>Description of service</td>
<td>Transport of individuals who are sick and/or injured. X number of individuals. No treatment is given during transport</td>
</tr>
<tr>
<td>Service attribute 1</td>
<td>Place where pickup occurs</td>
</tr>
<tr>
<td>Service attribute n</td>
<td>Time</td>
</tr>
<tr>
<td>Producer attribute 1</td>
<td>Number of individuals</td>
</tr>
<tr>
<td>Producer attribute n</td>
<td>Geographical area served</td>
</tr>
</tbody>
</table>

participants that were employed when designing the system to verify that the system indeed has been developed in accordance with the stipulated properties.

5.1.3 Additional four steps to enhance the development process

After using and adapting the steps outlined by Hallberg et al. [62], we add four extra steps which we deem useful in developing systems which contain service-discovery facilities. See Figure 5.2.

**Rules design & Knowledge base construction** From the interviews, observations and scenario construction, we first extract relevant knowledge about the domain, its tasks, and its participants. This knowledge is then formalized into rules that are used to describe the activities and decisions that the participants perform.

After creating the rules, a knowledge structure that may be accessed programmatically is created, using the rules as a basis for what to reason about.

**System design** When the knowledge gathering and light weight prototyping has been performed, the system can be designed. We stress that we do not believe that any coding in standard languages should take place at this point in the development cycle, but rather we suggest keeping development at a level that uses UML-diagrams or similar systems [121, 122].
Evaluation platform with Wizard of Oz evaluation Based on the system design, a compatible Evaluation platform is created that builds on knowledge from the Prototype as well as the System design. Ideally, the Evaluation platform can be an evolution of the Prototype. This Evaluation platform should be constructed with a Wizard Of Oz evaluation in mind.

To ascertain if the initial assumptions are correct, we strongly advocate the use of a Wizard Of Oz evaluation with the evaluation platform as a base, as well as using the previously constructed scenario. Using test subjects that if possible, were not involved in constructing the system and scenario is recommended. One possible outcome of the Wizard Of Oz evaluation is that there may be services that are deemed unnecessary as well as services that are missing. Another possible outcome is that the system is deemed to perform incorrect recommendations. If this happens, it is reasonable to assume that the world description or rules are incorrect in some respects, and accordingly adjust, add, or remove them as required.

Decision to develop full scale system The final step in our recommended procedure is to use the information gathered from the Wizard Of Oz evaluation to decide if the system should be developed or not. This must take into account any possible need to redesign the rules and/or the world knowledge. It is entirely possible that the perceived benefit from having a service discovery system may not be significant enough to warrant a full scale development.

5.1.4 Logs and maintenance Logs are a valuable tool, not only in the sense of logging for development purposes, but also in terms of providing a history for the users. Their purpose is thus dual. A history is important for the users since it provides them with information about what other users have done, and may also influence their decisions. Statistics derived from logs are also important to enable further development in the right direction, the statistics create a faster, more stable, and more accurate system.

The maintenance of service discovery systems of this kind is also a challenge. Barring the use of a self-organizing system, such as evolutionary algorithms, crowd-sourcing or neural networks, see Section 5.5.4, we are left with committees or a benevolent dictator. The need for a common platform is also expressed by Zhu et al. [180, page 83]. If we use the committee approach, which is similar to the World Wide Web Consortium W3C World Wide Web Consortium [175], we generally obtain a wide array of input, even though the implementation is left to the person responsible. The decision making process may take additional time, however while
5.2 The layers and the motivation behind them

When constructing this model, we decided to have two layers. This was motivated by elements we considered that the previous systems lacked as well as things we considered necessary for a model to work in reality. In the following subchapters, we first present the motivation behind the layers, and then the layers themselves.

5.2.1 Motivation for two layers

As mentioned in Chapter 1, there are several reasons for having multiple layers. The first reason is similar to the identified issues in the current service discovery approaches. The second reason is that we note that to produce a runnable, maintainable system, it is beneficial to modularize it in accordance with standard practice in building computer systems in hardware as well as in software [128]. The third reason is that by using a modular design, we may revise our system by adding, subtracting or replacing parts of it. This is especially true in the case when we find new aspects of service discovery that need to be taken into consideration.

Currently we have two layers, as can be seen in Figure 5.1 on page 66. These two layers are:

Layer 1: The Enhanced Traditional Service Discovery Layer

Layer 2: The Relevant Service Discovery Layer

5.3 The Enhanced Traditional Service Discovery Layer

In traditional service discovery, the focus lies clearly on the machine and its connectedness. In the traditional method establishing communication is paramount. These connections are established since the traditional challenge has been to connect machines in an automated and unobtrusive manner. Parallels may be drawn to the internet and the establishment of TCP/IP, as well as previously discussed subjects such as UPnP and USB. See Sections 1.1.2 and 2.5.1.

Compared to a normal Traditional Layer, Figure 5.3, our new layer first contains a more advanced way of representing and reasoning about the using the benevolent dictator approach gives us one clear gate which everything has to pass, and a more agile process, similar to that of the linux kernel development process, even though the final result may not be to everyones liking.
services. This more advanced way of representing the services as well as devices is required to perform a more advanced reasoning. We believe that using an ontology to enhance the representation is a step in the right direction. A normal traditional layer by comparison, in its most basic sense only contains information regarding that there is a device present at a certain address.

Second, we use a Peer-To-Peer system to handle the low-level connectivity issue. Ideally, this Peer-To-Peer system should be agnostic with regards to networks, and allow the user/programmer the freedom of not having to worry about such issues.

Separately, these two extra additions give some advantages, but together they produce a powerful combination. This combination occurs because with the additional logic in the Enhanced Traditional Layer, the system is able to not only do sophisticated service discovery, but it can also process advanced queries and return more sophisticated service discovery results.

What is lacking in this layer is a search that is based on relevance for the human being. This deficiency prompted us to develop the next layer.

5.4 The Relevant Service Discovery Layer

A major point that we wish to convey in this work is that it is not enough to merely discover services. The services discovered have to be important enough to be relevant to the user. This is why we developed the Relevant Service Discovery Layer. See Figure 5.4.

In addition to the Enhanced Traditional Layer with all its capabilities, the Relevant Service Discovery Layer also incorporates an expert system. This expert system has several abilities, chief among which is the ability to process results from The Enhanced Traditional Layer. This processing uses
information regarding the situation, the results, and the user, in order to eliminate the non-relevant search results in post processing.

Furthermore, the expert system also monitors the user as well as the surroundings for changes in variables that the system has access to. This includes, but is not limited to: Time, Schedule, Activity, Location and so on. This also holds true for the variables that are accessible with regards to the services themselves.

For The Relevant Service Layer we have also introduced a Proactive search that searches for relevant services based on the knowledge it possesses. This is different to the monitoring, in that the system constructs searches that it sends out to the Enhanced Traditional Layer.

### 5.4.1 Ontology in The Relevant Service Discovery Layer

For the Ontology in The Relevant Service Discovery Layer, we tried to identify what information would be necessary to enhance the service discovery experience. The information included may for instance relate to actors involved, items in the world that affect the actor and services, etcetera. An example of a small ontology from our example lunch scenario in Section 6.3.2 (page 96), can be seen in Figure 5.5.

### 5.5 User & World Model

Getting a more useful service discovery can take many forms. However, one classic way in computer science to enhance the ability to get a an increase in system performance is to provide it with more information.

It should be noted, that this increase in information in itself does not provide better performance automatically, since it needs to be combined with updated algorithms. These updated algorithms may also not add to
much execution time, nor CPU-load in comparison to the extra usefulness they may add to the user.

We have chosen to increase the amount of information available to the system by creating a User Model as well as a World Model.

### 5.5.1 User Model & its general rules

An important part of our two layers is the User Model. The User Model has a number of explicit and implicit assumptions. One assumption is that there are users, and that they have certain needs, wants and desires. What we do not assume however is that the user is always a human. The user may very well be a machine or other service. See page 76 for a more in depth discussion regarding human vs. non-human entities as service users.

It is important to recognize that in certain situations, a third intermediary may act as a facilitator or orderer of services. In the case of a rescue operation for instance, a third party (rescue coordinator), located in a situation room, may order a service for a service consumer, (rescue worker). Having this third party means that the service-discovery system must not only take the consumer of the service into account, but also the person/agent ordering the service. Thus, there are two aspects to the discovery of the service in cases when the situation is one where a third party is involved:

1. Ensuring that the appropriate service is something that the facilitator deems that the service user(s) need(s)/want(s), as well as an interface that the facilitator can use.

2. Ensuring that the service is something that the service user(s) need(s) / want(s).

Notice the subtle difference between what the service user needs/wants and what the facilitator believes the service user needs/wants.
Our User Model incorporates a number of properties that we deem to be useful in most instances of service discovery. The reasons to include these particular properties are derived from the earlier analysis of what services actually are. See Section: 2.1. Below, we will list those that we believe are universal. Naturally extra precision may be added by adding even more specific information for the task or environment at hand, but we must stress that this must be accompanied by adding an equally good selection of rules that are adapted to the situation and properties of the environment.

**History:** One way in which humans relate to their surroundings is through history. This history determines many things. One thing in that is particularly important for services, is what has been done before at a particular location. By knowing what has been done before and what is associated with this particular place at a particular time, it is possible to find out what is a likely service to be used at this location. When using a service that has a particular lay-out, such as a cell-phone tower, and considering that it caters to specific people that may travel a certain route daily, it is conceivable to envision that the service-discovery system would look for towers along this route to not only facilitate service-discovery, but also service utilization.

**Places:** Where a person has been is valuable knowledge for a service discovery system. The location may be linked to using particular services or types of services. There are also advantages to knowing what other people have been there and what services they have been used, especially if the people have similar profiles as the current user.

**Activities:** Knowing what activities have taken place and then cross-referencing them with either similar activities or finding out what types of services have been used by users that have done similar activities have used, is another way of getting to possible recommendations that are relevant to the user.

**Location:** A person’s location is immensely important when using services. Some services only exist in certain locations due to their physical limitations. A simple description of a place gives a lot to a service discovery system.

**Local time:** The local time is important since a person exists in time and time and time of day affect a person. (The same may be true for services.)

**Preferences:** Preferences are an individual’s long term wants or goals. Desires that do not change over time.
5. AN ENHANCED MODEL OF SERVICE DISCOVERY

**Needs:** These are things that *have to be satisfied*, at some point or at multiple points in time to have a working user. An example is food, which is needed at times by a *human* user to keep functioning.

**Agenda:** We have deliberately chosen the word Agenda. The word Agenda differs from Calendar in that it incorporates Preferences moving through time. A Calendar is a description of Events to take place, and the overall Agenda is more concerned with goals. However, there is a subtle difference between Goals and Preferences. It is possible to see Preferences as never changing goals. “I prefer Entrecôte to Filet Mignon”, compared to “I would like to order some tickets for the Opera”. The Preferences and the Agenda may be viewed as Explicit goals. There are also goals that are implicit, and they are closely related to Human knowledge. For instance, the bodily functions of a human must be adhered to and may be viewed as implicit goals.

**Human knowledge:** This is a difficult item to tackle. Many attempts have been made, for instance CYC [91], Mindpixel [146] and, Open Mind [150]. However, the approach we have used is to put this knowledge in this in relation to the other items in the User Model. As an example, we may use Local Time. The Local Time determines much of human behavior. For instance, it is known that people commonly sleep during the night. This knowledge is something that may be used to construct a rule regarding service recommendations. For instance that recommendations should not be performed when another rule indicates that the user is asleep. Processed together with an Agenda for instance, this becomes especially powerful.

As can be seen above, it is assumed that the general user is human. There is however, no principal limit in having a computer (or robot) as a user of the service discovery system. We have not explored this in our study, but our systems allows for either treating a non-human as a human, or adding additional information to be used when performing service discovery for non-humans, by including information about what type of entity is utilizing the system in the service-discovery requests. One aspect that one could consider with a non-human using the system, is that perhaps a rapid response with many services is to be preferred to a shorter list, since a non-human entity may rapidly utilize many services and be able to process information at a faster rate than a human.

5.5.2 World Model & its general rules

Another important part of the two layers is the world model and its general rules. Whereas the the user model deals with the internal and external
properties of the user, the World Model deals with the properties of the world.

**Model**

The World Model has Places as its major part. These places are located in space. However, we have also incorporated places that move, so-called Transport Places. These Transport Places, have all the properties that a Place has, but in addition, may move.

A place is not necessarily a point. It may also be a an area or a volume. In the real world, certain services are connected to certain places. For instance, a fishing service may be connected to a lake or part of an ocean. A service may also be contained inside a building. Services may also overlap each other’s Places. In addition, services may move from one place to another or be in motion. In the model, this is may for instance be represented with GPS- vectors, time, and if so, a required speed: \{Lat, Long, Height, Time, Speed\}.

Furthermore, a service may have a finite number of uses, such as a matchbox with matches or a river may run out of fish. At the same time, a service may also have a regenerative power if it is not used for a while or if it is underused.

**Rules**

The Rules that govern the world have to do with the World as it is, and many are derived to a certain extent from how the services are located in the world. For example, it is often the case that a service is related to a place, a time, have a duration, as well as a maximum number of patrons that it may serve at any given time. Naturally, there are also general rules that handle more general knowledge about the world, such as that it is not possible to be at two places at the same time.

### 5.5.3 Reasoning in the two layers

There are two types of reasoning and these are confined to the different layers respectively. In the Enhanced Traditional Layer, the reasoning is concerned with the properties of the services themselves and how to meet the search requests that come in. For instance, depending on the priority for different properties, the first layer will decide which service to select based on speed and quality, which is a common tradeoff in services. The reasoning in layer one differs from the second layer in that the second layer concentrates on reasoning about the world and the people/actors in it. In the second layer, the search request for the first layer is constructed as a
function of knowing the needs of the user as well as knowing the features of the surrounding world and the virtual world.

### 5.5.4 Alternatives to the ontologies and rules

Naturally, there are other ways of approaching the challenge of collecting, organizing, and drawing inferences and conclusion from the knowledge required to make useful recommendations for services.

Some possibilities that emerge are Case based reasoning, Neural Networks, Evolutionary Programming or pure algorithmic systems. Let us address these in turn:

#### Case based reasoning/classification

Using this approach, we first need to identify and retrieve similar cases from memory. Then the current problem needs to be mapped to the similar case and perhaps sub-sections of the problem have to be subjected to case-based reasoning in turn. After mapping, a test of the new solution has to be performed. This test may lead to judging the new solution as adequate or in need of more mapping/adaption. Finally, if and when an adequate solution has been found, it needs to be stored as a new case in memory.

An advantage over rule based reasoning is that we here have a certain ability to adapt to new situations without having to write new rules. This adaption may also take place proactively (Limthamaphon and Zhang [95]). However, case-based reasoning is by nature to a certain degree uncertain and this should be kept in mind.

#### Neural Networks

Neural Networks have to be trained. A major hurdle in applying them in the case of service discovery is that there is a high degree of flux in the environment. This flux leads to complications when training neural networks. A Kohonen network for instance, needs to know all the possible outcome classes to be able to train on it. It is possible that this training could be simplified by constructing more generic recommendations that would point the user to the generic service type. However, this would require additional descriptions in the services as to which type of generic service class they would belong. Inherent in the neural network model is also a lack of explainability or motivation as to why certain choices and decisions are made. However, Al-Masri and Mahmoud [4] have shown that it is possible to use Neural Networks in a service-discovery context.
Evolutionary Programming

In the case of Evolutionary Programming and using Evolutionary Algorithms for service discovery, we believe that this is an interesting way forward. It does however bring with it some caveats. For instance, it is very likely that the system would have to be trained in a similar manner to Neural Networks. In fact, there would have to be a certain amount of constant training due to the ever changing structure of what services are available, are removed, and are added.

Faulty recommendations are to be expected in the beginning of the lifecycle of such a system, and the users would have to be aware of this. Likewise, a certain degree of faulty services would be suggested. Analogous to the Neural Networks, there is no clear cut path nor explanation as to why a certain service is recommended and others not, except that it has been selected over time. Even though decision trees may be produced if requested, there is a high likelihood that that would not make sense to the users. However, if we try to to adapt the work of Bonakdarian [20] it may be possible to arrive at more understandable evolutionary algorithms with a higher amount of explanatory as well as predictable value and behaviour.

Pure algorithmic systems

By pure algorithmic systems we mean systems where the algorithm and the data are contained in the same code base. This is often also combined with a low level of abstraction. If a purely algorithmic system is to be used, all the algorithms must be programmed and maintained. Maintainability is a major factor in software development, and in systems such as service discovery systems, where it is almost certain that a part of the work describing the services has to be done by the services providers themselves, there is a need to provide simplicity.

If a purely algorithmic system was to be used, it is conceivable that the algorithms would be too complicated to write for the layperson. Furthermore, the addition of the facts to the knowledge in the system would be further complicated by having to employ a third party to add these, especially in the case of maintaining the algorithms, even if only small changes were needed.

It is obvious that this approach also wanders into NP-completeness. From a maintenance standpoint, it is easier to split the system between knowledge and rules and apply general rules to a changing set of knowledge.
5. AN ENHANCED MODEL OF SERVICE DISCOVERY

General observation regarding explainability or traceability

Arguing for Neural Network’s, Evolutionary algorithms, and to a certain extent Case Based reasoning, it should be noted that there has not been any request, nor attempt from the user’s side to find the motivation for the recommendations from the system. However, the system was never perceived as giving completely faulty or wrong recommendations, and thus, we cannot make any predictions as to how the users would react if this was the case.

5.5.5 Rules and how they should be interpreted

From the above, it can be observed that it is possible to end up with many rules. This vast amount of rules needs to be organized in such a way that it does not require massive re-editing when rules are added or subtracted.

Thus, a recommendation for the type of rule-based reasoner that is to be employed emerges: It is advantageous if the rule-based reasoner that is to be employed in a service discovery system of this type, is one that does not require any special ordering of the rules.

5.6 Evaluation Platform

In our Model/Method for service discovery, we also include an Evaluation Platform. We advocate carrying out an Evaluation of the proposed service discovery system using the Wizard Of Oz methodology. The reason for this recommendation is to catch assumptions that may have been made early on in the development process.

A major contributing factor in deciding to include an Evaluation Platform is that even though users have been involved in developing the system [130] that we subsequently tested, the users themselves were not able to predict how they would react when presented with the services that they themselves requested. This is not uncommon in user-centered development, but highlights the need to simulate and test a reasonable extent what happens when the user gets what he or she wants.

Furthermore, it is unlikely that even the user expert or developer knows beforehand how other users will react to getting the system they want. In our case, we assumed that users would be satisfied by getting the suggested services. However, as shown in chapter 7, there were additional facts that emerged when “perfect suggestions” were presented.
In this chapter we, present our two layers in the order they were created. Having presented the layers in Chapter 5, we implement them all in proof-of-concept systems. (For an overview of how the layers fit together see Figure: 6.1.) The first layer, (The Enhanced Traditional Layer), is realized in a system called Odení. As the first layer was created to address features that were identified as beneficial to service discovery, its implementation, Odén, is presented in conjunction with information regarding ontologies and the Ontology Web Language (OWL). Odén is discussed in Section 6.1.

The second layer, (The Relevant Service Discovery Layer), is realized in a system called Magubi, and it continues where Odén leaves off. Magubi acts as a common-sense post-processor on the results from Odén, and may in addition initiate searches. The common-sense and post-processing is

![Evaluation Platform Diagram]

Figure 6.1: Our two new layers are shown with back background in the Figure.
done through the use of ontologies and a Rete-based [49] algorithm. Section 6.3 covers MAGUBI.

With these two layers in place, we explored the possibility for evaluation. We subsequently created an evaluation system called RESPONSORIA. RESPONSORIA is a web-based crisis-management system, built on top of MAGUBI\(^1\). RESPONSORIA is built using the SOA-approach and in addition to MAGUBI and ODEN also uses an established enterprise system for the back-end. Being web-based allows it to run across a wide range of internet connected devices, including smart phones and tablets. RESPONSORIA is presented in Section 7.1.

### 6.1 Exploring the Enhanced Traditional Layer using ODEN

In ODEN, we enhance service discovery using a P2P framework combined with semantic models of services using OWL [171]. OWL has its roots in the Semantic Web [17] and Description Logic [8] fields, and can be used to create ontologies to represent many different kinds of knowledge. An ontology in computer science is usually defined as an *explicit specification of a conceptualization* [57] of a domain. For our purposes, we can use ontologies to describe a shared conceptualization of the domain of services, devices and other concepts that could influence the service-discovery process, such as different kinds of contexts [140]. Using ontologies enables service-seeking peers to reason about available services and devices, and make intelligent and informed decisions about which services to use, and how.

#### 6.1.1 OWL

OWL is a standard representation language for ontologies, and as such has good tool support. Moreover, the OWL Services (OWL-S) [171] ontology is written in OWL, providing further motivation for using this representation.

The OWL-S ontology for semantic web services is the basis for our work. It provides a set of concepts for modeling some aspects of services. For example, it lets us model inputs and outputs, preconditions and effects, and the relations that different processes have to each other. However, a more comprehensive ontology is needed for service-discovery in ubiquitous computing in general, since OWL-S does not include concepts for *device capabilities* and *context* [41]. We will also demonstrate how such an extended ontology can be integrated into the P2P network.

\(^1\)and therefor also ODEN
6.1. Exploring the Enhanced Traditional Layer using ODEN

Figure 6.2: ODEN system hierarchy.

6.1.2 ODEN

Our experimental research platform, ODEN, uses ontologies expressed using OWL, and elaborates on OWL-S’ concepts for the semantic modeling of services and devices. ODEN integrates these ontological descriptions into the P2P-system in a way that preserves the fundamental P2P characteristics that we believe are important with respect to independence of network infrastructure, and direct P2P discovery.

While the P2P-system handles basic networking duties, service ontologies enable reasoning and the evaluation of services. ODEN has two layers: the low-level communications infrastructure (P2P-layer), and the high-level ontologies and reasoning. These two layers are largely independent of each other. See Figure 6.2. We could rewrite the ontologies without changing the way in which they are integrated into the P2P-layer, and we could replace the P2P-layer and still use the same ontologies. This independence is an important feature, as both P2P systems and ontologies are rapidly evolving fields.

The following subsections will discuss discovery-enabling ontologies, how ODEN integrates such ontologies into the P2P-layer, and describe in detail how the service-discovery procedure in ODEN takes place.

Ontologies and Reasoning

Service-seeking peers must be able to evaluate service descriptions, compare their pros and cons, and deduce facts that might be needed to make a decision as to which service to use, and how. To evaluate OWL ontologies, peers need an inference engine. Several inference engines are available.
The peers we have implemented currently use Java Theorem Prover [47] (JTP), as it offers a good combination of usability, performance and documentation. It is possible to substitute this for other inference engines if necessary. In fact, all evaluation in ODEN takes place at the service consumer, so different peers can use different engines and completely different methods of evaluating services.

The service descriptions in ODEN are currently based on a hierarchy of OWL classes, shown in Figure 6.3, based on the OWL-S Profile class. OWL-S encourages subclassing the Profile class to present properties specific to particular types of services to agents that wish to evaluate the service. We introduce a class called DeviceProfile, with properties common to all types of devices. For example, each DeviceProfile can have several DeviceConfigurations, each of which can, in turn, hold several ServiceParameters. This design lets us express different combinations of parameters that the device supports. For example, a printer may have a high-resolution black-and-white configuration, and a low-resolution color configuration.

Further subclasses of DeviceProfile define properties specific to different types of devices. As an example, we have created a PrinterProfile class which includes properties for paper size, resolution, as well as other properties. New subclasses can be added for any type of device, and relations between device types can be expressed using subclassing and
other OWL constructs (e.g., disjointness, unions).

Communications Infrastructure

In O\textsc{den}, each peer represents a device or service. Rather than having a classic client–server approach, O\textsc{den} uses peers that can both consume and provide services, and it does not make assumptions about pre-configured servers. Our current P2P-layer uses J\textsc{xta}, which was designed specifically for ubiquitous and ad-hoc networks, and can handle these issues using abstractions such as peers, peer groups and advertisements. O\textsc{den} builds on the basic peer discovery mechanism in J\textsc{xta}; that is, advertisements and searching for service names. O\textsc{den} augments this mechanism with a layer of semantic information. We define two important additions to service advertisements that peers providing services using O\textsc{den} must implement:

2. Pointers to OWL files that describe the service (Figure 6.4).

Furthermore, service names must follow a common hierarchy. This hierarchy maps one-to-one to the OWL-S Profile hierarchy described in Section 6.1.2, as shown in Figure 6.3. OWL-S uses WSDL interface descriptions to provide a concrete grounding to its abstract descriptions of methods, so our use of WSDL follows naturally from our choice to use OWL-S. Peers wishing to evaluate a service need to retrieve the OWL-files that describe the service. This remote communication is also kept language-independent by using SOAP [170]. SOAP is also an XML-format, and an associated protocol for remote invocation also standardized by W3C. To use SOAP in the J\textsc{xta} infrastructure, O\textsc{den} uses a J\textsc{xta}–SOAP bridge that embeds the SOAP messages in J\textsc{xta} messages. This bridge builds on existing work, and has been extended to better interact with the other parts of O\textsc{den}. Integrating WSDL and SOAP into J\textsc{xta} is straightforward, as J\textsc{xta} uses open XML messages and advertisements, and is not committed to any particular standard for remote communication.

The goals of the P2P infrastructure J\textsc{xta} [119], (short for “juxtapose”) are interoperability, platform independence, and ubiquity. J\textsc{xta} does not in itself contain service-discovery facilities. Through the J\textsc{xta} protocols, peers using different transport protocols and hardware platforms, and programmed in different languages, can interact with each other. Currently, J\textsc{xta} has support for TCP/IP and HTTP networks. Java is the language used to write the J\textsc{xta} reference implementation. There are also versions for J\textsc{2me} [156] (Java 2 Micro Edition), a very light-weight C language implementation suitable for embedded devices [158], and several other implementations are in development.
Figure 6.4: A service advertisement in ODEN. The name places the device or service in a hierarchy; the WDSL field provides a programming interface to the service; and the OWL-S tags point to detailed semantic descriptions, retrievable from the service-providing peer.
In JXTA, peers are organized into peer groups. A peer group can be used to represent the context of peer interactions, types of service, current state, location, and so on. Representing this type of context is of the utmost importance in ubiquitous computing [140]. Any peer that wishes to make a service available on a JXTA network needs to create an advertisement of the service. An advertisement is a small piece of XML data that announces the existence and some properties of a peer, a peer group, or a pipe. The peer then needs to publish the advertisement. Publishing an advertisement allows other peers in the same peer group to find it by using a standardized search mechanism, until the expiration time of the advertisement has passed. At that time, the service provider should publish a new advertisement if it still wishes to provide the service. When a peer finds an advertisement, it usually puts it in its local cache. Other peers can then retrieve it from there as well as retrieve the advertisement from the actual service provider. This mechanism provides additional redundancy and scalability of JXTA networks.

Advertisements in JXTA have a string Name field. When a peer wants to locate other peers, it can use these names to guide its search. If we adopt the convention of using colon-separated strings, such as

```
Device:Printer:Printer1
```

or

```
Service:E-shop:UsedComputersInc
```

for the advertisement names, they can be used to indicate what type of service the peer provides, in a hierarchical fashion. The discovery mechanism in JXTA also allows * as a wild card, so for example a peer could search for

```
Device:Printer:*
```

in order to find all peers with names starting with Device:Printer.

However, the name of a device or service often does not provide sufficient information. Until there is a globally agreed-on hierarchy of devices, the search string above would perhaps result in a few matching printing services, but other printers could be named for example Hardware:OutputDevice:Printer, or something entirely different, and these services would not be discovered. A user looking for a used PC may be happy to find UsedComputersInc, not knowing that it only sells Macs.

**Service Discovery Procedure**

The procedure for a peer to find, evaluate, and use a service in OdEN is as follows:
6. EXPLORING THE SERVICE-DISCOVERY LAYERS

Figure 6.5: The PDA-client starting in ODEN. It starts by loading OWL-files into its JTP knowledgebase, starts the PDA peer for P2P-networking and finally creates and joins the Office peer group.

1. Retrieve advertisements from other peers that are of the correct type, e.g., Device:Printer::*.

2. Retrieve the OWL files, indicated by the advertisements that describe the services provided by these peers. Calling a getFile SOAP-method that all service-providing peers in ODEN must implement retrieves the files.

3. Load the OWL-descriptions into an inference engine. Evaluate the data and decide which service(s) to use, if any.

4. Extract the WSDL-interface description of the service from the services advertisement, and call the SOAP-methods described therein in order to invoke the service.

6.1.3 ODEN running

We will illustrate a run of ODEN through an example. We start by running the simulated PDA-client that we will use to find the service we want. As can be seen in Figure 6.5, the PDA-client starts by loading the OWL-S files and the corresponding OWL-S profile structure. If there already exists a peer-group for the Office (created by other peers), it is joined at this point in time. If no such group exists (such as in this case), then the PDA-client creates such a group.
6.1. Exploring the Enhanced Traditional Layer using ODEN

Figure 6.6: Simulated PDA-client GUI after start. Note that there is no device constraint selected in the left panel. The middle panel shows results when a search for services is done. The buttons to the right shows the corresponding information about the selected service.

Figure 6.7: The search for Printer panel in ODEN. To the left (A) can be seen the different properties that can be selected for the current device (in this case a printer), and also a column of numbers to the immediate right (B), where the user can set the priorities for the different properties. To the furthest right (C) are a number of buttons which when pressed bring up the corresponding information.
6. EXPLORING THE SERVICE-DISCOVERY LAYERS

Figure 6.8: Printer1 log. Upon a service search, the different devices/services send the profile, process, service, and grounding information.

Now, the PDA-client GUI appears (Figure 6.6). Currently, the system does not support searching for a generic Device:*:. By changing the mode of the PDA-client through selection of Device:Printer:*: from the drop-down button, the interface changes to reflect the possible search alternatives that exist for this type of device (see Figure 6.7).

We may now select our desired printer criteria (e.g., printer type, paper size). To the right of each of these criteria we may also specify their priority. For instance, it might be very important to get the printout in color. In fact, it might be so important to get the printout in color that we are willing to accept a reduced resolution. Another possible example situation is that we need the printout rapidly and we want it to be in color.

When the search button is pressed, the PDA-client peer starts by looking for advertisements that match Device:Printer:*:, in other words all Printers. The next step is to use these advertisements to download the OWL-S files from the devices themselves. See Figure 6.8. When these files have been received, they are injected into the PDA-client’s JTP database, after which the reasoning is performed and each result scored.

In Figure 6.9 we can see the result of a search where we have placed emphasis on color and ability to print transparencies. If we place a priority of 10 or higher on a certain attribute, we have chosen to interpret that as a necessary attribute. In other words, if a priority of 10 is selected for the attribute color, Printer1 will score 0 since it lacks the ability to print color, even though it might fulfill the other attributes.

6.1.4 Discussion of ODEN

A significant feature of ODEN is that the service consumer controls all evaluation of services. A broker or service-evaluation service could still be implemented to help resource-constrained peers find and use the services they need, but the choice to use such brokers is up to the service consumer, not to the service provider. Our service discovery architecture itself does not assume such a broker.
6.1. Exploring the Enhanced Traditional Layer using ODEN

Figure 6.9: Results from ODEN when we are prioritizing color. A higher number to the immediate right of the property indicates a higher priority for that property.

A second important point to note is that a service-seeking peer first filters out potentially interesting services using JXTA’s name-matching search, and then loads and evaluates only the services that match. This two-step process reduces network traffic and the amount of work the service-seeking peer has to do, since a smaller number of advertisements have to travel the network, and there is less data for the peer to evaluate. The first step is a coarse filtering, and the second, using the ontologies, provides precision.

As we mentioned in Section 6.1.2, (p.85), this process assumes a global namespace of devices and services. This is, of course, difficult to achieve, but there are different ways of alleviating this issue. At one extreme, the first step can be skipped completely, which happens if peers simply search for * instead of, for example, Device:Printer:. This may be feasible if the contexts of discovery, i.e., the peer groups, are sufficiently small. All evaluation of services would then be performed using the ontologies. This solution would be highly robust, but as we have mentioned, it may give the searching peer far too many service descriptions to evaluate.

At the other extreme, the namespace hierarchy can be extremely fine-grained, so a peer would search for a specific device. This approach will give the searching peer exactly what it was looking for, provided that it knows exactly what it wants, and that the hierarchy is stable. These are very strong provisions however, and this solution is very brittle.

We believe that the optimal solution lies somewhere between these two extremes. The exact layout of the namespace hierarchy is still an

---

For example: Device:Printer:LaserPrinter:HP LaserJet:*
open question, but it should only provide broad categories sufficiently general to avoid the brittleness of a too specific approach, and still be sufficiently specific to avoid flooding the searching peer with irrelevant advertisements.

A third issue we wish to mention is the concerns that have been stated regarding the performance of JXTA [61, 24]. It should be noted, however, that the JXTA platform is under development, and these problems are being addressed. Furthermore, our concern here is not maximum throughput of data between peers, but to build an architecture for service discovery. The evaluation of service advertisements, using an inference engine, currently takes much longer than retrieving the advertisements, so the speed of network traffic is not the most critical issue in this context. Using an inference engine locally has advantages and disadvantages. We put extra strain on a “weak” computer when we run heavy reasoners on it. On the other hand, we are not dependent on the network in order to evaluate and reason about services.

6.1.5 Observations regarding ODEN

It is worth noting that being JAVA-based, ODEN, is able to run on a multitude of platforms. This ensures a great deal of portability, although Java demands a certain amount of resources. Modularity is another property of ODEN. It is possible to plug parts in and out of ODEN at will. Thus, it is possible to substitute JTP, for instance, for Racer [59] or FaCT [69]. As mentioned in Section 6.1.2, (p. 85), the current P2P-part of ODEN – JXTA – is available in a Java 2 Micro Edition as well as in a light-weight C implementation, both of which lowers resource requirements. JXTA is also open source. Relying on XML-based formats for the descriptions makes for a heavier solution, but this is offset by the advantage of our being able to use general tools readily available, such as Protégé [83].

In terms of reasoning and extracting information about services, ODEN is able to use the power of JTP [47], which puts it beyond simple matching algorithms or profiles such as in Bluetooth [19]. ODEN is not able of taking the initiative, and in this sense we are left with a passive service discovery system. This passiveness is one of the issues that indicates the need for an additional layer in service discovery for ubiquitous computing. Related to this is the issue of planning. Sometimes in order to be able to utilize a certain service it might be necessary to plan in several stages. At other times, use of a service might seem straightforward, but in reality this might not be the case. An example of the latter might be the erroneous decision to allow the user to print to a printer behind a locked door. So far, we have not conducted large-scale experiments with ODEN as it is still under development.
6.2 Observations regarding the use of CYC-like systems in developing service discovery

As exemplified by ODEN, (see chapter 6.1), it is useful to have an advanced ontology for service discovery. Unfortunately, in its current form CYC (see Section 3.3.9 on page 53 and 5.5.1 on page 76), is not a viable solution at this point in time for our purposes.

It is clear that common sense can be useful in the service discovery process, but it is also clear that using a self-developed ontology and reasoning system is more appropriate for our purposes than CYC. However, lessons learned from CYC have influenced the design of our own system. The bulkiness of CYC, together with the large number of tools is one of the main reasons that it is difficult to adapt it to our purposes. On the other hand, the comprehensive nature of CYC and all its tools were what attracted us in the first place. In this particular case, it was the ability to reuse an ontology once it has been created. As we have seen, the possibility for reuse of CYC was low in our setting. This is partly due to CYC’s size, and partly due to lack of familiarity with the tools, as well as a lack of an interface that facilitates the familiarization and exploration of CYC. It is important to remember that there is a point to having common-sense knowledge if we want to function in the world. In the words of Marwin Minsky [112]:

For each different kind of problem, the construction of expert systems had to start all over again, because they didn’t accumulate common-sense knowledge.

The difficulty lies in knowing when an ontology is reusable or extendable and when it is time to start over [135]. Some implications for designing Ontologies for Service Discovery in Ubiquitous computing can be found in Chapter 8.1.4.

6.2.1 Decision: Building our own ontology for our domain

Initially, we were very optimistic about CYC’s potential for service discovery for ubiquitous computing. However, after lengthy evaluation and initial prototyping, we arrived at the conclusion that the potential gains from using CYC would not outweigh the amount of work that had to be done adapting CYC to our domain. We believe that in its present form, CYC is not appropriate for use in our service discovery domain. This is due to a number of factors:

- The complexity issue. Even though Cycorp has tried to get around the complexity issue by creating what they call microtheories, in
reality there still exists the need to limit, when it comes to the actual search and inference, so that it stops within a finite amount of time. This limitation of the search space is due to the huge amount of data available, and also due to the fact that the branching factor is large.

- Where to start searching. Due to the vast amount of knowledge available, it is not easy to know where to start searching. Which microtheory should be activated? How do we know we have the correct synonym for an expression? This uncertainty may lead to frustration for the ontologist/programmer.

- Difficult to penetrate API/application. CYC’s sheer size and amount of functions makes it a daunting task to try to use the correct features. The fact that it is difficult to get started with CYC is also a major drawback. This is corroborated from other users as well [104]. Furthermore, there is a lack of comprehensive documentation. There are some tutorials available, but not to the extent that would be required to fully understand the system.

- Missing feature. A feature was missing in CYC which was bug-reported to Cycorp. To be more specific, the planner was non-functional. This was a major drawback since planning, or the knowledge of what must come before and after, is one of the most important tasks in a common-sense scenario.

The first two points are central in that they concern fundamental issues in the nature of CYC-like systems. The final two points concern engineering/implementation issues. Productive reuse of code and information is generally beneficial to development. However, as shown by our investigation into the world of CYC, there are issues that arise when the mass of knowledge and code becomes very large, and these issues need to be addressed. We concluded that in the specific case of CYC, it was too big, too convoluted and too complex for us to be able to integrate into our project within a reasonable amount of time, which forced us to seek an alternative solution: In our specific case, we developed a solution ourselves. If, however CYC came with a large library of demo applications, complete with code, both of which demonstrated its different capabilities, it is likely that this would enhance its proliferation.

6.3 Exploring The Relevant Service Discovery Layer using MAGUBI

MAGUBI is a system designed to work together with Oden (Figure 6.10) and to act as a common-sense post-processor by filtering out any possibly
6.3. Exploring The Relevant Service Discovery Layer using MAGUBI

![Diagram of system hierarchy with GUI, MAGUBI, ODEN, JXTA, JTP, JAVA VM]

Figure 6.10: The location of MAGUBI in the system hierarchy. MAGUBI acts on-top of ODEN filtering results and generating queries based on world knowledge.

inappropriate results that ODEN might generate. MAGUBI can also initiate service discovery through ODEN on behalf of the user. This modular design is advantageous in that the design, like the rest of the architecture, is interchangeable. There are some benefits of having MAGUBI on top of ODEN, for example:

Consider users who would like to print documents. A service discovery system like ODEN can find a printer that can print according to the users’ requirements. However, this does not take into account actually retrieving the printed document. At the MAGUBI-level, there is information that printers must be physically accessible to the users. For instance, systems like MAGUBI can avoid printing to printers behind locked doors.

6.3.1 Scenarios and use cases

Working with scenarios is important since it gives us richer understanding of what the user might need in a particular situation. Consider an office foyer setting where people with different goals, personal desires, and equipment, are coming and going all the time. In this foyer, we have a wall-mounted touch screen. The start state for the screen is to display a welcome screen with the time, the coming public schedule of events, as well as two buttons that allow the users to either search for people or rooms. To illustrate the potential, we construct a few scenarios:

**Morning** People are arriving at work. They are interested in their immediate schedule, especially to find out if it has been changed, as well as their high priority messages and e-mails.

**Lunch** People go out to lunch. At this point in time, food-related information as well as traffic information is interesting to them.
6. EXPLORING THE SERVICE-DISCOVERY LAYERS

Getting off work People want to get home as quickly as possible. Traffic flow, as well as bus and train timetables are interesting at this point, along with any errands that they might have to do.

Business trip Upon commencing a business trip, it is important to know whether or not the method of transportation is working correctly. Is the train running on schedule? Has the flight been cancelled? Will they be able to reach the train station in time with a bus or do they have to call a taxi?

Colleagues Whether or not a colleague has just passed by a co-worker physically, or has not yet arrived, might be important/useful information for said co-worker. (Imagine a meeting that both are scheduled to attend.)

New employees New co-workers have a different set of needs and wishes compared to people that have worked there for a long time. The layout of offices as well as contact information and a useful guide to current events and how to get there are important.

Visitors When people are a temporary visitors in an area, they have needs similar to the new user even though the emphasis will most likely be on current events. However, their rights, such as access to rooms, may differ considerably.

To illustrate the principle, we have chosen to work more in depth, running the service discovery system with the lunch scenario.

6.3.2 Lunch scenario up close

Howard is working a usual day at the office, and as the clock approaches noon, he gets up from his desk, gets ready to go to lunch, and starts to leave the building, see Figure 6.11. As he passes the big information screen just inside the entrance, it provides him with a recommended restaurant for lunch. He gets into his car and drives to the restaurant.

What has actually happened here? The system has knows about Howard and his activities. It knows who he is, where he is and necessary temporal information, such as his daily routine, calendar, and text-correspondence. Using this information, the system can make predictions and recommendations. This information is collected through simple, ubiquitous means.
6.3. Exploring The Relevant Service Discovery Layer using MAGUBI

Figure 6.11: Howard going to lunch. His Bluetooth cellphone provides his identity. ODEN and MAGUBI work in concert to provide a suitable recommendation knowing his preferences and that he has access to a car.

Figure 6.12: One of the touch-screen entrance information terminals at the Computer and Information Science department at Linköping University. (Far right of the picture.) The terminal provides basic agenda information as well as contact and location information to persons, events and locations.
Using a cell phone and Bluetooth beacons, we can get access to all the required information. The cell-phone provides us with a unique ID. The Bluetooth beacon provides us with a unique location.\(^3\)

The motivation behind the lunch-scenario can be found at one of the campuses of Linköping University [98]. At the Computer and Information Science Department [154], there are integrated wall displays at the entrances. See Figure 6.12.

Normally, these displays are used by students or visitors to find information about the agenda, rooms, or employees. However, for the employees themselves, this is an under-utilized resource and could provide extra information throughout the day. Since most employees and visitors carry a cell phone with Bluetooth, it is logical to try to utilize the already existing infrastructure. This is true in most other settings too, since ownership of a Bluetooth cellphone is pervasive in most parts of the world where Ubicomp is gaining ground. The employees usually pass the display when they arrive, at lunch, and when going home for the day. Most of the time, the employees do not use the displays at all. If the displays could be made to show useful information when the employees pass without any requiring extra effort, then a ubiquitous and pervasive use could be achieved. Another reason for choosing lunchtime is that most people have an interest in knowing about lunch alternatives as well, as in getting recommendations around lunchtime. This scenario can be generalized to any screen that people pass with their Bluetooth cell phone.

6.3.3 MAGUBI system architecture

MAGUBI can be seen as a post-processor of the results from ODEN, as well as an initiator of service discovery requests on the users’ behalf. MAGUBI is comprised of two parts: (1) an ontology created in Protégé from the information in the lunch scenario, and (2) a set of rules in Jess [50] that operates on the said ontology with the aforementioned scenario as a base.

Ontology up close

As can be seen in Figure 6.13 there are several classes that support the scenario in Section 6.3.2. We will touch upon some of the available information in the different classes. The class LunchGuest (LG) is at heart of the system. It contains information about what food and restaurants the LG likes and dislikes, the last known location of the LG, what restaurants

\(^3\) Nilsson et al. [117] showed that an accuracy of about 1.7m ± 1.7m is possible. Using Bluetooth beacons with lower power output (class 3, 1mW) would yield even higher accuracy if desired, since their range is on average 10cm and maximum 1m [18], even though in this case we are already well within necessary accuracy to carry out our task.
the LG has and has not visited, and wether or not the LG has access to an automobile.

The class Place with its attribute location, which it gets from the Bluetooth beacon, provides us with vital information about where we are as well as when we last visited the location. The class Place is actually an abstract class and has as such no instances. One level down we find: PlaceHome and PlaceWork which are self-explanatory. The class PlaceLunch is also central in this scenario. It contains information about among other things, the menu, whether or not the place is open at the moment, and the number of times the place has been visited by the LG. We also find an abstract class called PlaceTransport. PlaceTransport contains information about the speed and capacity of a transporter. It has two subclasses called PlaceTransportCar and PlaceTransportFoot.

An important part of the ontology for the user is the LunchGuestCalendarAppointment class. This contains information about when and where an appointment is, as well as whether or not the LG has been there before. The two final classes are Food, and GMT. GMT is a provider of the time and Food represents the different types of foods that are available.

Rules up close

To facilitate understanding the system, the most important rules are presented below. There are several rules/structures that keep track of future plans as well as of what has happened in the past. If there is an appointment in the near future, the system triggers certain behavior. If not, appointments or events in the past are taken into consideration and contribute to a recommendation.

There is a rule that checks for restaurants that serve food that the users prefer. Provisions have been made to check whether or not a restaurant is open or closed at the current time, thus indicating possible problems. The system also keeps track of whether or not the users have access to an automobile or not. Not-visited restaurants are continuously updated and recommended if they match the users’ preferred foods.

6.3.4 Control architecture of MAGUBI

The diagram in Figure 6.14 shows the control flow in MAGUBI. The diagram serves as an illustration for how the different rule prerequisites are interrelated, and is not meant as an absolute algorithmic depiction. As far as priority goes, the most important thing is finding out if there is an appointment scheduled at the current time plus 1, \( t + 1 \). If so, the appointment is presented and the user is notified of two things:

1. If the restaurant is closed right now.
Figure 6.13: Diagram over the MAGUBI lunch-ontology, containing information regarding places, transportation, appointments, and agents.
6.3. Exploring The Relevant Service Discovery Layer using MAGUBI
6. EXPLORING THE SERVICE-DISCOVERY LAYERS

Figure 6.14: Diagram of the MAGUBI control flow, showing the reasoning process in the rule-based expert system, based on either results from ODEn or the common-sense knowledge ontology.
2. If it is possible to get to the lunch appointment punctually as well as the at that time recommended method of transport.

If there is no appointment in \( t + 1 \), there are two possibilities:

1. The system will first check what was done at regular intervals (e.g., days, weeks, months, and years in the past). In the case of a hit in the history, the system proceeds with a recommendation based on previous appointments/recommendations/visits, for example a birthday, as well as whether or not the restaurant is open right now.

2. When there is no hit in the history, the system performs a search for new restaurants and if one is found, the system checks for compatibility with the user’s preferred food and presents the new restaurant to the user.

In case of the restaurant being incompatible with the user’s food preferences, the system will present a previously visited restaurant with a menu that is compatible with the user’s palate.

### 6.3.5 Running MAGUBI

In Figure 6.15, MAGUBI can be seen acting proactively on data supplied through the Protégé ontology. In this particular example, MAGUBI recommends a new restaurant that has not been visited before. In doing this, it checks for appointments in the future, as well as in the past, before proceeding with recommending a newly discovered service in the form of a restaurant that serves food that the user likes. As shown, several rules do not fire in this scenario. In particular, appointment-related rules are not activated due to the fact that there is no appointment in the immediate future, nor are there any appointments at significant times in the user’s history. (Such as a day, a week, a month in the past, and so on.)

### 6.3.6 Lessons learned

In developing MAGUBI, we have tried to produce ubiquitous computing by enabling access to pre-existing infrastructure through enhanced service discovery software. The approach scales at a technical level inasmuch as it is able to handle multiple users, since there is no central server that can be overloaded with requests. In MAGUBI, there is one server per user. Scalability in this sense is also true for the case with OdEN, since we base it on JXTA, a system which has been tested for network performance [61]. When it comes to the scalability of extending the system to encompassing situations and scenarios beyond the lunch scenario, the scalability is not as straightforward. It is not that it is impossible per se, but this approach
6. EXPLORING THE SERVICE-DISCOVERY LAYERS

Figure 6.15: Running MAGUBI. Showing logging outputs as well as the recommendation from the system.
6.3. Exploring The Relevant Service Discovery Layer using MAGUBI

demands the developer to be proficient in ontology engineering and its support tools, such as Jess and Protégé. The lunch scenario rules can be generalized into rules that cover generic terms, such as a *meeting* rule that covers both lunch and other types of meetings.

In creating these rules and ontologies, we see the need for professional ontologists and rule-developers. It is not likely that the average users of the system have the skills required to create and maintain the ontologies and rule sets, nor should it be expected of them, in the same manner that it is not expected of an average automobile-owner to be able to perform advanced automobile maintenance. One way forward is to simply let users buy ready-made ontologies and rule-sets for certain domains, and have the system ask them for the appropriate data. Another way is to have better tools aimed at the consumer level, with a more intuitive, simplified, and probably graphical representation of rules and ontologies, or possibly a simplified language similar to the languages used on wikis for example. Yet another way forward is to share ontologies and rule-sets among users who use trust-based buddy systems. In this way, we get publicly editable rule-sets and ontologies as well as peer-review, which combined, should in theory prove beneficial for rule quality as well as rule creation.
6. EXPLORING THE SERVICE-DISCOVERY LAYERS
Chapter 7

EVALUATING RESPONSORIA

This chapter has five main parts: The evaluation platform RESPONSORIA itself, the reasons for developing this platform and performing the evaluation, the method used in the analysis of RESPONSORIA, the analysis of RESPONSORIA, and finally the results from the analysis.

7.1 RESPONSORIA & Integration of MAGUBI in it

Here we present RESPONSORIA, its architecture, the current realization of its architecture, communication within as well as outside of RESPONSORIA, and the integration of MAGUBI into RESPONSORIA.

Crisis management is a complex task that involves inter-organizational cooperation, and the sharing of information, as well as the allocation and coordination of available resources and services. It is especially challenging to incorporate new, perhaps temporary, actors into a crisis-management organization while continuing to use the same command-and-control (C²) system.

Local emergency-response organizations must maximize their use of existing resources. Therefore emergency-response organizations need appropriate C² systems to coordinate not only their own resources, but also to take advantages of other local actors. The local nature of response coordination imposes additional challenges for the development of C² systems. In particular, the C² systems must support coordination across organizational boundaries at the local level.

The work presented here in this chapter builds on the results of a previous study that focused on exploring the possibilities for service orientation in C² system for emergency management at the local level, Pilemalm and Hallberg [130]. Through interviews and scenario studies the previous study identified requirements and developed an initial mock-up, which was used for scenario-based evaluation.
7. Evaluating Responsoria

Based on a preceding requirement-analysis study involving interviews and workshops with crisis-management staff, we have developed a prototype system C² system that facilitates communication, collaboration, and coordination, for further exploring the use of SOA in development of C² systems for local emergency management.

A salient feature of this system is that it takes advantage of a mash-up of existing technologies, such as web-based mapping services, integrated in an open service-oriented architecture. By taking advantage of lightweight solutions capable of running as web applications within standard web browsers, it was possible to develop a scalable structure that supports decision making at multiple levels (operational to tactical) without the need to modify the system for each level. The use of C² systems implemented as web applications creates new possibilities for incorporation of multimedia components, such as popular web-based multimedia features.

Service-oriented architectures (SOA) provide new technologies for the development of C² systems. This approach is based on a set of loosely-coupled services offered by multiple actors rather than a single monolithic system.

This section reports the result of a prototype SOA implementation that builds on a previous requirements engineering study for service-oriented C² systems for local emergency response. The results illustrate how it is possible to develop lightweight C² systems using state-of-the-art Web and SOA technologies. However, a number of organizational and maintainability challenges remain.

In the remainder of this section, we explain the technical background, outline the methods used, and present the result, in terms of the system developed and its components. We discuss lessons learned from the prototype development as well as advantages and disadvantages of the service-oriented approach for C² development. In addition, we discuss the possibility of automatically integrating multimedia services into the C² system via a service-discovery mechanism which uses knowledge about the services and the situation to determine which services to display.

7.1.1 Introduction

The objective of this section is to present the service-oriented prototype system, RESPONSORIA, and to discuss the use of SOA to realize C² systems for Crisis management. RESPONSORIA illustrates the use of current Web and SOA technologies for C systems. The goal for the RESPONSORIA prototype is to (1) further explore the potential of SOA in the emergency-management context; (2) illustrate the concept of SOA to practitioners, decision makers and other actors; (3) identify potential weaknesses of the SOA approach and its supporting technologies; and (4) pinpoint obstacles in the development process. Although this prototype by no
means implements all the services required for a complete C² system it uses the relevant SOA technologies and communication protocols as the underlying information infrastructure. For example, RESPONSSIA is based on a state-of-the-art Web-based front end combined with current SOA frameworks and application servers.

Citizens have high demands for well-functioning emergency responses when a crisis occur. However, it is difficult to be fully prepared due to the fact that the time, location, and nature of crises are hard to predict. Neither is it financially possible to have staff and equipment resources exclusively standing by for all conceivable crises. In reality, it is necessary to make use of resources at hand from different actors to create improvised solutions (Mendonça et al. [108]). The local community is commonly the level on which crises must be handled (Haddow and Bullock [60]).

The complexity of major response operations involving multiple actors combined with the inherent intricacy of a multitude of different organizations and organizational levels acting according to different mandates and regulations imposes serious challenges for emergency management (Jungert and Hallberg, 2008). It is essential that these actors can allocate resources and synchronize activities in an efficient manner (Shen and Shaw [142]).

Crisis management at the local-community level presents many challenges [60]. Two of the most significant challenges are: (1) The management and coordination of external actors with regards to participation in solving the crisis situation and (2) the design and use of the command and control (C²) system for handling daily activities as well as extreme events. The first challenge is commonly handled by using human actors as intermediaries between the crisis-management system and the crisis-management staff. Typically, the second challenge is addressed by employing dedicated C² systems for crisis situations.

A disadvantage of employing dedicated C² systems, however, is that they are used in serious situations exclusively, which means relatively infrequent use. Infrequent use leads operators being uncertain of how to perform certain actions within the system, which affects overall crisis-response performance. Furthermore, infrequent use contributes to a lack of knowledge about how the systems perform in real situations.

Command-and-control (C²) systems can assist response commanders in situation awareness, planning, and resource allocation (Jungert et al. [80]). However, traditional C² systems are difficult and time-consuming to develop, especially because they must allow collaboration with multiple actors and their diverse information infrastructure. Furthermore, the development of C² systems requires a deep understanding of the activities, objectives, and actors involved, as well as the information needs of these actors. For local communities, it is important to integrate local and
regional resources from e.g. rescue services, police and healthcare, into a lightweight C² system that facilitates cooperation.

Recently, service orientation has emerged as a new approach to system development and to system architectures. Such service-oriented architectures (SOA) are now used to implement information systems for a wide range of business applications. By organizing C² systems as a set of services, commanders can have access to a multitude of resources and assistance from different organizations. In this model, each actor can make services available to others by publishing them on a network. Client systems can then call services to perform tasks, and sometimes even discover new services and accommodate for changes in service availability. One of the goals of service orientation in the emergency-response context is to increase the flexibility of resource allocation, for instance by making use of resources normally used for other tasks.

In crisis situations, time is a critical factor. Frequently, it is the case that different C² systems as well as other information systems must interact on an ad-hoc basis. It often happens that these systems cannot interchange data or interpret data that other systems provide. In practice, these limitations are currently handled by human intermediaries and by a liaison staff between the crisis-management organization and the systems employed by the external actors. For example, if the crisis-management organization needs transportation, the liaison staff is forced to contact the transportation companies directly by telephone, since the crisis-management organization has neither direct access to nor knowledge about the systems employed by the transportation companies or the transportation resources currently available (Mendonça et al. [108]). This type of ad hoc communication sometimes leads to a bottleneck because it uses personnel resources.

Although C² systems can assist response commanders in dealing with situation awareness, planning, and resource allocation [80], the traditional approach to C² systems may lead to extensive system-development times as well as difficulties in integrating the different actors and their heterogeneous systems. Unless system designers have a substantial level of understanding of the different actors involved as well as of their objectives, activities and information needs, the result will be systems ill-suited to the task. Furthermore, it is essential that the different actors in the local community can synchronize, coordinate, and distribute resources (Shen and Shaw [142]). Moreover, it is important to integrate local and regional resources, for example from fire and rescue services, the police force and medical-care services in the overall crisis response. Today, it is possible to develop lightweight C² systems that facilitate cooperation based on state-of-the-art web technologies. Such web applications can integrate new services, including multimedia, in novel ways. For instance, C² systems implemented as web applications can relatively easily support extensions consisting of a mash-up of web components from different sources.
7.1.2 Proposed solution

As described in Section 7.1.1, the challenges facing crisis management are system-related as well as organizational. The proposed solution is based on a combination of several existing technologies and consists of two parts: RESPONSORIA and MAGUBI. RESPONSORIA is a Service Oriented Architecture (SOA) based Command and Control (C²) application designed to help staff when handling a crisis situation. It is responsible for the interaction and connectivity between different services, devices, and users, once these have been selected for inclusion in the situation [78]. RESPONSORIA is the product of a previous study [130] which determined the need for a SOA-oriented C²-system through interviews with crisis management personnel. Specifically, the system needed to support (1) deployment with a minimum of effort. (2) High standards compliance. (3) High modularity. (4) Update and maintain parts as well as the whole system while the system is operating. Further goals that needed to be incorporated included that it should incorporate a desktop application look and feel. MAGUBI is responsible for service/device/actor discovery and for the recommendation of different services/devices/actors. Sections 7.1.7 and 7.1.8 describe RESPONSORIA and MAGUBI, respectively.

7.1.3 Multimedia and crisis management

Although the aforementioned challenges (such as the cooperation between different actors) are significant, the incorporation of multimedia into C² systems may help in addressing them. However, the incorporation of multimedia in traditional C² systems has been challenging and difficult. This has been particularly problematic because situational awareness is essential to crisis management.

![Diagram of the OODA loop](image)

Figure 7.1: The OODA loop (in Brehmer [23]). The OODA loop stands for Observe, Orient, Decide, and Act. Normally, it refers to a single person doing this cycle. However, the OODA loop can also be used when referring to organizations. Ultimately, the OODA loop describes how an individual or organization reacts to an event.
To create C²-systems that work in real situations, it is necessary to incorporate grounded theory. A common theory used in planning for this type of situation is the OODA loop\(^1\) [63], see Figure 7.1. The OODA loop states that there are different phases in the decision-making process. These phases are: Observe, Orient, Decide, and Act. To achieve a successful outcome from the decision-making process, it is important to support the different phases properly. To provide this support, the C² system used by the commanding staff must be OODA-loop aware, in that it supports different phases in an integrated way. In Section 7.1.4, we discuss this need in detail and present how our model and current implementation tackle the issue. There have been many enhancements to the original OODA loop. Brehmer [23] proposed the Dynamic OODA (DOODA) loop model, which introduces what he refers to as “additional sources of delay” in the process. Examples of such types of delay are information delay, which is the time between actual outcome and the decision-maker being aware of it; dead time, which is the time between the initiation of an act and its actual start; and time constant, the time required to produce results.

7.1.4 Multimedia technologies and their integration into RESPONSORIA

The OODA loop and RESPONSORIA

RESPONSORIA supports the OODA loop in multiple ways. First, it supports the first run through of the OODA loop by providing a rich environment in which to conduct observations. It is worth noting that RESPONSORIA also allows for observations to be performed from the field directly in the tool, thus supporting the orient part of the OODA loop. Second, it integrates tools for making sense of the data observed. One example of this is the possibility to visualize numbers quickly as charts (see Figure 7.2), further aiding in the orient and decide parts of the OODA loop. Third, it provides means to issue commands, supporting the act part of the OODA loop.

Integration of technologies into RESPONSORIA through service discovery

As mentioned above, the integration of different technologies through different services is a key factor in creating a viable crisis management system. This integration may be performed in different ways:

1. **Manual integration.** In its simplest form, we are able to integrate technologies and services by just adding URLs. This may even be

\(^1\)Although the OODA loop was originally designed for military situations, it is used in many other areas as well.
7.1. RESPONSORIA & Integration of MAGUBI in it

performed by users, either individually or together. In essence, a wiki-type interface is created in which the users construct the application in concert.

2. Automatic integration. While manual integration is certainly possible, automatic is preferable. One of the most important reasons for automatic integration is the labor savings it allows. To obtain automatic integration, we propose the use of service-discovery systems such as MAGUBI.

The RESPONSORIA model, especially the web-based user interface, can benefit from both manual and the automatic integration. Today’s web browsers are capable of displaying and utilizing a wide range of media and technologies out of the box. In our solution we have focused on technologies built into the browser, such as JavaScript, JPG, and PNG. Specialized data formats may be converted using a web service.

Potential multimedia technologies

Since situation awareness (or orientation, as specified in OODA) is one of the highest priorities when addressing a crisis, we will briefly mention some resources and technologies that may enhance this and still be easily integrated into the main system. We will also relate these techniques to the OODA loop. A web-based crisis management system makes it possible to rapidly tie in new services as they become available.

Personal video streaming

One of the possible technologies that is easy to integrate into RESPONSORIA is live video streaming. Services such as Bambuser [11], Qik [145], Flixwagon [48], and Kyte allow the user to broadcast live video over the internet using their own mobile phone as a transmitter. This means that every cellular phone is now a potential live-coverage camera in a crisis situation. This technology is instrumental in the observe and orient parts of the OODA loop.

Online charting

Another possible technology consists of charting applications, for instance the chart API from Google as shown in Figure 7.2, or Complan [90] as can be seen in Figure 7.3. It becomes easy to integrate this type of multimedia by merely including a URL. Apart from their rapid integration using URLs it is also possible to convert textual data into diagrams on the fly. These charts may be rapidly created using web services or webpages that feature simple user-interface components, such as drop down menus.
A possible drawback when using the simpler URL method is that the amount of data passed to the graphing application may cause the web server to report an error as the URL length expands beyond the web server’s limit. Nevertheless, it should be noted that the simple URL method does provide a rapid and uncomplicated way of producing charts from data.

Furthermore, the storage requirements of these diagrams are small, due to the fact that they exist as URLs. Using URLs also has the added benefit of saving bandwidth and computing time for the crisis management center, since pictures will not be served from the crisis management’s own data center but from a third party.

With regards to the OODA loop, online charting fits in the decision part, since it provides supporting information regarding which direction to go.

**Online animations**

Using technologies such as OpenLazlo, enables data from formats such as XML to be converted for instance into Flash or DHTML to be easily
7.1. RESPONSORIA & Integration of MAGUBI in it

Figure 7.3: Complan [90] showing different tasks to be performed in a crisis scenario and when to do them.

accessible online [87]. Ming [94] is a similar framework that generates Flash on the fly.

7.1.5 Method

This work has been conducted in seven major stages: (1) a literature study, (2) interviews with emergency response representatives, (3) design of a crisis scenario, (4) design of prototype that supports service-oriented C² system for emergency response, (5) a scenario based evaluation of the concept, (6) creation of a demonstrator system, and (7) evaluation. Stages 1–5 have been reported previously in Pilemalm and Hallberg [130] while this section focuses on Stage 6. However, as a background, we will start by briefly outlining stages 1–5. Stage 7 is reported later in Section 7.4 and 7.5.

Stage 1 was a literature study comprising the structure of the emergency response in Sweden and the crisis organizations responsibilities. Stage 2 involved interviews with representatives from the six major crisis organizations at local level. In Stage 3, the results from the literature study, visits, and interviews were used to develop a scenario of an incident in a local community. Stage 4 identified the necessary services that corresponded to the scenario and to design of a mock-up prototype of service-oriented C² systems. In Stage 5, the prototype of the user representatives was evaluated in a scenario-based session. The result of the evaluation was used to improve the concept.
Stage 6, which is reported in this section, started by assessing the requirements from the previous five stages, including the result of the evaluation. From this assessment, it was concluded that the architecture of the system needs to permit: (1) deployment with a minimum of effort. (2) High standards compliancy. (3) High modularly. (4) Update and maintenance of parts and the whole system while the system is operating.

One of the more prevalent and widely used SOA technologies is Java EE. Java EE brings with it a fault-tolerant, distributed, multi-tier, structure that employs an Application Server that allows the building of a highly modular system. It also provides persistence, transaction processing, concurrency control, etcetera through Java Enterprise Beans. Thus, a large amount of work concerning the surrounding system is handled automatically. Furthermore, Java EE has good support provided by a multitude of tools, for administering, and controlling large systems.

After selecting Java EE as our platform for the backend, we concentrated on meeting the requirements for the frontend. From the requirements, it was clear that the desired look and feel was that of a desktop application. However, it was also desirable to combine this look and feel with the other requirements of easy administration, deployment and so on. These requirements are not easy to satisfy simultaneously. Traditionally, a web-based solution will have ease of deployment, but may not provide a desktop application feel. We believe we have found a useful compromise in using Google Web Toolkit which enables both.

We verified the support for Business Process Execution Language (BPEL) in different Integrated Development Environments (IDE). Netbeans [22] became our IDE of choice for this project due to the built in support for both Java Enterprise Beans and BPEL.

7.1.6 Scenario

Devoid of their context systems are not understandable. Therefore the context of the scenario (adapted from Pilemalm and Hallberg [130]) is briefly presented. In a Swedish community of about 150,000 inhabitants, on the highway cutting through the community a truck carrying propane overturns. Shortly after, a bus carrying 25 people runs into the truck and as a result, another 10 cars are involved in a multiple collision. It is close to Christmas. The incident occurs near an overcrowded shopping mall and a large event arena where a Christmas concert is taking place. From this point the scenario presents how by using a service-oriented C² system, the local crisis response organizations initiate the response and collaborate by sharing information and resources. The scenario assumes that there exists a service-oriented infrastructure which allows the actors to provide and make use of services for crisis management.
7.1. RESPONSORIA & Integration of Magubi in it

7.1.7 The RESPONSORIA model and implementation

To better understand the potential of our model, we present the basic concepts and ideas behind it. In its most simplified form, the RESPONSORIA model is a Service Oriented Architecture (SOA) which uses web services as basis for the entire system.

A web-based user interface retains a desktop look-and-feel through the use of JavaScript, while keeping the solution accessible through standard web browsers, see Figure 7.4. A proxy in the web server handles communication with other components, such as application servers in RESPONSORIA. The pluggable structure extends to the application servers as well. Furthermore, RESPONSORIA utilizes Enterprise Java Beans (EJBs) in the form of web services for mapping, note-taking, logging, etc. It is straightforward to expand the system by incorporating other web services.

As mentioned, there is a desktop feel to the application itself. This is achieved by using the Google Web Toolkit (GWT), which enables developers to use Java syntax to program an entire web application. Through post-processing the application is transformed into a JavaScript application suitable for web browsers. In essence, the developer can program this in the same way as a normal desktop application, but still deploy it as a web-based application since the GWT compiler produces JavaScript code, which is deployable on Web servers. One major advantage of GWT is that it enables a look-and-feel similar to desktop applications while providing straightforward development and deployment. The RESPONSORIA client has been successfully tested on Apple OS X desktop, iPhone, MS Windows XP, Firefox, Internet Explorer, and Safari. We foresee that the system will work on most of the high-grade, hand-held machines currently available.

Since the system utilizes Java EE web services, it facilitates their cross-platform distribution in much the same manner as the main program does. The Java EE platform also comes with a host of features for portability, quality of service, and security.

System architecture

As it is SOA-system it is service oriented. The design of the system has modularity and flexibility as main goals. The system is comprised of two major parts (Figure 7.4). A is the front end and handles the user interface as well as communication with the services out in the field. It may considered to have the application logic. A may be web based just as it is currently. It may also follow a traditional desktop based paradigm. The only requirement is that A is standards-compliant when accessing the services out in the field by using SOAP and WSDL. B is the aggregated system comprised of a multitude of different web services. These may be
Figure 7.4: Architecture of the RESPONSORIA system. (A) Web-based user-interface client. (B) Server cloud consisting of a collection of implemented web services running on application servers.
based on any type of platform/underlying architecture' as long as they have WSDL operations and are able to communicate using SOAP.

Current realisation of system architecture

At the moment, RESPONSORIA is a SOA application with a web-based user-interface front end, which is comprised of a number of subsystems (Figure 7.4). The core of the system consists of a web server which delivers web pages to the browser client using Google Web Toolkit (GWT) [54]. This approach makes it possible to have a rich desktop application feel while retaining the advantages of a web-based solution for the users. The web server has a proxy that enables communication with other parts of RESPONSORIA. On the application-server side, there is a pluggable structure as well. In the basic configuration, RESPONSORIA uses web services in the form of Enterprise Java Beans (EJBs) for Logging, ID, Mapping, Note-taking, and so on. Due to the pluggable structure, it is easy to extend the system by adding references to other web services.

Currently, the system is running on a Glassfish 2 backend [126], which is an open-source application server implemented in Java. This approach allows the application server to run on virtually any available system. During development, we have successfully performed development and testing using a cross-platform approach in order to verify maximum functionality and be able to distribute the system over a multitude of platforms.

The services themselves are Java Enterprise Edition (JavaEE) [125] web services. This approach allows for the same type of cross-platform distribution ability as the main application. It also provides some significant features for security, quality of service, and portability.

Implemented services in the demonstrator

We have implemented a number of the services that were identified as necessary in the previous study. They are implemented as proof-of-concept services. We have also created a number of metaservices for internal use within the system. We give the following examples: IDMetaservice: Creates unique ID numbers for the requests in the system. Mobile phone positioning: Using Google Maps and geographic coordinates we plot the current position of a simulated mobile phone and/or its travel pattern for a selectable period of time. Distribute information: sends information to designated receivers. Request rescue services: enables ordering of rescue services for different tasks. Logging: automatic logging of activities that take place in the system. Note taking: possibility to augment the log with personal notes.
7. Evaluating Responsoria

Some of the services enable the creation of a distributed staff team comprised of members from the different organizations. The Logging service and the Note taking service In particular aid in this, as they allow the users of the system to implicitly see what others are doing as well as leave time-stamped notes to each other. C²-systems exist to support the role of the staff by allowing information exchange. In this case, an automated operational picture is provided by the C² system and is based on available information partly generated by the system and partly by its users.

7.1.8 MAGUBI service discovery and RESPONSORIA

While the RESPONSORIA model handles the usage of the services and the GUI, the MAGUBI model handles service discovery [79]. Since MAGUBI is targeted towards ubiquitous computing, it works well in crisis management situations that have many actors, services, and different types of media. Service discovery in MAGUBI can be performed in two ways:

1. **User activated.** By specifying the service or device that the user is looking for, as well as their potential properties and priorities, the user can instruct MAGUBI to search for matches.

2. **Automatic.** MAGUBI performs the service discovery itself. By using models that describe the user and world, it is able to decide which services are of interest to the users, and subsequently execute searches proactively for them.

Figure 7.5 shows the MAGUBI model, which is comprises of two parts: MAGUBI and ODEN. The whole model is named MAGUBI since it is the controlling module. The two parts are surrounded by auxiliary modules. Starting from the bottom in Figure 7.5, we can see the services and devices themselves. A peer-to-peer (P2P) subsystem keeps track of these services and devices.

ODEN is the subsystem responsible for the user-activated or more traditional service discovery. By using ontologies, ODEN is able to expand on the traditional concepts for semantic models of devices and services. After using a P2P subsystem to download semantic descriptions provided by the services and devices themselves, it evaluates them locally on the client. After evaluation, the results may be presented to the user, or post-processed in the MAGUBI module.

The MAGUBI module may either post-process results from ODEN or initiate searches on the users’ behalf. In the case of post-processing, MAGUBI inspects the results from ODEN and compares them to semantic information stored in its ontologies pertaining to: the world, devices, services, and the users. As an example, the user may try to locate transportation in the form of a taxi. In an ordinary service-discovery
system, the user will get a long list of available taxis. Using the ODEN subsystem, the user gets a shorter list, tailored to the exact specifications of the required properties that the user provided. MAGUBI goes one step further and, for example, filter out such taxis that might very well fulfill the required transportation properties, but may soon require refueling, and as such are realistically unusable, since there is more to the transportation service than merely being able to start it.

For the proactive part, MAGUBI may commence searches for services and devices that it judges appropriate for the user. These searches are based on the user and world models, in cooperation with the rule engine and its rules. Naturally, these searches are carried out through the ODEN subsystem and are subjected to the same post-processing as the user-initiated ones are.

At the top of the MAGUBI system is the GUI/DUI module (Larsson and Ingmarsson [86]). In this case, it is integrated into the web interface of RESPONSORIA and accepts requests from the user and also presents results from proactive searches that the MAGUBI module may do independently from the user.

### 7.1.9 A brief introduction to the prototype user interface

The basis for the development of the prototype user interface is the set of requirements identified by Pilemalm and Hallberg [130]. Figure 7.6 shows the main view in the user interface. To the left is the service selection
panel (A). This panel lists the resources, devices and services available. We have incorporated different layouts for the inclusion of the different resources. The first type of listing is an alphabetical one. Another type of listing is based on the order in which a specific task is carried out. A third type of listing may be based on recommendations from the service-discovery system. Users are also able to specify customized orderings themselves. A menu bar is placed immediately above the A panel. This enhances the perception of the application as a desktop application. This perception is particularly strong if used in conjunction with a full-screen capable browser.

In the service panel itself, the currently selected activity is shown, see Figure 7.6 area (B). As can be seen, there are tabs that enable the user to work with many different activities at the same time. Furthermore, as shown, the service panel itself also provides opportunity for incorporating different media and services. Figure 7.6 area (B) illustrates how the system uses Google Maps together with a web-service that tracks mobile phones.

Status information is displayed in the panel to the right, Figure 7.6 area (C). Currently, three tabs display various information, such as Request status, Activity status, and Task status. One particularly important feature is the log, Figure 7.6 area (D), which also contains a note-taking function. In order to enhance situation awareness, this log is designed to be shared by everybody using the system. It enables anyone to review what has happened, when it happened, and who did what.

Motivation for choice of GWT as front-end technology for this type of system

The previous study by Pilemalm and Hallberg [130] found that there is a need for systems that allow cooperation between multitudes of different organizations and at the same time enable an open structure for plugging in available, and in some cases, transient services. These systems should also simultaneously cater to personnel in the field as well as in the central command.

Traditionally getting an application-like feel has been a challenge when using the browser as an interface. In the early days of the World Wide Web, JavaScript was the way to get closer to this than what was normally possible on a browser. However, JavaScript alone does not lend itself very well to activities that require continuous data transfer between the network and the client.

GWT provide the properties sought after. We choose GWT because it provides advantages both for developers and end users. In terms of development, GWT allows writing the application in Java, which enables the developer to take advantage of a richer feature set than if the application was written in JavaScript. For the user, it allows a desktop
7.1. RESPONSORIA & Integration of MAGUBI in it

Figure 7.6: RESPONSORIA’s main interface. (A) Service selection panel, with pre-configured service groups for different scenarios. (B) Service panel showing the Mobile phone positioning service. In this case showing the a trace of a mobile phone for the last five hours. (C) Status panel, showing progress for different requests as well as tasks and activities. (D) Log. Showing all activity in the system as well as provider of a note taking function.
look and feel. (Especially when a browser that allows the removal of
the browser-frame is being used.) GWT is also GNU GPL-compatible,
which provides for easy access to the internal source and structure of the
framework. Furthermore, there are numerous third party libraries for
GWT. Among the ones we have used are libraries that handle graphics
manipulation, and sound feedback, as well as look and feel. These libraries
have saved development time and added features that would otherwise
have fallen outside of the time allotted to the project.

Another way of solving the interaction challenge in a browser is to use
proprietary software such as Adobe Flash. Unfortunately, this solution
requires the user to install a plug-in. Moreover, it places the developer
in the position of dependence on a single company. A combination of
technologies known as AJAX (Asynchronous JavaScript and XML) has
alleviated some of the difficulties, but there have been a lack of higher level
solutions.

Communication in the system

The desktop feel stems from a continuous link to the web-server (see
Figure 7.4). This open link enables the web browser to stream data back
and forth between itself and the web-server. By streaming data, the
application is able to react to user interaction in the same manner that a
desktop application does, without the need for the user to press Submit
or similar type buttons/controls. The communication in RESPONSORIA
follows web standards. Initially, the application is loaded into the browser.
A GWT-application may be viewed as having three parts (see Figure 7.4).
The first part is the part that gets loaded into the web browser. It is
comprised of JavaScript. The web browser part then has the previously
mentioned continuous contact with the second part of the application
which resides on the web server. For security reasons, the second part of
the GWT-application is restricted in what it can do on the server. Complete
functionality requires the third part, the GWT-proxy. This proxy is needed
for the GWT-program to be able to communicate with the outside world
and to utilize the services there.

Let us consider a service user by discussing the Mobile phone positioning-
service, which can be seen in Figure 7.6 area B. When the user has
started the application (by opening a webpage), the system first loads the
application in the form of JavaScript onto the web-browser, draws the
interface on the screen and then opens the connection to the webserver.
At this point, the user may enter a phone number to be positioned, as well
as a time interval. When the user clicks on Position, the system uses the
GWT-proxy to send a request to the correct web-service which returns a
result to be presented by the GUI.
7.1.10 Integration

In Figure 7.6, area A, there is a list of available services. As has been mentioned in [74], it becomes increasingly difficult to select effectively the right services for the situation and location as their number increases. This is true also in a crisis situation such as the one RESPONSORIA is designed for. In many service-based applications, the list of services appears either as one long list, or as a list which is statically subdivided on the basis of one criterion such as alphabetical order. RESPONSORIA alone provides several static views of the currently available services. These views range from being strictly alphabetical, to based on situation.

By integrating MAGUBI in RESPONSORIA it is possible to obtain a list of services that varies, not only according to services that are available, but also according to the current situation. By knowing who the operator is and what the operator is trying to accomplish, MAGUBI can create custom lists and present them in custom views. For example, by knowing that an ambulance crew cannot provide services for blocking off a street, MAGUBI then chooses not to include ambulance services.

Another example may be in the case of emergency transportation. By using knowledge about the type of emergency and transportation time for different modes of transportation, MAGUBI may choose to exclude ground-based services such as traditional car-based ambulances. This is especially important in cases such as heart-attacks when rapid response is important. Where less critical injuries exist MAGUBI may choose to optimize the services presented for different factors such as transportation cost, and thus opt for a ground-based transport instead of a helicopter-based one.

A further way in which MAGUBI may optimize services presented is by re-arranging the order of them. This is important since many services depend on other services in different ways. These ways include services being composed of other services, and the order in which services are executed.

The possibility for parsing Business Process Execution Language (BPEL) that will potentially be used in RESPONSORIA gives MAGUBI the ability to extract useful information about how different services depend on each other and how they are connected. By combining this with ontological knowledge about tasks, the world, and the users, reasoning about the services can be obtained in even greater detail and a more significant sorting of services created. This is especially valuable since ontology programming/creation often requires both knowledge about ontology engineering as well as a comprehensive grasp of the domain. BPEL on the other hand is designed for individuals with little or no programming experience, even though it is expected that they are subject-matter experts.
However, it should be noted that to use the BPEL tools may require training.

### 7.1.11 Observations

In this section, we discuss the redundancy feature of RESPONSORIA and the service-discovery mechanism that facilitates the integration of different technologies, media, and services. We also discuss the development process as well as matters of security, quality of service and maintainability.

#### Redundancy

There are several layers of redundancy in our model. As seen in area A of Figure 7.4, even though the web server is the weakest link in the concept, the server side can be hardened through off-the-shelf web-server technology solutions, such as backup servers that automatically engage if the main server fails, and other JavaEE features [125]. Area B of Figure 7.4 shows an example of the application servers. It is very likely that there will be a surplus of servers offering similar if not identical services. Through the use of service-discovery such as MAGUBI, rapid recovery is ensured if services fail.

#### Service Discovery

The technical aspects of service discovery through the use of the custom-built application MAGUBI has been mentioned in Section 7.1.8. Here, we will address the non-technical part of MAGUBI, namely its philosophical underpinnings. MAGUBI is a service discovery model and implementation that addresses service discovery from the perspective of the user rather than of the system. This perspective connotes an attempt to address the issue with discovery and selection of services.

Many traditional service-discovery systems only address the discovery part of service discovery. On the one hand, this focus helps the user, since services are discovered. On the other, the user is left wanting when it comes to how to perform service selection, since the user has to make the evaluation and selection manually. With MAGUBI, this evaluation and selection is offloaded from the user onto the service-discovery system. Furthermore, since MAGUBI has information about the world, situation, and the users, it is able to make proactive suggestions in terms of services based on what it deems necessary at the time.

#### Multimedia

As mentioned above, different multimedia services support different parts of the OODA loop. A service provider may help in the configuration of an
7.1. RESPONSORIA & Integration of MAGUBI in it

interface by providing information in the service descriptions about where in the OODA loop his/her particular service fits in. MAGUBI supports automatic classification into the different OODA categories depending on which rules are entered. We believe that this results in greater efficiency regarding both where in the GUI available services should be positioned, and also as a support to the selection of services to be included in the GUI.

Development

Although prototype development is not quite comparable to the development of a full-scale C² system, advantages and disadvantages of the SOA approach are still apparent already at the prototyping stage. Let us discuss some of the lessons learned from the development of the RESPONSORIA prototype and from the prototype itself.

There are different ways to structure the services. The selection of services we implemented was based on the requirements analysis and scenario description by Pilemalm and Hallberg [130] that was described earlier on page 116. This model means that the services are structured according to the scenario and to what the actors can deliver. However, it is possible to structure the services in alternative ways. It is particularly interesting to consider alternative ways to structure the RESPONSORIA client. One way is to break up the client user interface and structure it as a set of services. Such services would not necessarily correspond directly to organizational activities, instead to specific functionality in the user interface and end-user tasks. For example, we have experimented with making the list of services in the RESPONSORIA user-interface client itself a service (i.e., the client then asks this service to provide a list of relevant services). In a similar way, it is possible to develop the user-interface client as a set of services.

It is possible to create new services from an aggregation of services. For example, the Business Process Execution Language (BPEL) is an XML-based language for describing business processes which are supported by an aggregation of primitive services that together support process flows Koppmann et al. [82]. One of the advantages of languages such as BPEL is that they are designed to be used by subject-matter experts (i.e., non-programmers) to implement services (e.g., in a graphical flowchart language). Initially, we attempted to use BPEL to describe services in the RESPONSORIA prototype. For the service domain and the scenario in our C² application, however, we found that it was not meaningful to describe services as BPEL processes. In addition, it is unclear whether it is meaningful for non-programmers to use process-definition languages directly in the emergency-management C² domain.
7. Evaluating Responsedia

The software development tools are important for the implementation of services and service-oriented systems. Integrated Development Environments (IDEs) can support developers to create and maintain source code and definition files for SOA (e.g., classes, interfaces, WSDL, and BPEL). In our practical development work, we found the IDE to be both an indispensable tool and a limiting factor for SOA development. The IDE proved quite useful for creating source code templates and stubs and for generating WSDL and BPEL documents. However, we found the refactoring support for SOA-related changes limited and sometimes incomplete or flawed. Furthermore, manual troubleshooting and repair of overly complex XML documents (e.g., WSDL and BPEL) is difficult and time consuming.

Security, Quality of Service and Maintainability

Security is an important issue for C² applications. In general, it is difficult to secure SOA-based applications. For example, the change from traditional monolithic systems to networked services components creates new potential attack vectors. Although SOA frameworks can provide support for the encryption of network communication and authentication of services, there might still be issues in terms of potential denial-of-service attacks and analysis of communication patterns which can expose the system operation, although the actual communication content is encrypted.

Quality of Service (QoS), is a challenge in this type of SOA framework. Traditionally, QoS is defined as having sufficient bandwidth to satisfy the requirements of a particular service, which also holds for in this situation. As expected, the heterogeneous and multi-faceted amalgamation of services presents a challenge when it comes to deciding which service to use in a given setting. The quandary of selecting the right service is especially true when one wants to utilize services that are perhaps provided by service providers other than the traditional ones. This challenge has been pointed out before, and we have developed tools (MAGU®) that are designed to handle this type of situation (Ingmarsson [74]).

The short-term and long-term maintainability of service-based C² systems is a concern. During the systems development, we already found that it was sometimes difficult to maintain the prototype, because seemingly small changes to service functionality could result in multiple changes throughout the system. For example, the addition of a new parameter to a service call resulted in changes to the service, the service description, the service call, and the Web client and its proxy. Sometimes, it was more efficient to create a new service with the added parameter from scratch than to modify the existing one. This problem would be even more difficult with an operational system in a multi-organizational environment. We believe that it is necessary to accommodate for service maintenance from the very
beginning during system development and to reach consensus among the organizations involved in service evolvement.

**Summary of the prototype system development process**

The **RESPONsoria** prototype illustrates the use of the SOA approach to the development of C² systems for inter-organizational cooperation in emergency management at the local-community and municipal level. The prototype addresses two of the most important design goals set forth by Pilemalm and Hallberg [130], namely to (1) make it possible for the emergency-management teams to keep working in a similar fashion as to what they are used to and (2) allow the use of existing resources from several actors in a crisis situation. The prototyping effort highlighted both the advantages and disadvantages of SOA. The prototype system is a web application, which means that it does not require client installation. The use of web browsers running on standard hardware makes it highly available to users in crisis situations. Furthermore, this approach enables the relatively straightforward incorporation of rich multimedia into the C² system as well as mash-ups of multimedia components. The incorporation of multimedia components can be done in different ways—both manually and automatically.

A service-discovery system can potentially facilitate the automatic discovery and inclusion of services by using knowledge about the situation and the services available, as well as general world information. The prospect of the proactive inclusion of services and multimedia through the service-discovery system is appealing. We believe that the service-discovery view of multimedia mash-ups, combined with rapid inclusion and dismissal of actors and services, can be used to develop new types of dynamic C² systems. Moreover, we believe that it is important for the C² system to be aware of the general C² method used (for instance the OODA and DOODA loops) and to provide focused support for the different stages of the decision-making process.

The combination of interactive Web technologies such as GWT and SOA technologies such as the Java EE framework, makes it possible to develop lightweight C² systems with web-based user-interface front ends. However, there are technical challenges involved in this. For example, the development environments are still immature, which makes iterative development difficult. Furthermore, there are still organizational issues that need to be resolved, for instance in terms of agreeing on service interfaces, service functionality, service availability, and service maintenance. Nevertheless, we believe that the SOA approach is viable and that it can contribute to the development of C² systems.
7. Evaluate Responsoria

7.1.12 Knowledge acquisition, Ontology construction, and Rule creation

When integrating systems such as RESPONSORIA, ODEN and MAGUBI it is important to keep in mind potential pitfalls as well as their obvious strengths. The sources of strengths in these types of systems, may also create challenges. These issues will be touched upon below.

Knowledge acquisition

In obtaining the knowledge needed for the rule, as well as ontology construction, it is vital to perceive the difference between domain knowledge and operational knowledge (Noy and McGuinness [118]). When obtaining knowledge from parties affected, it is vital to limit the amount of middle men and hence interpretation. As Compton and Jansen [26] state with regards to extracting knowledge from experts:

The knowledge engineer finds that the expert’s knowledge is not so much recalled, but to a greater or lesser degree ‘made up’ by the expert as the occasion demands.

In addition, the experts make up differing knowledge structures, (Shaw and Woodward [141]), and when confronted with the fact that there are differences, they acknowledge this and do not have a problem with it.

Ontology construction

Since ontologies are viewed as an explicit specification of a conceptualization of a domain, it is important that they represent the world as the users (such as RESPONSORIA) view it. Often, one reason to construct an ontology is to make domain assumptions explicit. However, this involves a risk since the assumptions made are not normally visible to the user. It is therefor important to try to convey the assumptions made, or at least convey that assumptions are being made. Naturally, it is not possible to create an ontology that is mapped 100% one-to-one with the real world, and in particular with every individual user. This is also the case in a system where there is no ontology, or in situations where there are no systems and just people. The fact that there is never a total one-to-one mapping is often overlooked or excused in the more traditional settings.

Rule creation

It is clear that the rules governing systems such as RESPONSORIA need to be appropriate not only to the task in hand but also to the terminology. These conclusions are nothing new, but confirmations of old truths within
knowledge acquisition, ontology construction, and rule creation. There are commercial systems that have graphical interfaces allowing for rule creation. However, since terminology may differ even within departments and especially across them, there is a risk that the rules do not reflect the actual state of the world, or that they are the actual true rules.

7.2 Reasons for a Wizard Of Oz Study

As seen previously, there is a need for a system that assists the user to select the appropriate service. This is true in many situations, for instance:

- If the user does not know what services are available. There may be interface issues where services are competing for the same channel/modality to get to the users. There are also perhaps interfaces to services that the user is unable to get to directly or is unaware of. It is possible that MAGUBI or similar systems are able to help the user in these endeavors.

- There are too many services to choose from. This situation is already a challenge in many cases and is likely to become more so in the future. When services look alike, which service should the user choose? What exactly are the differences between different services? MAGUBI can assist the user by filtering out similar services, and only offer the ones that have the highest performance as well as best match the user’s needs.

- There may be procedural issues. It is common that there exists a situation in which certain procedures are to be followed and certain actions are to be taken. A system such as MAGUBI may aid and support the user to achieve greater compliance.

After showing that it is possible to build such a system (MAGUBI), it is necessary to determine the effects, if any, such a system would have. One way to obtain information about these effects is to build a fully operational system. However, a fully-fledged system takes an disproportionally large amount of time to build. Furthermore, this time may be spent going in the wrong direction before possible errors are detected. Hence, building a complete system a number of potential pitfalls.

A conclusion from the above is that a more effective way is to test a system that behaves like a complete system, but has less of the logic in the back-end. A Wizard-Of-Oz study\(^2\) (WoZ) is one possible way of obtaining the information regarding these effects. Another positive aspect of a WoZ

\(^2\)A Wizard-Of-Oz study, is a study where the role of the system is played by a human unbeknownst to the test subject.
is the ability to have a ‘perfect system’, unencumbered by the realities of Artificial Intelligence limitations. With perfect suggestions, there is a possibility to observe emergent phenomena. If knowledge is desired about a less-than-perfect system, that can easily be obtained, by degrading the performance with suitable responses. By emphasizing the qualitative aspects of such a study, information will be obtained about the nature of the use of these types of systems.

It is important that we do not prejudge what these types of systems may bring to the table. Naturally, it would be possible to measure the execution times of scenarios and other types of quantitative aspects such as number of clicks and so on. However, with crisis management systems as well as with other complex systems, there is a question with regards to the complex and dynamic nature of the task at hand. It is clear that while such aspects as task/scenario execution time is important, they may steer the research in a pre-biased direction, where underlying factors run the risk of being filtered out. Nonetheless, in order to be able to design a study, there has to be a frame around which to build it. We use a realistic scenario developed by Pilemalm and Hallberg [130] together with different branches of the emergency services.

### 7.3 Employed method for evaluating MAGUBI

As previously mentioned, we have employed a Wizard of Oz (WoZ) method when performing our study. A WoZ study may be quantitative or qualitative, or a mixture of the two. In our case it is qualitative, since we are interested in how and/or what, if at all, “perfect” suggestions for services have for effect on the emergency operators as well as other potential results.

For a more detailed explanation of what the WoZ method is, we refer the reader to Section: 4.1.2. In short, the WoZ method allows an experimenter to give a subject an impression of an intelligent system, even though the intelligence is not machine-made, but is instead a human posing as a computer. This approach allows for the evaluation of a system before it is constructed.

**Interaction, Think aloud, and Interviews**

As mentioned, the overarching method was the Wizard of Oz. However, as sub-methods were employed, there was a continuous interaction between the subject and experimenter as well as encouragement to think aloud / talk aloud in accordance with Lewis [93] and Ericsson and Simon [42]. Semi-structured interviews Lindlof and Taylor [96, p.195] were conducted after the scenario concluded. The choice of semi-structured interviews was made to allow for follow up questions to be asked during the interview.
7.4 Experimental and analysis setup

Below, we describe the participants, physical setup of the experiment, interview setup, as well as the analysis of the interview.

7.4.1 Participants

The participants chosen were two rescue professionals. Choosing rescue professionals, provides a higher degree of connection between real situations and the participants in the experiment, which in turn should provide a higher degree of validity and relevance. The participants were not the same as the rescue professionals that participated in developing the scenario. [130] In terms of type of employment, all of the participants are professionals such as fire engineers.

The participants work at a municipal fire station in the Linköping area, and to increase the ecological validity, [64] the experiment was conducted at the fire station, in the situation room. The participants are in the unique position of being able to compare between the system that they currently employ and RESPONSERIA. In terms of attitude towards technology, all participants were generally positive. The total time for each session was between 2 and 3 hours. This included the pre-experiment briefing, the experiment itself, and the post-experiment interview.

7.4.2 Physical setup

The technical platform for both the main experiment and the control, is a modified RESPONSERIA platform, in which the backend has been greatly altered to accommodate the Wizard-Of-Oz input. The front end remains a normal Google Web Toolkit [54] application. See: Figure 7.7

The user sits in front of the screen and does not know about the wizard/experimenter controlling the application. See Figure 7.8. The wizard uses the wizard interface (Figure 7.9) to control what is displayed in the service recommendation part of the interface. See: Figure 7.7. S/he has the ability to listen to what is happening as well as to see what is going on on the screen.

The difference between the main experiment and the control study is that in the main experiment the MAGUBI module is available and will appear to suggest suitable services, whereas in the control experiment the MAGUBI module is not available.

In addition to the above, the conversation between the subject and experimenter was recorded as well as any notes written down by the experimenter if so required. The subject was placed slightly in front of the experimenter so that the experimenter could see the screen of the subject, but not the other way around.
7. EVALUATING RESPONSORIA

Figure 7.7: RESPONSORIA at startup. Diagonally shaded area added to emphasize and denote area where service recommendations appear.

7.4.3 Ethical concerns

When conducting a Wizard-Of-Oz experiment, the very nature of the experiment itself dictates that certain knowledge is withheld from the user. Not sharing the knowledge that there is a Wizard, creates an ethical dilemma. To assure that the ethical policies of Linköping University were followed, the ethical review board was informed [45] and asked if a formal investigation was required. It was deemed that the experiment was simple and harmless enough not to warrant a formal ethical investigation [75]. After the experiment had been concluded, the participants were informed about how the experiment was conducted combined with a thorough explanation of the method used. The participants were then given an opportunity to erase all of their participation data should they so desire.

7.4.4 Experimental Procedure

Before starting the experiment, the user was informed that he/she would be testing a new system with and without a certain type of electronic support. The support was described as an A.I. or Artificial Intelligence system, and it was explained that the purpose of the experiment was to discern Qualitative differences (if any) between the system with and
without the A.I. enabled. Additionally, it was established that it was the system that was tested and not the participants, themselves.

The system layout and the functionality of the different areas was then explained to the subjects, as well as the different ways in which the services were listed. (Thematic, Alphabetical, etc.) Furthermore, the experimenter explained how to order a service by example. The subjects were informed about the scenario, as well as how the scenario was created. Further, the subjects were informed that they would be testing the system without support first and then with support enabled.

The subjects were informed about their role as a general leader over all the different branches of the rescue services, such as ambulance, fire department, and police. The experimenter then started to read from the scenario, and stopped when a point was reached in the scenario that demanded a service to be ordered. At this point, the subject was asked what service to order, then the scenario proceeded as before. About halfway through the scenario, the subject was informed that the A.I. was turned on and the user then started to receive suggestions from the A.I. regarding which services to choose. The subject then had the possibility of accepting or rejecting the suggestions from the A.I.
7. Evaluating Respondoria

After the scenario was concluded, the semi-structured interview took place and the subject was interviewed about the experience.

7.5 Analysis of experiment

After conducting the experiments, (Data Collection), the following phases of analysis were performed:

Coding In the Coding part of the analysis, different words that recurred were identified in the interviews conducted. These words were then sorted into different piles based on their semantic likeness. This process was then repeated to find themes, and then again to find concepts.
7.5. Analysis of experiment

**Interpretation** For the interpretation, the concepts were analyzed to find themes and dimensions on which to construct a model.

**Check for criteria previously established** The previously identified criteria and concepts were checked against the data to see if there were any anomalies.

**Validation** The findings were validated with the participants.

As can be seen, the order of the analysis is not the same as the order in which the experiment was conducted. The most significant reason for this difference is that we are dealing with a qualitative analysis. Although we have established some criteria for what to look for in the hands-on phase of the experiment, it is important to try and extract the significant qualitative aspects of the experiment through the interview analysis, before performing the actual evaluation of the hands-on phase itself. This ordering is done for two major reasons: Avoiding bias and enhancing the efficiency of the analysis.

Avoiding bias is important in itself when performing qualitative work. This way, the analysis of the hands-on phase is able to more accurately capture issues that the subjects themselves noticed. Also, it should provide for a more efficient analysis of the experiment, in that the number of iterations required to distribute the different observations into different qualitative categories should be reduced. After the iterations, we then arrived at results from the initial analysis, the model, and the previously identified dimensions.

### 7.5.1 Results from the analysis of RESPONSORIA

The following areas of interest were identified from the analysis of RESPONSORIA: Name of service, Symbols of a service, Interface to the service, The degree of accuracy of the Recommendation, The role and value of Wizard Of Oz studies in Service-Discovery Systems Development.

**Name of service**

The name of a service is the first step in understanding and using a service. The name may be interpreted in many different ways. For example “Information about object”, which in Swedish may refer to one or many objects. This service name, was interpreted as “many objects” by the research subject even though the service designers wanted to convey that there was only one object that was going to be used in the service. When interpreting a service name, it has a direct impact on whether or not users will elect to utilize a service. Hence, it is of paramount importance that proper naming is performed when constructing services. With proper naming
we refer to a name that tries to convey not only what the service does, but also such information such as how fast it process requests, how many users it can handle, and so on. Indeed, it is possible to imagine such things as dynamic naming of services to better convey status, load and so on. Furthermore, naming may also be done collaboratively and may include such things as ratings.

Symbols

Symbols are the next step in understanding a service and they are representative of different units/services/status. Sonnenwald [149] points out that there often are problems in a command setting when it comes to interpreting the meaning of symbols. If the meaning of these symbols are unclear, it may lead to potential disaster.

This is also culturally related, whether it is the culture of an organization, a major culture or another entity. For instance, when participants from many different types of different branches or sub-branches in the military are trying to work together, services and information shared might be interpreted differently by the participants from the different branches. Sonnenwald [149] states:

“individuals may not realize that they have specialized knowledge or skills that allows them to understand the implications of information that is considered basic in the one discipline but not in another.”

A shared understanding between the creator of the symbol and who selects the service is the key to selecting the right service. However, when a symbol is to be selected, it is often the creator of the service that is responsible for selecting an appropriate icon. This selection of icon may conflict with the user’s understanding of the service, and this is a challenge. This challenge may be overcome by allowing the users of the service to add symbols that they deem appropriate, once they feel they have an understanding of what the service actually does, and emphasizes the need for systems that quickly support correct service-identification and recommendation.

Interface

What became apparent while conducting the experiments was that having only a generic interface for a service is not beneficial to using the service. For even though it is possible to use a generic interface for each and every service, there are several shortcomings that this introduces. One is to invite misinterpretation of what the service does. With a generic interface such as a “To”-field and a “Message”-field, there is no possibility for the user
to deduce what the service is actually doing. Thus, it is possible to err twofold: First, by addressing the message to the wrong person; Second, by sending a message which is incompatible with the service in itself.

An argument can be made for interfaces that use the affordances concept from Gibson [51] or proper product semantics, to restrict the user’s ability to misinterpret what can be done by the service. Following the symbols, the Interface to the service itself is the next step in discovery of the service. A good interface will convey how the service is meant to be used and is in itself an extension of “service-discovery”. This implies that an important link in understanding and using a service is the service interface.

The perfect recommendation

In the qualitative study, the subjects were provided with perfect suggestions by the Wizard. By perfect suggestions, we mean suggestions that were in exact sequence and accordance with the sequence of services used in the scenario.

However, even though the system presents these perfect suggestions, it is not certain that the user will agree that this is the case, nor be in agreement with the ordering. So, there is no such thing as a perfect suggestion when there is a real user involved.

There are a number of reasons why this is the case. Among the foremost lies interpretation. The user may misinterpret what a service means, which leads to him or her not wanting to use a particular service, since the perception from the user may be that the service is something totally different from what is intended.

This also leads to the user not getting to stage three in the service-discovery process (Figure 7.10) where the interface has the potential to explain more about the service’s capabilities.

Another reason that may be combined with the first one, is that the user simply considers another service to be superior or more apt at the time.

The value of Wizard Of Oz in Service-Discovery Systems Development

A further conclusion from this work is that when designing a service discovery system a Wizard Of Oz study (See Section: 4.1.2) is warranted as a first step. The reason for using a WoZ-study before implementation is that in our experience, a test of a simulated complete system will yield insights that may aid in producing: 1. A more relevant and useful service discovery. 2. A system that is closer to being a second iteration system even though it is the first iteration.
7. Evaluating Responsoria

Figure 7.10: The continuing service-discovery process. **Step 1**: Needs are identified. **Step 2**: Service is identified for/by user. **Step 3**: The service interface is discovered. **Step 4**: The result of the use is discovered and interpreted. **Step 5**: User 1 evaluates the whole experience and may communicate this experience to User 2, as well as using the evaluation in subsequent interactions with the system. **Step 6**: User 2 uses the information from User 1 to form an opinion of the service before using it.

7.5.2 Suggested model

Service discovery does not end, it is continuous throughout the use of the service. It is continuous from the moment that the need occurs, either through internal or external facilitation, until the service has stopped being used. From the qualitative interviews as well as from observations from the experiments, the following model is proposed, see Figure 7.10:

**Step 1** Needs are identified. Either through a by-the-user perceived need, or through the noticing of a recommendation from the system.

**Step 2** A service is identified and localized by/for the user.

**Step 3** The service is about to be used and the interface becomes part of the continuing process of service-discovery. Here is an opportunity to include, for instance, flow charts of how the service is thought to work as well as semantic information in the interface.

**Step 4** The service is used, and how the result is interpreted ends up is a part of the future discovery of the service.
The lasting impression of the service use is the last link in the chain of service discovery and affects how the user is going to talk about the service with other potential users, and in the continuation of the potential use of the service.

### 7.5.3 The dimensions

In addition to the issues identified above from the interviews, the qualitative analysis gave additional insights.

Figure 7.11 shows the identified axes. On the X-axis is the amount of information that is available to the user and on the Y-axis is the amount of assistance available to the user.

Currently, the user is in the lower left quadrant. By employing systems like RESPONSORIA with MAGUBI, it is possible to make a transition to the upper right quadrant, which would greatly improve the user’s experience as well as providing the possibility to enhance the performance of the organization in accordance with what resulted from the interviews.

Looking into the specifics of each “island” we find the following: On the axis of Much assistance and No assistance we find the A.I. (Much assistance), and Manual (No assistance).

- **A.I.** The A.I. will provide assistance and suggestions with respect to service suggestions. Important aspects within this genre are:
  - **Correctness** Is the service suggestion the correct one, or is it not compatible with what the user deems correct? The concept of correctness is key here.
  - **New thinking** The A.I. gave rise to “new thinking”, or thinking outside the box, according to the interview material. This was due to suggestions of services/actions/devices that were deemed equally appropriate or correct, as the one that the user had in mind, but had not occurred to the user.
  - **Gain time** The opportunity to save time by getting relevant service suggestions was appreciated. This was particularly the case when the user was not familiar with what services were being offered. In terms of ecological validity, it makes sense that this is a useful feature, since it is highly likely that the users will be in a situation where all the services will not be known, but rather in a situation where there will be a significant number of unknown services.
  - **Timesaving** To save time is closely related to gaining time. Gaining time is actually a byproduct from Timesaving. However, timesaving is more related to speeding up already occurring operations.
Figure 7.11: Identified dimensions and properties in the service-discovery situation, and how the affect the ability to choose a service. Note that even with perfect suggestions/recommendations from a wizard, perfection is not experienced by the user.
7.5. Analysis of experiment

**Assign** Where to assign limited resources is important. A conclusion from the evaluation is that an additional aspect of service selection and recommendation is to assist the user in assigning the limited resources. In the experiment, just as in the adopted scenario, the focus was on finding the correct service. However, an additional help for the user would be if the system also took into consideration where the resources make the biggest difference.

**Simpler** Simpler or simplification of the selection process. When there is no AI-assistance, the selection process is considered more complex. It is viewed as easier to judge a suggested service as a good and appropriate suggestion than to actively come up with a selection by oneself.

**Good suggestion** The A.I. was considered to give good suggestions. This is not surprising since the A.I. was in fact a wizard suggesting services in line with the scenario.

**Suggestion** The A.I. did make a lot of sensible suggestions according to the participants. These suggestions were sensible for instance in terms of which category of services were offered. Furthermore, that it was suggestions/recommendations rather than a forced showing of a specific service was greatly appreciated.

**Order** A greater sense of order was experienced in the case when the A.I. was active.

**A.I.** The A.I. itself was the subject of much discussion. There was a general consensus of being impressed with the A.I.’s capabilities.

**Focus** The A.I. was perceived as enabling a greater focus on the task at hand. Participant: “I became focused.”

**Exactness** A feeling of increased exactness was experienced when it came to selecting the services. This feeling was not always due to the service suggested to be exactly the one required, nor the one thought of, but it was reassuring for the participant that the recommendation was either spot on or in the ballpark. It provided a confirmation that their own assessment was the correct one.
7. Evaluating Responsoria

- **Manual** Selecting services manually comes with some challenges. Below are comments on some of these issues.

**Evaluate & the Services themselves** Evaluating a service for selection is difficult for a multitude of reasons: As mentioned, the name of the service is the first hurdle. Another one is simply to find the service in the list of those available.

**Difficult to choose, Which Services, & Large selection** Because there is such a large selection, there is a real challenge when it comes to selecting a service.

**Much information** There is a lot of information to assess in any given situation. However, often this is unstructured and this lack of structure lends itself to uncertainty, and makes a manual selection of services harder.

**Recommendation** Even though exact suggestions for services are not received in a manual scenario, feedback from the subjects was that they appreciated a general recommendation as to which area of services to concentrate on, even though they had to select the service manually such as: “Current recommended area is: Police services”.

On the Little vs. Much Information axis we find Uncertainty (Little information), and Information retrieval (Much information):

- **Uncertainty** At the bottom of the spectrum lies the uncertainty group. This group shows the issues related to uncertainty when selecting services.

  **Clarification & Elucidation** A need for clarification & elucidation when selecting services. This issue is related to the uncertainty of knowing what a service does or is.

  **Incorrect** Uncertainty may also lead to an incorrect selection of services based on a wrongful notion of what the service does.

  **Question** The uncertainty leads to a multitude of questions that hamper service selection.

  **No communication** One issue identified when selecting what services to use is communication or lack thereof.

  **Conflict** Uncertainty may also cause conflict, between what services to choose or in which order to choose them.

  **Erroneous recommendations worst** With regards to uncertainty, it is also worth mentioning that an A.I. that gives obviously wrong recommendations creates uncertainty and damages the relationship between the user and the A.I. quite rapidly. One suggestion
for combatting this damage is to let the A.I. recommend more
general services or service areas, to increase the correctness.

- **Information retrieval** Below, we list the different aspects of information retrieval that we have identified in this setting.

  **General** Initially, general information or so called *overview* information is interesting for the operator. It was found that services that conveyed general information was preferred in the beginning of the scenario and as a new situation developed.

  **Specific** Consequently, services that provided specific information gained increasing importance as the scenario unfolded. The reason for the move from general to specific was that the operator expressed a desire to obtain an initial understanding of the situation in order to assess which services to be employed further on.

  **Identification** To be able to decode and process the information gathered as well as identify needs and services make for a speedy and correct selection of services.

  **Ambient information** Getting information from the surroundings and colleagues inside the situation room is an important part of being able to select services.

  **Collection & Gather information** Facilities and tools must be provided to facilitate processing and collection of information.

  **Information** A clear prerequisite for information retrieval is that there is some information to retrieve at all.

  **Get good information** The quality of the information collected is crucial to cater to correct decisions in service selection.

  **Flow** It was contended that the increased assistance from a log gave a better flow to the task at hand. “It felt like I had a buddy that worked with me” (Translated)

  **Observation** Provision of means to observe (the term is used loosely here) the situation is crucial in order to be able to retrieve information about it.

### 7.6 Summary of findings

In summary, in this section we have contributed the following findings to the body of science:

**General observations** The name of service has a direct bearing on what the user expects from the service and is the first decision
point the user has whether or not to even *click* on a service. Similar concerns exist for the symbols of a service or device, and this also has a direct impact on the Interface. Using a Wizard Of Oz study as part of designing this type of system is recommended, as it can catch mistakes at an early stage as well as generate a first iteration system that is closer to a second iteration system. Furthermore, we conclude that a user will never feel that s/he gets perfect recommendations all the time, even when the recommendations are agreed upon by human experts in the domain.

**Proposal of a model of continuous service discovery** Step 1: Identify needs. Step 2: Identify corresponding service. Step 3: The interface to the service becomes part of the service discovery process. Step 4: The service is used and influences the next discovery of the service. Step 5 and 6: The users evaluate and transmit their experience to future possible users of the service which in turn use the transmitted experience to influence their service discovery of the service.

**Dimensions/axes found in the service-discovery situation** With regards to the user service discovery process/experience, two axes were identified: Axis 1: Level of assistance. Axis 2: Level of information. Axis 1 determines how much assistance the computer offers the users in selecting the service. Axis 2 determines how much information the users have about the service offered.
Chapter 8

DISCUSSION

From a mainly Ubicomp perspective this chapter discusses: The Nature of services, Service presentation and development, and Other issues.

The chapter revolves around the different stakeholders in service discovery: the user, the service designer/programmer, and the service provider. All of the sections in this chapter provide useful information for the above groups. For issues focusing on the user we recommend Sections 8.1.3, 8.2.3–8.2.4, 8.3.2–8.3.4, 8.3.6, 8.3.8–8.3.9, for designer/programmer-issues Sections 8.1.1–8.1.2, 8.1.4, 8.2.1–8.2.2, 8.3.1, 8.3.5–8.3.9, and for service-provider issues Sections 8.1.1–8.1.2, 8.2.1–8.2.2, 8.3.2–8.3.5, 8.3.8–8.3.9.

8.1 The Nature of Services

Here we focus on Service properties, Resources, Extra system vs. intra system support, and The implications for designing ontologies for service discovery in ubiquitous computing.

8.1.1 Service Properties

Services can appear in many forms. Moreover, it is not uncommon in the service discovery field that devices are viewed as services. For instance, a headset in Bluetooth provides the service headset. This makes sense when the headset can be seen, since the users can make the mental connection between the headset and some (or all) of the potential services it might render. It is therefore important that service discovery systems reflect user expectations as well as real technical data. A possible discrepancy between the perceived properties of the service and the actual properties can lead to inconsistencies in usage. It is therefore necessary to try and correctly anticipate such properties. Our suggestion for these properties are as follows: The degree of physical manifestation: There might be real
8. DISCUSSION

physical services, such as a printing service or a restaurant service, but there might also be entirely virtual services such as a weather information service. *The degree of closeness:* Certain virtual services might be perceived to be as close as certain physical services. *The degree of uniqueness:* If there is only one instance of the service it is unique and vice versa. *The degree of cost:* The monetary or time expenditure associated with the service. *The degree of delimitation of the service:* A remote control can control a multitude of services whereas a setting knob on a radiator only controls that particular radiator. *The degree of time:* If the service is with or without a start in time and/or with or without an end in time, i.e. if it is continuous, instant, or perhaps intermittent or delayed. *The degree of initiation and initiator:* It is important to know if this service was passively received, consciously searched for, or initiated on the users’ behalf by the system. On occasion, other services may initiate a service. *The degree of load on the service:* A service that on paper, is better than another, might under heavy load perform worse than a service that, on paper, is worse but that has a lighter load. For example, a slow printer with a non-existent print queue will produce the printout faster than a fast printer with a very long print queue. *The degree of popularity of the service:* As seen in Section 8.3.3, the popularity of a service can be seen as an indicator of how satisfied users are with that particular service. However, a service might be popular for many of reasons, some of which might not necessarily be related to the actual performance of the service.

8.1.2 Resources

When using a service, resources are consumed. The resources consumed may be virtual or real. Regardless, there is a limit to their availability and the amount that can/may be used per user. Cancelling services, Booking multiple services, and Getting the wrong kind of services, are all related to matters of resources.

**Canceling services**

Using services is not all about finding or suggesting the right ones, but is also about canceling, especially when utilizing a service is associated with a cost. When starting to use a service and then realize half way through that the service is not necessary, it must be easy to cancel the service. In fact, a smart service discovery system may suggest canceling services based on what events that are have transpired, or even automatically cancel services under certain circumstances.

In a C² system for rescue services it is easy to imagine a scenario where there is a fire alarm at point A, resources are dispatched to the scene, but
before they arrive, the alarm is deemed false and the resources are turned back.

However, users or systems may be committed to different degrees to a service, depending on how far along into the usage of the said service they are. Therefore a premature cancellation may also be associated with a varying cost, and this may also influence which resource should be sent, especially in the case of resources, which as in the case above, may simply be rerouted, rather than it being necessary to dispatch new resources from the original location. Depending on the type of service, there may be several reasons for this rerouting, but in the above example, it is likely that the reason is speed. At other times the reason may be one of cost, even though the delivery/completion time of the service may be longer/slower.

**Booking multiple services at once & their cost**

A likely development in the future is that service discovery systems will book/entertain several services at once. It is reasonable to assume that this possible usage will be tied to the potential cost of the services used.

In the above statement we can already start to see market forces coming to bear on the service landscape. In the future, it is probable that services with have the same division between free and paid that applications currently have. It is also possible that ad-supported services will exist in a similar manner to ad-supported applications today. Like purchasing the application today, there may be many different types of subscriptions, such as pay-as-you-go, time-based subscription, one time fee, and so on.

Moreover, the cost for a service and/or locking-in a user in a subscription leads to a marketplace similar to the selling of produce on the town square. Future service discovery systems will need to support price negotiation as well as be able to quickly adapt and find alternatives when prices go up and down as supply and demand fluctuate. This price fluctuation is a key difference between today’s application market places and the future service market places, since the latter will be much more high-paced.

**Getting the wrong kind of service**

Superficially, it is tempting to let a service discovery system recommend a service that exceeds the desired properties of a service. However, there might be tacit properties that have not been specified and/or that are in direct conflict with the ones specified.

One example may for instance be when renting a car and the system suggests a free upgrade to a bigger car. The user might reject this offer since s/he knows that the fuel consumption of such a car is higher than...
accepted. Indeed, the user may prefer to be downgraded to a smaller car instead if it has a lower fuel consumption.

Analyzing this erroneous upgrade suggestion, we conclude that the service discovery system has failed in multiple ways. First, there is a lack of information in the user/world model and/or rules. The system does not know that the user prefers a car with a low gas consumption to a free upgrade. However, it is reasonable to assume that this type of error will always be present since a system cannot know everything.

Second, a way is needed to add information to the user/world model when something goes wrong as well as to be able to update any rules that may exist. This updating may for instance be done by asking the user to provide reasons why s/he refused the alternative service offering.

8.1.3 Checklists as a way to select the right services

Checklists are a way of keeping track of what needs to be done, (which services to use), in many C² situations. However, checklists by their very nature can be viewed as external to the system rather than as part of it. What is part of a system and what is not is a question that has been debated throughout the history of science. Our opinion is that if something can be gained in knowledge or understanding by considering it separated from the system rather than part of it, then the separation is motivated. This separation has been debated for instance when it comes to whether or not to include the human as part of a system. In our case, there is a question of Extra system vs. Intra System support.

Macaulay [102] posits that checklists may be viewed as a tool for reflexivity. In certain settings the checklists may afford orientation without prescription. However, in situations such as starting aircrafts, controlling nuclear reactors, and handling accidents, checklists prescribe what to do and in which order.

As a checklist is normally on paper its physical nature is an important element. Being a physical item, it carries with it certain properties and affords certain actions. For instance, as described by Bång [12], in an emergency room setting, it is not only the information on a medical chart in itself that is important, but also how it is positioned in the room and on the surfaces. Informal structures are created where for instance the position of a chart conveys information about the progress of the treatment of a patient. This way of conveying information by placement of the paper based artifacts is also true in other settings such as in aircraft, where placing a checklist between the seats of the pilot and co-pilot, indicates that the checklist is not complete [100].

One drawback of checklists is that while in one sense they allow for a changing situation, (items may for instance be omitted), they are not
dynamic in the same sense that an open system is. For example, in a take-off situation, the Auxiliary Power Unit (APU) of an aircraft, is normally scheduled to be turned off before take-off, but in certain situations it may be required to turn it off after take-off, thus creating a situation where the checklist is not valid [100], and the pilot would have to remember to turn off the APU. In contrast, a dynamic (intra) system would know what needs to be done and is context aware, (such as at which airport the aircraft is currently).

In situations that are unique and highly dynamic, (such as emergency management), the checklist provides a sub-optimal tool for management. It is unable to adapt to specific situations and will provide a more generalized approach to solving the problem [39, 66]. An additional possibility is to include checklists within systems such as MAGUBI and let the checklists be dynamically created. Relating to the question above in this way we allow the checklist to become part of the system.

Clearly there is a case to be made for a system such as MAGUBI, which by being dynamic, avoids many of the problems that checklists have. However, we note that checklists and pen and paper remain superior in extreme conditions, when all other technology stops functioning.

8.1.4 Implications for designing ontologies for service discovery in ubiquitous computing

When designing ontologies in this Ubicomp situation, there is a trade-off between larger, more comprehensive ontologies and smaller lightweight ontologies. The trade-off is between expressiveness, maintainability and how heavy the ontology is to process.

It is our conclusion that, for service discovery in ubiquitous computing, a big, heavy ontology of the CYC-type is probably not a viable solution. Within a smaller ontology, a possible way to get a limited, deeper coverage of a particular focus is to use ordinary people and a constrained automatic gathering process to filter out erroneous entries. Three projects that used this approach were Mystic [167], Mindpixel [146], and Open Mind [150]. More recently, a similar solution in the current setting is the approach taken by MIT with ConceptNet [89]. ConceptNet is a semantic network automatically constructed from the Open Mind Common Sense Project. ConceptNet does not have the logical framework of systems such as CYC, but instead emphasizes semantic connections much like WordNet [111]. A straightforward example is when ConceptNet puts your microwave on defrost when you take out an item from the freezer, since the concept of freezer is connected to the concept of microwave in the ConceptNet. There are, however, two caveats to consider when looking at ConceptNet and similar approaches: ConceptNet lacks the explicability of a well-organized
ontology. It only has the ability to say: “X is related to Y.” CYC, on the other hand, can motivate why it reasons in a certain way.

von Ahn and Dabbish [162] used this type of information gathering to collect labels for picture-data. The strength of this type of collection is that it is quick for amassing great amounts of data. However, the drawback is that there is no human control of the resulting structure and it is more difficult to construct a more descriptive structure. The contrasting approach is to have a staff of ontologists constructing the ontology, such as in the case of CYC.

A further conclusion and implication is that when designing ontologies, reasoning systems and the like for use in service discovery for ubiquitous computing, it is very important that they are designed so that they are easy to reuse and adapt, since they otherwise will be ignored because they convey obscurity and because of the massive effort required to begin to use them. Another recommendation is that the benefits and contributions that the ontologies might bring to a specific Ubicomp project must be very clear, otherwise once again they risk being ignored. The issue of extendability/scalability is discussed in Section 6.3.6. This issue is integrated into the grounding problem: To get the system to cope with as many novel situations as possible without the need for an ontology that tries to cover everything, is important.

From building our systems, we can conclude that it is possible to build an ontology fairly quickly for a limited domain. However, this is not something that is easy for the layperson. Consequently, there is a need for ontology construction systems which cater to the said layperson. A possible way forward is a Wikipedia-like [168] ontology creation system which has a simplified language to create ontologies and also makes it possible for incremental change to the ontologies by the community in which they are used.

8.2 Service Presentation and Development

Here we discuss: Service discovery versus user discovery, Support for developers and service creators, The smartphone as interface to the world and the services in it, and How to present appropriate services to the users.

8.2.1 Service discovery versus user discovery

If the whole notion of service discovery is reversed, it becomes user discovery. This approach is an interesting step beyond current solutions. More precisely, the issue is not so much “the users finding services and devices” but rather “the services and devices finding the users” or “users finding users”. Expanding on this subject, we can compare this task to
8.2. Service Presentation and Development

what a headhunter does when finding a person with the right competence for a task. If applied to a Ubicom setting, for instance a military command setting, we can see that finding the right person to perform a task is crucial to a successful campaign.

A possible configuration for user discovery is to allow certain methods for the system to access the user. With the lunch scenario (Section 6.3.2 on page 96) as an example, using user discovery would mean that restaurants need to convey their menu somehow to Howard. One way of letting them do so is to allow a special area on the wall screen to be accessible to services that want to present themselves to Howard. At lunchtime, this area would prioritize services that wanted to present lunch alternatives.

The print situation is another example where user discovery may bring advantages. When printing, choosing a fast printer with a long print queue, might result in having to wait longer for a printout than if a slower printer without any documents in queue is chosen. By assuming the role of printer consumer, the user can become the receiver of the services instead of the initiator, and a printer that has a short or non-existent print queue can search for the user instead of the other way around.

8.2.2 Support for developers and service creators

As mentioned above, until now there has been a high degree of focus on enabling connectivity for services and less focus on providing the programmer/designer/developer with a model to announce and present his/her services. Due to the lack of research in this field, we find ourselves in a situation where the creators know very little about how their service is going to be presented to the users, and are therefore not in a position to do anything about it. Potentially, there is much that can be done about this.

If standards for how to accomplish this presentation were to be incorporated into the programming APIs\(^1\), and IDEs\(^2\) were tailored to include service presentation in a ubiquitous environment, presumably the creators would have a much easier task in presenting his/her service to the users. For DUs\(^3\) this is also important. Models of what goes where, or for that matter the ability to control what goes where, or at the least knowing that \(X\) is definitely not going to end up on \(Y\), is important for the people designing the DUI, whether they are UI-designers or programmers or a combination thereof.

One possible approach to fulfilling the definition of adaptive presentation of DUs and services would be to do it in a way similar to \LaTeX\ [84] That is to say, the creator specifies the logical structure of the service and its interactive capabilities, requirements and restrictions when programming

\(^1\)Application Programming Interface
\(^2\)Integrated Development Environment
\(^3\)Distributed User Interface
8. DISCUSSION

it, but the system decides when, how, and where to present it. This would place very little extra burden on the programmer, and at the same time simplify for the system, since it would have a clear logical structure for where to put things.

One solution would be to give the programmer a kind of WYSIWYG\textsuperscript{1} interface to the service-discovery/handling system. An analogy to this would be the layout managers in, for instance, Java or Cocoa. In this service presentation builder things such as: “this must be presented in color”, “X has to be presented with at the most 20 cm lateral spacial separation from Y,” “if this cannot be presented to the users at eye-level, it must be read out aloud instead,” could be clearly shown and specified. Without catering to the developer’s and creator’s special needs for services in ubiquitous computing the said services will not be created.

Mashups and combinations of services are likely to become more commonplace as the service-discovery field matures and becomes more interconnected with our daily lives. To support this development, the development of services and the service discovery systems that manage them, must be done with agility in mind. Furthermore, we can be certain that the flow from utilizing one service to another will more often than not be new and loosely coupled. In the future, service discovery will be more about setting the services in the right order, rather than finding them.

8.2.3 The smartphone as interface to the world and the services in it

It is evident that the smartphone is emerging as the interface to the world and its services. This emergence will put pressure on service providers in the following fields:

\textbf{Speed} A service has to be fast in response time and turn around time once the request has been made, otherwise it will not be used or be selected away from.

\textbf{Understandability} As mentioned in this thesis, the name of a service has an impact on how and if it will be used. The same is true for the interface, results and other HMI-parts of the service. A further conclusion is that it may lead to faulty use of the service or improper interpretation of the results.

\textbf{Usability} There is a connection between understandability and usability. The difference we would like to emphasize here is the speed, effectiveness, and ease-of-use of the interface.

\textbf{Discoverability} If the service is not discoverable it will not be selected.

\textsuperscript{1}What You See Is What You Get

154
The above points need to be optimized for use in a smartphone together with a service-discovery system that also is tailored towards a smartphone.

Note that also device eco systems are developing. De Facto standards for certain services such as Apple’s Airplay for wireless sound playback are emerging, and it is very likely that more manufacturers will seek to lock-in the consumer. We already see that the smartphone is taking the place of the remote, for instance within home cinema receivers. The future service discovery systems will have not only to cater to this situation, and ideally present pre-configured combinations of devices and services into aggregated devices and services within the eco system, and also provide easy interfaces for the users to create their own combinations.

### 8.2.4 How to present appropriate services to the users

When deciding how the appropriate services are to be laid out and presented, a multitude of factors, as well as a recursive SD-issue are involved. Imagine, for instance, that our user enters a room, and wants to know what services are available before service discovery, reached its current state, it was necessary to look through all the services available and decide manually which service to utilize. With MAGUBi this task would be significantly reduced.

A lot of effort has gone into the first two steps of service discovery, namely the technical solutions for establishing contact and communication, and with ODEN and MAGUBi, evaluating and reasoning about which service to select. Having shown that these challenges can be met (at least in the limited domain we have focused on), it is time to push ahead and look at the next research issue; different ways of presenting the discovered services in the best manner to the users. In our opinion, this issue is a subject that is worthy of attention and will play a major role in the future services-oriented world. Research conducted in cognitive science and biology has produced a number of findings in relation to this. Most notably, the notion of affordances by James Gibson [51], i.e. the concept that the appearance of an object communicates to the users about how to use it. A chair for instance affords sitting. The concept of affordances can be beneficial when thinking about the challenges ahead for service discovery. It is, however, difficult to directly translate the notion of physical affordances from a physical object to virtual affordances from a service, although it is a good starting-point. At the moment, what these service-affordances are is an open question. However, the different properties of services that we mentioned in Section 8.1.1 hint at what might be deemed important characteristics for these affordances. Furthermore, due to the Ubicomp nature of these services, the use of different modalities is most likely going to play a role [133]. Hopefully, the ability to combine different channels and modalities when presenting and interacting with the services...
A significant amount of the work in service discovery has understandably focused on web-services. In one sense, these are located a step away from the user. What is done when the web service is found and utilized and has received the information is less well researched. This is especially true vis-à-vis the users. That is to say, the relationship between a user and a service from a service-discovery perspective is something that it is necessary to incorporate into a service-discovery system if it is to be successful. If a service is tied to a physical object, maybe that object in itself can be augmented or enhanced in some way as to radiate details about the service/s it offers access to. This conveyance of information has been explored in ambient-computing.

In the case of a virtual service, however, it is harder to know how to advertise and interact with it. One possible solution is to advertise the service through an object with which the users could interact with the service. Using conventional thinking, this advertising has the potential of resulting in all services being presented on the users' PDAs with the possibility of a massive interaction chaos as a result. There is a danger that the interfaces presented are different in every instance, even if they are connected to the same service. This challenge also exists for services that use conventional means. How the services are presented will influence the use and reasoning around them, as shown in our WoZ-study. (For instance the name of a service.) Consequently, one challenge and research issue for future designers and architects will be how to include general ways of interaction with services into the common living environment [76], since much of tomorrow’s living is going to be in a very service-rich, pervasive-, ambient-, Ubicomp-environment.

### 8.3 Other Issues

In this section, we examine security, privacy/ethical issues, measuring success for service discovery from a user and organizational perspective, observations regarding complex systems, rogue service, service descriptions, wiki interfaces, formal rules vs. evolving rules, a choice between a transparent decision process or not, and intelligent systems vs. task guidance systems.

#### 8.3.1 Security

Knowing if a service is what it claims to be, and if it will do what it claims to do, is part of deciding whether or not to use it. Subsequently, the evaluation of services from a security perspective and the handling of rouge services
is an important part of the future of service discovery, if the user is to have a satisfactory experience and if the adoption of these types of systems is to become widespread.

**Evaluating a service**

Before using a service or device, most of us spend some time evaluating it from different perspectives. This might be in the form of a visual inspection, making sure that the service/device is not broken, fraudulent, or perhaps mismanaged. This becomes more important, the more expensive the service/device is. Another way of judging the quality of a service might be through word of mouth, or by endorsement by a trusted third party. Similar approaches can be seen today on the web when it comes to transactions and websites.

**Rouge services**

Rouge services can have many meanings. One meaning is that there is a possibility for freeloaders on a service. This possibility has existed for as long as services have existed. Often, freelading on a service can be made possible by the very same service that it is providing. For instance, on a network, there is a possibility to use already existing authorized resources — such as laptops — to create wireless DHCP based access points, thus allowing potentially unauthorized clients to become part of the network.

Another meaning of the term *rouge service* is when a service is offered that is not wanted. (Service spam if one so will.) This is especially damaging in contexts where limited resources are available, such as network or processing power. While not the focus of this thesis, the concept of rouge services nevertheless has an impact on service discovery systems. We would like to highlight this issue as an important challenge in the years ahead for service discovery. It is highly unlikely that automatic service discovery will be utilized in a situation where the automatic service discovery system lacks means of dealing with them.

Yet another meaning of *rouge service* is a service that in itself creates problems for other services by utilizing them too much. This over utilization may be in the form of making too many requests in a short time span for example and needs to be identified and prohibited.

**8.3.2 Privacy/ethical issues**

The delicate balance between what is efficient and what is a breach of privacy is something that designers of systems that handle people’s data have always struggled with [2] and will most likely continue to do so in the foreseeable future [16]. For instance, Lederer et al. [88] use a metaphor
called *faces*, to describe how people present themselves in the virtual world. This metaphor is inspired by Goffman’s [52] research into how people present themselves in society.

In systems such as MAGUBI and ODEN, the issues are different from those of a banking system for instance. Even so, it is wise to consider the possible implications for privacy violations. Knowledge about the user will have to be stored somewhere. It could be in the form of a general ontology, but then we run the risk of it being too general and not benefitting from user-specific information. On the other hand, a too specific ontology runs the risk of becoming brittle.

Furthermore, a specific ontology becomes potentially dangerous if it falls into the wrong hands. This would suggest that the current approach whereby we evaluate the service offerings on the personal device, is correct in a setting where personal data cannot be trusted to be sent over the network to be processed by other devices. Moreover, this points towards the need to have public profiles, like the *faces* mentioned above, that can contribute to the overall quality of service discovery and only share such data as the user deems it is correct to share.

The challenge is to combine an effortless Ubicomp experience with privacy. Lederer et al. [88] suggested that the users should review logs. However, this task is time-consuming and users cannot be expected to do this in every situation. Constantly asking the user for consent is likely to result in the user switching off the security system. In addition, reviewing logs addresses the problem after the fact, rather than in advance. A community-based filtration system where known users recommend and vouch for services is a promising direction for the future. However, as a side note, we observe that one possible location for a security log could be being inter-spliced with the current general log of our system, since the general log is a feature that was requested and was also well received. However, we caution that further testing is required to evaluate this possible placement, since it may produce too much security logging data and overwhelm the other entries in the log. A possible way around this is filtering, or that only certain operators can see the security information.

### 8.3.3 Measuring success for service discovery from a user and organizational perspective

To date, the focus has been on the infrastructural as well as the hardware aspects of service discovery. As the field of service discovery matures, new issues emerge. As hardware and software problems are being resolved, metrics for success from a user perspective within service discovery will become increasingly important. This is due to a number of reasons. One is a more traditional reason; to reduce the overall load on the infrastructure and software. Two newer reasons are to reduce the load on the human and
to increase user satisfaction. In this setting, it is important to remember that, for instance, faster response times and the ability to find all services do not automatically lead to a more satisfied user [77], but that other, less tangible factors such as enjoyment, type of interaction and design might play a significant role in the success of a system, and may sometimes even lead to the users choosing a system that in actual performance (for instance words per minute) is lower than the subjectively less attractive system.

One possible way of getting user satisfaction metrics for service discovery is to measure how many of the services discovered and presented to the users were actually used. Used services could automatically be classified as correctly identified and presented. This approach contrasts with the more traditional metric which is the ability to find all available services in a given situation. The idea for this new metric originates in the enhanced semantics of ODEN, as well as the common sense rules of MAGUBI. The incorporation of these metrics into MAGUBI's ontology and the rule engine is possible because of the pluggable nature of said systems.

An important part of metrics is the ability to stifle the erroneous usage of services. An example of this is when an emergency service becomes the norm due to popular demand. The ability of the service provider to affect the service metric to decrease usage is thus an important aspect of the system. Finally, measuring features such as coverage and speed is also important when comparing different rule sets and ontologies.

What users deem as a success metric is not necessarily a success metric for organizations. For instance, the ability to print many copies on a printer is beneficial for a user; however, for the organization, frivolous printing leads to unnecessary costs.

Logs as part of the service discovery system and its success

In a multi-user system that has logs, it is the logs that provide a resource if mined for data concerning service usage. In combination with an aggregation of the world model, as well as the user model, this data mining can, in addition to serving as a receipt for successful service discovery, provide valuable suggestions as to the order in which services are selected as well as which services are suitable for a certain situation.

8.3.4 Observations regarding complex systems

It is extremely unlikely that a system would be able to prevent mistakes from occurring when selecting services. However, two things are important to keep in mind while developing these systems:

- Things go wrong sometimes. Alfred Holt remarked:
It is found that anything that can go wrong at sea generally does go wrong sooner or later, so it is not to be wondered that owners prefer the safe to the scientific. It is also found that it is almost as bad to have too many parts as too few; that arrangements which are for exceptional and occasional use are rarely available when wanted, and have the disadvantage of requiring additional care. Their very presence, too, seems in effect to indispose the engineer to attend to essentials. Sufficient stress can hardly be laid on the advantages of simplicity. The human factor cannot be safely neglected in planning machinery. If attention is to be obtained, the engine must be such that the engineer will be disposed to attend to it.”

– Holt [68, p.8]

- The causal factors, or known precedents, of the thing that goes wrong are rarely, if ever, new. Miller [110, p.60] Ferry [46] These known precedents, are catalogued and used when trying to prevent similar accidents in the future. See Figure 8.1.

Hence, there is support for identifying and addressing these causal factors. We suggest the following approaches:

1. Identification of the factors that cause the wrong service to be suggested or selected. This identification is a task that is going to become more important as we move more into service based, ubiquitous, and ambient computing.

2. Suggest services that in themselves try to minimize the risk of things going wrong. Here the challenge is to strike a balance between a
service that is focused enough to get the job done and at the same
time be general enough not to be brittle when faced with challenges.

8.3.5 Service descriptions

Service descriptions should relate to the user of the service. If the user of
the service is a human, they should relate to the human. If the user of the
service is a machine, they should relate to the machine. When we use the
word relate, we mean that the description should contain information that
is relevant to the user.

Furthermore, we need to adapt the service description to fit the data
that is readily available from sensors as well as to continue to use well
established conventions. It is with relative certainty that we can conclude
that sensors to determine many kinds of information about a user are
today becoming readily available. For instance, today’s cellular phones,
contain many different sensors that continuously record data, but that
these sensors are so far underutilized. As we have seen, sensors such as
accelerometers, temperature, GPS, compasses, cameras, Bluetooth, Wi-Fi,
and so, on will become commonplace. There are already services, (such
as Google Latitude) that allow users to share their position with friends
through the combination of GPS and online services (see Figure 8.2).

This large amount of sensor data presents a rich opportunity to opti-
mize service selection. It has also been demonstrated that a considerable
amount of information can be deduced from simple sensors [9] such
as accelerometers. Using these sensors, we are able to deduce many
different facts. For instance: the rate of turning (compass), movement
(accelerometer), and so on. From these sensor data we may extrapolate
8. Discussion

A number of properties. These properties then become interesting to model and to include in our service descriptions.

For example, to facilitate selection it is advisable to include information into the service discovery system about the kind of activity for which the service is best suited. Then, based on which activity is inferred from the sensor data, a proper service selection can be made.

8.3.6 Wiki interfaces

As has been demonstrated in Chapter 7.1, service discovery systems play a crucial role in maintaining a high degree of situational awareness, for example in command and control systems. Since service discovery is a hard challenge, we cannot expect the service discovery systems to do all of the work for the users when it comes to locating and selecting the services that the users want to utilize. However, with some help from the users, a sophisticated system such as MAGUBI makes it possible to achieve good results.

Since much of an application’s performance in use is dependent on the user interface, the UI in itself has a big part to play in service discovery. The novel approach here is to enable the users to construct the interface jointly through wiki-like techniques. One way of doing this is to let the user specify the generic type of information they wish to fill the specific interface part with. For example, if a user needs a video feed from a site of an accident, s/he specifies this type of content at a particular part of the interface. It is then possible for the system to suggest what might be appropriate to fill the designated part with. Another possibility is that another user fills the space with the appropriate content. In essence, the UI may be constructed in concert by different users collaborating. In the case of the UI with the video feed, another user may decide to add a request for a chat. This request may then be filled again by the service discovery system or by another user. In this way, an interface based on the same principles as Wikipedia [168] is constructed.

8.3.7 Formal rules vs. evolving rules

One of the weaknesses of a system based on rules is that the rules are mostly static. This static state makes sense for common-sense rules, or rules involving facts that are static. However, it does not make sense in situations where there is a need for rules which change. By incorporating approaches such as the one from Yang et al. [177], we believe that it is possible to obtain an improved performance.

Delving deeper into the previously mentioned (Section 2.3, page 25), BPEL-connected research, we note that according to Grigori et al. [56] research is currently underway to automatically analyze the graphical
structure of a business process, in order to enable selection of similar services. Similarly, Xiao et al. [176] analyze the behavior in BPEL-processes and produce behavior templates, which are then used to match similar BPEL-processes and Wombacher et al. [169] illustrate how transformation from BPEL to annotated deterministic finite state automata can accomplished.

8.3.8 A choice between a transparent decision process or not

A common theme when computers make suggestions, selections, or use AI is whether or not the decision process should be made visible, and if its is made visible, should it be visible as default or just when requested? An alternative to visualizing the decision making process is to visualize the constraints that affect the decision making process in joint cognitive systems, such as the FRAM model [67]. Hollnagel [67] models input (I), output (O), Preconditions (P), time (T), resources (R), and control (C). It is possible to link the output from one function for instance to the input of another. Our argument then is that the system should visualize the effects of choosing one service over another rather than attempt to have the system show the reasoning behind the selection of a specific service.

8.3.9 Intelligent Systems vs. Task Guidance Systems

Ockerman et al. [123] mention the difference between user interaction while using an Intelligent System versus user interaction when using a task guidance system. The way Ockerman et al. [123] describe it, an intelligent system is situated between the human and the task, and everything the human does is filtered through the system. The intelligent system also has sensory information from the environment and actuators to influence the environment. (See left part of Figure 8.3.) In contrast, a task guidance system is situated to the side of the environment and the human. (See right part of Figure 8.3.) Thus, the task guidance system has no way of influencing the environment and also has no knowledge about the environment.

MAGUSB falls somewhere between these interaction styles. Although in principle the system cannot be prevented from interacting with the environment, the design is one where influence on the environment from the system is not allowed. Furthermore, as shown in the right part of Figure 8.3, MAGUSB includes access to sensors, which is an adaptation and addition to the interaction that is described by Ockerman et al. [123].

Subsequently it would be permissible to classify MAGUSB as an Intelligent Task Guidance system. In conclusion, we believe that the future lies where MAGUSB is situated in a blend of the two types of systems, with the
addition of the possibility for the system to engage in select tasks with the environment.
Chapter 9

SUMMARY & CONCLUSIONS

In this chapter we summarize the experience from building three systems, which implement a novel two-layered service-discovery architecture, and the empirical investigations of its appropriateness for providing good advice to a user. In addition, we draw conclusions from the work.

9.1 Summary

As stated in Chapter 1, this work addresses the challenge of:

“How to find an approximation to ideal service discovery in terms of achieving an optimized balance between searching for, and presenting the user with the very best service alternative (according to given criteria), and the usefulness of a good-enough support service as experienced by a user.”

We address this challenge by analyzing and describing service discovery, and we finally create + evaluate a novel architecture consisting of two layers:

1. The Enhanced Traditional Layer addresses one of the shortcomings of the regular systems by allowing richer and more human-oriented service descriptions.

2. The Relevant Service Discovery layer addresses a further shortcoming of the regular systems, by incorporating world and user knowledge thus enabling more informed decisions regarding which services to look for, and present to the user.

These layers are realized and verified first in the ODEN and MAGUBI systems, respectively. Finally, they were integrated into the RESPONSORIA
system. This is a proof-of-concept system that uses the domain of command and control in a rescue-operation setting to verify that the approach works in practice as well as in theory. The final stage is the wizard-of-oz evaluation of RESPONSORIA.

9.2 Conclusions

We believe that significant progress has been made on the technical aspects of service discovery, such as connectivity, and that it is time to shift towards user-centered service discovery. From the work with The Enhanced Traditional layer and ODEN, we conclude that it is possible to develop service-discovery systems that are agnostic with respect to networks, and that are Peer-to-Peer enabled (see Section 5.3). This approach allows users to search for services and devices in a more detailed and fine-grained manner than before, as well as letting the systems reason about the services found before presenting them to the users. An additional conclusion from this work was that to further enhance the user experience, a thorough processing of the results is required, as well as a proactive search for services to benefit the users by providing timely and relevant discovery (see Section 5.4).

To attain this user-oriented, proactive service discovery, we developed and used an ontology for a small subset of the world. However, the creation and maintenance of such an ontology is difficult and time consuming. At the same time, this new level of service discovery is required if service discovery is to reach a pro-active and user-centered state that enables the users to discover services that they did not even know of, or realize that they had use for. We conclude that to attain a dynamic, high-level service discovery, it is necessary to find ways for enabling collaborative co-creation of ontologies or structures that provide similar functionality (see Section 6.3.6).

Based on on-site interviews and studies, we have identified two axes and subsequently four positions that users may find themselves in when using these types of systems (see Section 7.5.3). The two axes identified are Information and Assistance, see Figure 7.11 on p. 142. The current systems leave the users in the situation of little or no information in conjunction with no assistance. Systems such as RESPONSORIA and MAGUBI on the other hand move the users towards the quadrant of a large amount of information and significant assistance. The other alternatives are much information without assistance, as well as assistance without any information. In these cases, the users may end up overwhelmed by the amount of information as they have no assistance, or baseless recommendations.

An additional conclusion is that there is a continuous service-discovery process, see Section 7.5.2 on p. 140. In other words, the service-discovery
process never ends. It takes place, before, during, and after the use of the service. An example of this is that the service-discovery process may be transitive. That is to say, the discovery and quality-of-use of services may be shared through the person who has used the service, to other people.

A further conclusion from this work, is that a Wizard Of Oz study (see Section 4.1.2) is in general a warranted step when designing service-discovery systems and in particular, for Service Oriented Architectures under development. The reason for using a WoZ-study before implementation is that, in our experience, a test of a simulated complete system (populated with mock-up services) will yield insights that may aid in producing (1) a relevant and useful service discovery, and (2) a system that upon its release has a matured user-interface.

The final result from this work is that even if we put the impossible aside, and engineer a situation where we supposedly find all available services, such as in the WoZ-case, we are still left with the last part of discovery, that is presenting the results to the user. For example, crucial to the users’ selection of a service is that they can map the name of the service to the expected outcome. In addition, we note that there is no such thing as the perfect service to be recommended. During the experiment, the users did not agree 100 percent of the time that the service or services presented were the most appropriate ones, even though they were presented with results certified by experts as the most correct ones. However, this discrepancy is true even in normal situations when there is a human performing the recommendation. It may, from time to time, be prudent to remember this fact when we try to get close to, or exceed human performance in computers.

Nevertheless, we see a bright future ahead for service discovery, in particular in the Ubicomp field. Smart-phones and other mobile devices are rapidly becoming the way in which we interact with much of the virtual and physical world. It is clear that the support that these types of service-discovery systems can bring is welcome and, in an increasingly connected world, they are also a necessity if we are to utilize the promised potential. These devices are increasingly aware of the users’ intentions due to a higher degree of automatic information collection and aggregation that now exists. This increase in information leads to a development much in line with the proposed service-discovery architecture, especially the relevant service discovery layer, presented in this thesis.
9. SUMMARY & CONCLUSIONS
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


<table>
<thead>
<tr>
<th>No</th>
<th>Author</th>
<th>Title</th>
<th>Year</th>
<th>ISBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Michael Reinfarken</td>
<td>Fundamentals and Logical Foundations of Truth Maintenance</td>
<td>1989</td>
<td>91-7870-546-0</td>
</tr>
<tr>
<td>2</td>
<td>Jonas Löwgren</td>
<td>Knowledge-Based Design Support and Discourse Management in User Interface Management Systems</td>
<td>1991</td>
<td>91-7870-720-X</td>
</tr>
<tr>
<td>3</td>
<td>Henrik Eriksson</td>
<td>Meta-Tool Support for Knowledge Acquisition</td>
<td>1991</td>
<td>91-7870-746-3</td>
</tr>
<tr>
<td>4</td>
<td>Peter Ekland</td>
<td>An Epistemic Approach to Interactive Design in Multiple Inheritance Hierarchies</td>
<td>1991</td>
<td>91-7870-784-6</td>
</tr>
<tr>
<td>5</td>
<td>Patrick Doherty</td>
<td>NML3 – A Non-Monotonic Formalism with Explicit Defaults</td>
<td>1991</td>
<td>91-7870-816-8</td>
</tr>
<tr>
<td>6</td>
<td>Nahid Shahmehri</td>
<td>Generalized Algorithmic Debugging</td>
<td>1991</td>
<td>91-7870-828-1</td>
</tr>
<tr>
<td>7</td>
<td>Nils Dahlbäck</td>
<td>Representation of Discourse-Cognitive and Computational Aspects</td>
<td>1992</td>
<td>91-7870-850-8</td>
</tr>
<tr>
<td>8</td>
<td>Ulf Nilsson</td>
<td>Abstract Interpretations and Abstract Machines: Contributions to a Methodology for the Implementation of Logic Programs</td>
<td>1992</td>
<td>91-7870-858-3</td>
</tr>
<tr>
<td>9</td>
<td>Ralph Rönquist</td>
<td>Theory and Practice of Tense-bound Object References</td>
<td>1992</td>
<td>91-7870-873-7</td>
</tr>
<tr>
<td>10</td>
<td>Björn Fjellborg</td>
<td>Pipeline Extraction for VLSI Data Path Synthesis</td>
<td>1992</td>
<td>91-7870-880-X</td>
</tr>
<tr>
<td>11</td>
<td>Staffan Bonnier</td>
<td>A Formal Basis for Horn Clause Logic with External Polymorphic Functions</td>
<td>1992</td>
<td>91-7870-896-6</td>
</tr>
<tr>
<td>12</td>
<td>Kristian Sandahl</td>
<td>Developing Knowledge Management Systems with an Active Expert Methodology</td>
<td>1992</td>
<td>91-7870-897-4</td>
</tr>
<tr>
<td>14</td>
<td>Mats Wären</td>
<td>Studies in Incremental Natural Language Analysis</td>
<td>1992</td>
<td>91-7871-027-8</td>
</tr>
<tr>
<td>15</td>
<td>Mariam Kamkar</td>
<td>Interprocedural Dynamic Slicing with Applications to Debugging and Testing</td>
<td>1993</td>
<td>91-7871-065-0</td>
</tr>
<tr>
<td>16</td>
<td>Tingting Zhang</td>
<td>A Study in Diagnosis Using Classification and Defaults</td>
<td>1993</td>
<td>91-7871-078-2</td>
</tr>
<tr>
<td>20</td>
<td>Ulf Söderman</td>
<td>Conceptual Modelling of Mode Switching Physical Systems</td>
<td>1995</td>
<td>91-7871-516-4</td>
</tr>
<tr>
<td>21</td>
<td>Andreas Kågedal</td>
<td>Exploiting Groundness in Logic Programs</td>
<td>1995</td>
<td>91-7871-538-5</td>
</tr>
<tr>
<td>22</td>
<td>George Fodor</td>
<td>Ontological Control, Description, Identification and Recovery from Problematic Control Situations</td>
<td>1995</td>
<td>91-7871-603-9</td>
</tr>
<tr>
<td>23</td>
<td>Mikael Petersson</td>
<td>Compiling Natural Semantics</td>
<td>1995</td>
<td>91-7871-641-1</td>
</tr>
</tbody>
</table>


