Personalization of User - Interface Agent Interaction

Silvia Noemí Schiaffino

Prof. Dra. Analía Amandi
Advisor

A thesis report submitted for the degree of
Doctor in Computer Science

Facultad de Ciencias Exactas
Departamento de Computación y Sistemas
Universidad Nacional del Centro de la Pcia. de Bs. As.
June 2004
Abstract

It has been demonstrated that users would rather interact with their interface agents like they interact with their human secretaries [Reeves and Nass, 1996]. Thus, they expect from interface agents the same qualities and behaviors they expect from a secretary, such as efficiency, politeness, cooperation, and proactiveness. Since different users have different expectations with respect to their agents, the interaction between users and agents has to be personalized in order to suit each user.

The goals of personalization within Interface Agent technology have been: identifying a users’ preferences, goals, and habits; building user profiles with this information; and providing personalized assistance to users with respect to their user profiles. However, interface agents have not deeply considered the personalization of user-agent interaction.

Through an experimental study, we have discovered a set of user-agent interaction issues that require personalization [Schiaffino and Amandi, 2004]. These issues are: the type of assistant each user wants (submissive, proactive); the type of assistance the user expects in different contexts (alerts, suggestions, actions on the user’s behalf); users’ preferences regarding interruptions; users’ tolerance to agents’ errors; how much control each user wants to delegate to agents; how much explicit feedback users are willing to provide. These issues as well as other human-computer etiquette rules can make the interaction between users and agent a success or a failure [Miller, 2004].

In this work, we propose an approach to personalize the interaction between users and agents that solves the following problems: assisting a user with the type of assistance he expects in different situations; and providing this assistance in the modality he prefers, that is interrupting or not interrupting him. Our proposal is three-fold, since we propose: the definition of a user interaction profile that contains a user’s assistance requirements and interruption preferences; the profiling algorithms needed to build the user interaction profile; and how to use this profile to personalize the interaction between interface agents and users.

Keywords: personalization, human-computer etiquette, interface agents, user profiling
Declaration

The work in this thesis is based on research carried out at ISISTAN Research Institute, Fac. Cs. Exactas, UNICEN.

The work was carried out with the support of CONICET, Comisión Nacional de Investigaciones Científicas y Técnicas, through a doctoral fellowship grant (from April 2000 to March 2004).
Acknowledgments

I would like to take this opportunity to thank all those who have contributed to this thesis, directly or indirectly.

I would first like to express my gratitude to my advisor Analía Amandi for her constructive criticism and guidance.

I owe my gratitude to the chair, Marcelo Campo, and to all the members of ISISTAN for making of this institute an appropriate environment for research.

I would like to thank CONICET for providing financial support for the work in this thesis.

Thanks to Marisa for her valuable comments on my English writing.

I also want to thank some undergraduate students who participated in the development and evaluation of software related with this thesis. Thanks to Patricio García, Miguel Armentano and Eduardo Oficialdegui.

I am deeply indebted to my parents for their support and sacrifices without which I certainly would not have reached where I have.

Finally, I would especially like to thank my boyfriend Diego for supporting, encouraging and loving me.

Silvia
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Declaration</td>
<td>iv</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>v</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Personalization and Interface Agent Technology</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Problem Overview</td>
<td>5</td>
</tr>
<tr>
<td>1.2.1 Users’ Assistance Requirements</td>
<td>7</td>
</tr>
<tr>
<td>1.2.2 User’s Interruption Preferences</td>
<td>8</td>
</tr>
<tr>
<td>1.3 Proposed Solution</td>
<td>9</td>
</tr>
<tr>
<td>1.3.1 User Interaction Profile</td>
<td>9</td>
</tr>
<tr>
<td>1.3.2 User-Agent Interaction Profiling Algorithm</td>
<td>10</td>
</tr>
<tr>
<td>1.4 Thesis Overview</td>
<td>12</td>
</tr>
<tr>
<td><strong>2 User- Interface Agent Interaction: Open Personalization Issues</strong></td>
<td>13</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>13</td>
</tr>
<tr>
<td>2.2 Personalization Issues: Empirical Study</td>
<td>15</td>
</tr>
<tr>
<td>2.2.1 Experiment Participants</td>
<td>15</td>
</tr>
<tr>
<td>2.2.2 Experiment Procedure</td>
<td>16</td>
</tr>
<tr>
<td>2.2.3 Experiment Results</td>
<td>16</td>
</tr>
<tr>
<td>2.2.3.1 Do users prefer submissive or authoritative agents?</td>
<td>16</td>
</tr>
<tr>
<td>2.2.3.2 Allowing actions on the user’s behalf</td>
<td>18</td>
</tr>
<tr>
<td>2.2.3.3 Does the user want suggestions or just warnings?</td>
<td>19</td>
</tr>
<tr>
<td>2.2.3.4 Do users tolerate agents’ errors?</td>
<td>21</td>
</tr>
<tr>
<td>2.2.3.5 Do users object to providing explicit feedback?</td>
<td>23</td>
</tr>
<tr>
<td>2.2.3.6 Do users accept agents’ interruptions?</td>
<td>23</td>
</tr>
<tr>
<td>2.3 Lessons Learned</td>
<td>24</td>
</tr>
<tr>
<td>2.4 Summary</td>
<td>26</td>
</tr>
</tbody>
</table>
3 Personalizing User-Agent Interaction

3.1 Introduction .................................................. 28
3.2 Proposed approach: Overview ............................... 32
3.3 Proposed user profile ......................................... 33
3.4 Building User Interaction Profiles ......................... 36
  3.4.1 Proposed Interaction Profiling Approach ............... 36
  3.4.2 Data Model ................................................ 37
  3.4.3 Proposed learning algorithms overview ................. 40
3.5 Detailed decision making algorithm ......................... 42
  3.5.1 Characteristics of the proposed algorithm ............. 45
    3.5.1.1 Domain dependence .................................. 45
    3.5.1.2 Threshold values ................................... 45
    3.5.1.3 Comparison of situations .......................... 46
    3.5.1.4 How to compute the confidence on an agent action 46
3.6 Summary ....................................................... 47

4 The WATSON Algorithm: Does the user want a Warning, an
  AcTion, a Suggestion or Nothing? ............................. 48
4.1 Introduction ..................................................... 48
4.2 WATSON Inputs and Outputs .................................. 49
4.3 WATSON Overview ............................................. 52
  4.3.1 Mining Association Rules from User-Agent Interaction Experiences ........................................... 54
  4.3.2 Analyzing the interestingness of rules .................... 57
  4.3.3 Pruning the discovered rules ............................. 59
  4.3.4 Building facts from hypotheses .......................... 62
4.4 Incremental Learning and Profiling .......................... 66
  4.4.1 Update of association rules .............................. 66
  4.4.2 When to update? ......................................... 69
4.5 Example ........................................................ 71
4.6 Summary ....................................................... 78

5 The IONWI algorithm: Does the user prefer an Interruption or a
  Notification Without Interruption? ............................. 79
5.1 Introduction ..................................................... 79
5.2 Interruptions: Concept and Categories ........................ 81
5.3 Working with interruptions: related research ............... 83
  5.3.1 Visual and Multimodal Strategies for Interruption .... 83
5.3.2 Interruption Length, Complexity and Relationship with the Primary User Task ........................................... 84
5.3.3 Temporal Strategies for Interruption .......................... 84
5.3.4 Cost and Utility of Interruptions ............................... 85

5.4 The IONWI learning algorithm .................................... 86
5.4.1 Algorithm Inputs and Outputs ................................ 87
5.4.2 Learning users’ interruption preferences with IONWI .... 90
5.4.3 Analyzing the interestingness of rules ..................... 91
5.4.4 Pruning the discovered rules ................................. 92
5.4.5 Building facts from hypotheses .............................. 94

5.5 Incremental Learning ................................................ 96
5.6 Assisting the user with IONWI .................................... 98
5.7 Example .................................................................. 102
5.8 Summary .................................................................. 106

6 Applications and Tools implemented ............................ 108
6.1 Introduction ............................................................ 108
6.2 Calendar Agent ...................................................... 109
6.2.1 Overview of the agent ........................................ 110
6.2.2 Learning user profiles in the calendar management domain .................................................. 111
6.2.3 Personalizing user-agent interaction .................... 113

6.3 User Profiling Toolkit ................................................. 115
6.3.1 Toolkit Overview ............................................... 115
6.3.2 Selecting input files ............................................ 116
6.3.3 Selecting and running a profiling algorithm ............. 117
6.3.4 Selecting and running a decision making algorithm ... 120
6.3.5 Visualizing a user profile .................................... 122

6.4 Design and Implementation ........................................ 123
6.4.1 Overview ........................................................ 123
6.4.2 The profile building components ......................... 125
6.4.3 Implementation details ....................................... 127

6.5 Summary .................................................................. 131

7 Determining WATSON and IONWI Parameters ................ 132
7.1 Introduction ............................................................ 132
7.2 Coefficients to determine negative and positive rules ....... 133
7.3 Threshold values used to discover rules with similar confidence .................................................. 133
7.4 Values for minsup and minconf ................................. 134
7.5 Parameters of the certainty function ........................................ 138
7.6 Number of rules to be generated by Apriori .......................... 138
7.7 Value of the certainty threshold $\delta$ ................................. 141
7.8 Number of instances in the dataset ................................ 141
7.9 Parameters for incremental learning ................................. 142
7.10 Minimum confidence values for different assistance actions .... 146
7.11 Similarity thresholds for comparing two situations ............... 147
7.12 Summary ................................................................. 147

8 Experimental Evaluation ......................................................... 148
8.1 Introduction ................................................................. 148
8.2 User Profiles and Datasets ............................................. 149
  8.2.1 User profiles and datasets for $WATSON$ ........................ 149
  8.2.2 User profiles and datasets for $IONWI$ .......................... 152
8.3 Evaluating $WATSON$ ability to learn a user’s assistance requirements 154
8.4 Comparing $WATSON$ against the confidence-based algorithm .... 156
8.5 Evaluating $IONWI$ ability to learn a user’s interruption preferences . 161
8.6 Conclusions ................................................................. 166

9 Conclusions .............................................................................. 167
9.1 Summary ................................................................. 167
9.2 Main Contributions .................................................... 168
9.3 Limitations ................................................................. 169
9.4 Future Work and Research ........................................... 170

Appendix ....................................................................................... 172

A Interface Agents .............................................................. 172

B Questionnaire ..................................................................... 178

C The Attribute-Relation File Format .................................... 181
  C.1 Overview ................................................................. 181
  C.2 The ARFF Header Section .......................................... 182
    C.2.1 The @relation Declaration .................................. 182
    C.2.2 The @attribute Declarations .............................. 182
  C.3 ARFF Data Section .................................................. 184
    C.3.1 The @data Declaration ..................................... 184
    C.3.2 The instance data ............................................ 184
## Contents

C.4 Full example on the ZOO dataset ............................................. 185

### D Association Rules

D.1 The Apriori Algorithm ...................................................... 189
D.2 Incremental updating of association rules .............................. 194
  D.2.1 The FUP Algorithm .................................................. 195
  D.2.2 The FUP2 Algorithm ................................................. 198
  D.2.3 The DELI Algorithm .................................................. 202
  D.2.4 Estimation of the difference between old and new association rules ................................................. 203
D.2.5 Drawing a random sample ............................................. 204
D.2.6 Estimating the symmetric difference ................................. 204

### E User Profiles Descriptions

E.1 User 1 ............................................................................. 208
  E.1.1 Type of User ............................................................. 208
  E.1.2 Preferences with respect to calendar management ................ 208
  E.1.3 User Assistance Requirements ....................................... 208
  E.1.4 User Interruption Preferences ....................................... 209
E.2 User 2 ............................................................................. 209
  E.2.1 Type of User ............................................................. 209
  E.2.2 Preferences with respect to calendar management ................ 209
  E.2.3 User Assistance Requirements ....................................... 210
E.3 User 3 ............................................................................. 210
  E.3.1 Type of User ............................................................. 210
  E.3.2 Preferences with respect to calendar management ................ 210
  E.3.3 User Assistance Requirements ....................................... 211
  E.3.4 User Interruption Preferences ....................................... 211
E.4 User 4 ............................................................................. 212
  E.4.1 Type of User ............................................................. 212
  E.4.2 Preferences with respect to calendar management ................ 212
  E.4.3 User Assistance Requirements ....................................... 212
  E.4.4 User Interruption Preferences ....................................... 212
E.5 User 5 ............................................................................. 213
  E.5.1 Type of User ............................................................. 213
  E.5.2 Preferences with respect to calendar management ................ 213
  E.5.3 User Assistance Requirements ....................................... 213
  E.5.4 User Interruption Preferences ....................................... 214
E.6 User 7 ......................................................... 215
    E.6.1 Type of User ......................................... 215
    E.6.2 Preferences with respect to calendar management ....... 215
    E.6.3 User Assistance Requirements .......................... 215
    E.6.4 User Interruption Preferences .......................... 215
E.7 User 8 ......................................................... 216
    E.7.1 Type of User ......................................... 216
    E.7.2 Preferences with respect to calendar management ....... 216
    E.7.3 User Assistance Requirements .......................... 216
E.8 User 10 ......................................................... 216
    E.8.1 Type of User ......................................... 216
    E.8.2 User Assistance Requirements .......................... 216
E.9 User 16 ......................................................... 217
    E.9.1 Type of User ......................................... 217
    E.9.2 Preferences with respect to calendar management ....... 217
    E.9.3 User Assistance Requirements .......................... 217
    E.9.4 User Interruption Preferences .......................... 218
E.10 User 28 ......................................................... 218
    E.10.1 Type of User ......................................... 218
    E.10.2 Preferences with respect to calendar management ....... 219
    E.10.3 User Assistance Requirements .......................... 219
    E.10.4 User Interruption Preferences .......................... 219
E.11 User 30 ......................................................... 219
    E.11.1 Type of User ......................................... 219
    E.11.2 User Assistance Requirements .......................... 220
E.12 User 31 ......................................................... 220
    E.12.1 Type of User ......................................... 220
    E.12.2 User Assistance Requirements .......................... 220

Bibliography .................................................. 222
## List of Figures

1.1 How interface agents work (Adapted from [Maes, 1994]) .......................... 3
1.2 User profile ....................................................................................... 10
1.3 Proposal overview .............................................................................. 11

2.1 Levels of expertise of the participants .............................................. 15
2.2 User-agent interaction styles .............................................................. 17
2.3 Preferences of the different types of users ........................................ 18
2.4 Assistance actions allowed by users .................................................. 18
2.5 Users' acceptance of agents' actions on their behalf ......................... 19
2.6 Preferences of the different types of users ........................................ 20
2.7 Ability to distinguish different assistance requirements .................... 21
2.8 Users' preferences regarding suggestions ....................................... 21
2.9 Users' tolerance to agents' errors ..................................................... 22
2.10 Users' tolerance to different types of errors ...................................... 22
2.11 Users' tolerance versus interaction stage ......................................... 22
2.12 Users' reactions towards providing explicit feedback ........................ 24
2.13 Users' reaction towards interruptions .............................................. 24

3.1 Proposed User Profile ........................................................................ 34
3.2 Proposed profiling approach ............................................................... 38
3.3 Proposed profiling algorithms ............................................................ 41

4.1 Learning and decision making algorithms ......................................... 49
4.2 Example input file for WATSON .......................................................... 52
4.3 WATSON Overview ............................................................................ 52
4.4 WATSON main steps ......................................................................... 53
4.5 Example of a taxonomy ..................................................................... 58
4.6 Formulating hypotheses from user-agent interaction experiences ...... 62
4.7 Definitions of $D$, $D'$, $D_-$, $\Delta_-$ and $\Delta_+$ ............................. 67
4.8 Input file “overlapping” ..................................................................... 72
4.9 Association rules generated by Apriori ............................................. 72
## List of Figures

4.10 New user-agent interactions ........................................... 76

5.1 Learning and decision making algorithms .......................... 80
5.2 IONWI Overview .......................................................... 87
5.3 IONWI main steps ......................................................... 88
5.4 Example input file for IONWI .......................................... 90
5.5 Task model for adding a meeting ...................................... 101
5.6 Relevance of an interruption to a task .............................. 102
5.7 Input file for IONWI ...................................................... 104
5.8 Association rules generated by Apriori ............................... 105

6.1 Calendar Agent ............................................................ 111
6.2 Calendar Agent overview ............................................... 112
6.3 The agent helps the user to schedule a new event ................. 114
6.4 User-interface for an interruption ..................................... 114
6.5 The agent is hidden in the tool-bar ................................... 114
6.6 Example input file for WATSON ....................................... 116
6.7 Pre-processing the profiling algorithm input ........................ 117
6.8 WATSON parameters .................................................... 118
6.9 Basic algorithm parameters ............................................. 119
6.10 Running a profiling algorithm ........................................ 119
6.11 Results of the basic algorithm ....................................... 120
6.12 Decision making .......................................................... 121
6.13 Profile visualization ...................................................... 122
6.14 Main components of our interface agents ........................... 123
6.15 Detailed view of the components .................................... 125
6.16 User interaction profile building components ..................... 126
6.17 Incremental profiling components .................................... 127
6.18 Profiling algorithms hierarchy ........................................ 128
6.19 Incremental algorithms hierarchy .................................... 129
6.20 Profile ................................................................. 130
6.21 Rule filtering ............................................................. 131
6.22 Decision making ........................................................ 131

7.1 \textit{minsup} values versus Number of hypotheses (WATSON - User 1) . 135
7.2 \textit{minsup} values versus Number of hypotheses (WATSON - User 2) . 135
7.3 \textit{minsup} values versus Number of hypotheses (WATSON - User 3) . 135
7.4 \textit{minsup} values versus Number of hypotheses (WATSON - User 4) . 136
7.5 \textit{minsup} values versus Number of hypotheses (WATSON - User 5) . 136
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6</td>
<td>\textit{mins}up values versus Number of hypotheses (\textit{IONWI} - User 3)</td>
<td>137</td>
</tr>
<tr>
<td>7.7</td>
<td>\textit{mins}up values versus Number of hypotheses (\textit{IONWI} - User 4)</td>
<td>137</td>
</tr>
<tr>
<td>7.8</td>
<td>\textit{mins}up values versus Number of hypotheses (\textit{IONWI} - User 7)</td>
<td>138</td>
</tr>
<tr>
<td>7.9</td>
<td>Number of instances versus Number of hypotheses (\textit{WATSON})</td>
<td>141</td>
</tr>
<tr>
<td>7.10</td>
<td>Number of instances versus Number of hypotheses (\textit{IONWI})</td>
<td>142</td>
</tr>
<tr>
<td>8.1</td>
<td>\textit{WATSON} precision (holiday dataset)</td>
<td>155</td>
</tr>
<tr>
<td>8.2</td>
<td>\textit{WATSON} recall (holiday dataset)</td>
<td>155</td>
</tr>
<tr>
<td>8.3</td>
<td>\textit{WATSON} precision (overlapping dataset)</td>
<td>156</td>
</tr>
<tr>
<td>8.4</td>
<td>\textit{WATSON} recall (overlapping dataset)</td>
<td>156</td>
</tr>
<tr>
<td>8.5</td>
<td>\textit{WATSON} precision (new event dataset)</td>
<td>157</td>
</tr>
<tr>
<td>8.6</td>
<td>\textit{WATSON} recall (new event dataset)</td>
<td>157</td>
</tr>
<tr>
<td>8.7</td>
<td>\textit{WATSON} precision (time dataset)</td>
<td>157</td>
</tr>
<tr>
<td>8.8</td>
<td>\textit{WATSON} recall (time dataset)</td>
<td>158</td>
</tr>
<tr>
<td>8.9</td>
<td>Average \textit{WATSON} precision</td>
<td>158</td>
</tr>
<tr>
<td>8.10</td>
<td>Average \textit{WATSON} recall</td>
<td>158</td>
</tr>
<tr>
<td>8.11</td>
<td>\textit{WATSON} versus Confidence-Based Algorithm (holiday dataset)</td>
<td>159</td>
</tr>
<tr>
<td>8.12</td>
<td>\textit{WATSON} versus Confidence-Based Algorithm (new event dataset)</td>
<td>160</td>
</tr>
<tr>
<td>8.13</td>
<td>\textit{WATSON} versus Confidence-Based Algorithm (overlapping dataset)</td>
<td>160</td>
</tr>
<tr>
<td>8.14</td>
<td>\textit{WATSON} versus Confidence-Based Algorithm (time dataset)</td>
<td>160</td>
</tr>
<tr>
<td>8.15</td>
<td>\textit{IONWI} precision (new event dataset)</td>
<td>162</td>
</tr>
<tr>
<td>8.16</td>
<td>\textit{IONWI} recall (new event dataset)</td>
<td>162</td>
</tr>
<tr>
<td>8.17</td>
<td>\textit{IONWI} precision (holiday dataset)</td>
<td>163</td>
</tr>
<tr>
<td>8.18</td>
<td>\textit{IONWI} recall (holiday dataset)</td>
<td>163</td>
</tr>
<tr>
<td>8.19</td>
<td>\textit{IONWI} precision (overlapping dataset)</td>
<td>163</td>
</tr>
<tr>
<td>8.20</td>
<td>\textit{IONWI} recall (overlapping dataset)</td>
<td>164</td>
</tr>
<tr>
<td>8.21</td>
<td>\textit{IONWI} precision (time dataset)</td>
<td>164</td>
</tr>
<tr>
<td>8.22</td>
<td>\textit{IONWI} recall (time dataset)</td>
<td>164</td>
</tr>
<tr>
<td>8.23</td>
<td>Average \textit{IONWI} precision</td>
<td>165</td>
</tr>
<tr>
<td>8.24</td>
<td>Average \textit{IONWI} recall</td>
<td>165</td>
</tr>
<tr>
<td>A.1</td>
<td>How interface agents work (Adapted from [Maes, 1994])</td>
<td>174</td>
</tr>
<tr>
<td>D.1</td>
<td>Discovering large itemsets using the Apriori algorithm</td>
<td>194</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Definitions of several symbols [Cheung et al., 1997] .......................... 68
5.1 Relationships between user tasks and agent actions .......................... 100
7.1 Number of rules versus Number of hypotheses (WATSON) ................... 139
7.2 Number of rules versus Number of Hypotheses (IONWI) ..................... 140
7.3 Ratio values for different sizes of the increment database (WATSON) 144
7.4 Ratio values for different sizes of the increment database (IONWI) ........ 145
7.5 Ratio values for different sizes of the increment database (IONWI) ....... 146
8.1 Characteristics of the datasets for WATSON ................................. 151
8.2 Characteristics of the datasets for IONWI .................................................. 153
A.1 Examples of interface agents ............................................................... 176
D.1 Transaction Database ........................................................................ 190
D.2 Support for Itemsets in Table C.1 and Large Itemsets with support
    of 40% ..................................................................................................... 190
D.3 Confidence of some association rules where minconf is 60% .......... 190
D.4 Finding candidates sets using apriori-gen ........................................... 192
List of Algorithms

1  High level decision making algorithm ............................ 30
2  Our high level decision making algorithm ........................ 33
3  WATSON and IONWI Overview ..................................... 43
4  Detailed decision making algorithm ............................... 44
5  WATSON Overview ................................................... 54
6  WATSON learning algorithm ........................................ 65
7  Incremental WATSON .................................................. 70
8  IONWI Overview ....................................................... 91
9  IONWI learning algorithm .......................................... 96
10  Incremental IONWI .................................................... 97
11  Decision making with IONWI ....................................... 99
12  Determining the relationship between interruptions and primary user
tasks ................................................................. 103
13  Basic Algorithm ....................................................... 189
14  Find association rules given large itemsets ....................... 191
15  apriori-gen function .................................................. 192
16  Function count ......................................................... 193
17  Apriori algorithm ...................................................... 193
18  FUP - First Iteration .................................................. 197
19  FUP - The k-th Iteration ............................................... 199
Chapter 1

Introduction

“When I hit my thumb with a hammer, I get mad at myself, at the person making me do the work, and at the cruel and malignant universe, but I do not get mad at the hammer. By contrast, when my computer crashes, or fails to give me a file that I know is there, or tells me that I failed to shut it down properly when it was the one that crashed, I get mad at it. True, if I stop to think about it, I may get mad at the people who designed, programmed, or last updated it, but my immediate reaction is more personal. Even my language, as I write this, is illustrative: I hit myself with the hammer, while my computer does thing to me [Miller, 2004].”

1.1 Personalization and Interface Agent Technology

Personalization in software applications constitutes nowadays a valuable technology. Personalization is the process of adapting a computer application to the needs of specific users, taking advantage of the knowledge acquired from the analysis of the user’s behavior and user data. As users become more proficient in their use of computer applications, and they are exposed to a wider range of experiences, they may well become more demanding, and their definition of what constitutes a good service or a good application may be refined [Bonett, 2001]. User satisfaction is the ultimate aim of personalization. It is motivated by the recognition that a user has needs and preferences, and meeting them successfully is likely to lead to a satisfying relationship with him.

At this point, a distinction should be made between customization and personalization. The difference between them lies on the control of the adaptation process [Treiblmaier et al., 2004; Nielsen, 1998]. Customization is a user-initiated and user-driven process. Customization occurs when the user configures an application manually, adding and removing elements in his configuration according to his needs and preferences. It uses adaptable system components which users can tailor to their specific needs. These systems use static profiles that can be changed by
1.1. Personalization and Interface Agent Technology

the user, such as the Web Portal Daily Routine\(^1\), which enables users to adapt the content and layout to their preferences.

By contrast, personalization is system-initiated, system-driven and requires adaptive components. In order to make modifications appropriate to the needs of the individual user, both approaches require detailed information about the user. Personalization, however, also requires the system to monitor user behavior in order to adapt automatically, and users are thus unable to control how the system adjusts to their behavior.

There is great diversity in how personalization can be achieved. There are various related areas addressing personalization with different approaches, such as user modeling systems [Kay, 1995, Kobsa and Pohl, 1995], intelligent agents (particularly interface agents) [Maes, 1994, Lieberman, 1995], intelligent user interfaces [Maybury, 1999, Ross, 2000], mixed-initiative interaction systems [Horvitz, 1999a], adaptive systems [Brusilovsky and Maybury, 2002], and attentive user interfaces [McCrickard and Chewar, 2003]. In turn, within these areas, features classified as personalization are wide-ranging from simple display of the end-user's name on a Web page to complex catalog navigation based on deep models of users' needs and behaviors. Similarly, personalization technologies range from commonplace use of databases, cookies and dynamic page generation, to pattern matching and machine learning algorithms, rule-based inferencing and data mining [Kramer et al., 2000].

From the approaches mentioned, our work is framed within interface agent technology. Interface agents are computer programs designed to assist human users with their computer-based tasks in a personalized manner. Interface agents are capable of learning a user’s interests, preferences, priorities, goals and needs, and providing him proactive and reactive assistance in order to increase the user's productivity with respect to a computer application. Interface agents can help a user by suggesting courses of action, by performing tasks on his behalf, and by alerting him about relevant situations, for example. A commonly used metaphor to understand interface agent paradigm is comparing them to a human secretary or personal assistant who is collaborating with the user in the same work environment [Maes, 1994]. Initially, a personal assistant is not very familiar with the habits and preferences of her\(^2\) employer and may not be very helpful. The assistant needs some time to become familiar with the particular working habits of her boss. The assistant becomes gradually more effective and competent as she acquires knowledge about him. This

\(^1\)http://www.dailyroutine.com

\(^2\)For simplicity, we use “she” for the assistant and “he” for the boss and the user, but we do not mean to be sexist.
1.1. Personalization and Interface Agent Technology

Figure 1.1: How interface agents work (Adapted from [Maes, 1994])

knowledge is acquired by observing how the employer performs tasks, by receiving explicit instructions and feedback from the employer, by asking him for information or by learning from other assistants’ experience. Then, the assistant can carry out activities that were initially done by the employer, she can suggest him the execution of tasks, she can notify the employer about interesting situations and warn him about problems that may arise. In the same way, an interface agent can become more competent as it interacts with a user and learns about him. Figure 1.1 depicts interface agent’s functionality.

Interface agents have been used in numerous application domains, mainly in those related to information overload management and to the execution of repetitive tasks, such as information filtering [Billsus and Pazzani, 1999], information searching and information retrieval [Lieberman et al., 2001], email handling [Segal and Kephart, 1999], meeting scheduling [Payne et al., 2002], and e-commerce [Morris et al., 2000].

As intelligent agents take on more complexity, higher degrees of autonomy and more “intelligence”, users start to expect them to play by the same rules of other complex, autonomous and intelligent entities in their experience - namely, other people [Miller, 2002a]. As agents become more complex, adaptive and autonomous, the importance for them to show appropriate behavior increases, and conversely, the sensitivity of users to inappropriate behavior will increase [Whitworth, 2002].

In this context, Horvitz [Horvitz, 1999a] pointed out some problems with the use of interface agents: poor guessing about the goals and needs of users, inadequate consideration of the costs and benefits of each agent action, poor timing of agent actions, and inadequate attention to opportunities that allow a user to guide the invocation of agent services and to refine potentially suboptimal results provided by the agent.

3 The reader can find more information about interface agents in Appendix A.
1.1. Personalization and Interface Agent Technology

As regards these problems, particularly the first one, we consider that interface agent developers have paid little attention to a major issue when assisting a user: how to personalize the interaction between interface agents and users. The interface agents developed thus far have concentrated on obtaining a user’s preferences and working habits with respect to a given computer application, but they have not learned how to properly interact with him and how to provide him assistance without hindering his work. Little research has been done towards personalizing the interaction with users.

Recalling the assistant metaphor, each person works and interacts in a certain way with his or her personal assistant. It has been demonstrated that users apply the schemas learned for interacting with humans to other agents that behave in some minimal ways like humans [Reeves and Nass, 1996]. Consequently, we can say that each user interacts in a personal way with his interface agent and he surely expects from the agent the same behavior he expects from his human assistant. In turn, the kind of assistant a user wants, the actions a user expects from his agent, the agent’s errors he tolerates and the type of assistance a user requires in a given context will vary from one user to another. For example, a user would probably not object to being interrupted by an agent with notifications or suggestions, provided that he accepts this from a human assistant. A different user would probably dismiss an assistant of such a kind, and thus, he would never tolerate such a behavior from an interface agent.

The consequences of not fulfilling users’ expectations can be negative for an interface agent. Some experiments [Ruvini and Gabriel, 2002] have demonstrated that users tolerate some agents’ errors as far as they achieve an acceptable performance, and they are less tolerant of agents that are not competent. When an agent makes mistakes – very likely in early, learning stages – users may place less trust in the agent. In an extreme case, users may adopt a strategy of completely ignoring or disabling the agent [Tieman et al., 2001]. For example, a disappointed user may limit the agent’s capabilities to a minimum, allowing it to act only upon his request. He can also diminish his interaction with the agent, ignoring its actions although some of them might be useful. An annoyed user because of its agent’s behavior (for example, one that always interrupts the user with irrelevant information) might prefer working without the agent and, thus, abandon it.

An example of negative consequences for software assistants is the decision of Microsoft product managers to "kill" the Microsoft Office Assistant, named Clippit, because it has been the source of wide scorn among developers. Microsoft listened to users’ complaints and made Clippit easier to hide in Office 2000. But the complaints
kept coming and now Clippit does not even show up by default in Office XP\(^4\).

In summary, personalized assistance and personalized interaction are necessary and desirable features in interface agents, since they give agents the ability to assist users as they expect. The more user-agent interaction aspects interface agents can personalize, the more successful they will be in their interaction with users. Consequently, they will avoid user rejection.

In this work we consider personalization from the point of view of the interaction between the user and the agent. Our personalization approach aims at improving the interaction between the user and the agent so that the user can count on a mindful assistant and a collaborator that knows not only his interests but also when and how he wants to be assisted.

The rest of this chapter is organized as follows. Section 1.2 describes in detail the problems we are tackling in this work. Section 1.3 presents our solution to these problems. Section 1.4 describes the organization of this thesis report.

1.2 Problem Overview

If an interface agent is to offer help at the right time and of the right sort, then it must learn not only which the user preferences and habits concerning a certain computer application are, but also how the user usually works and interacts with his agent, and what kind of assistant the user wants.

Some works have been carried out towards this direction. In [Horvitz et al., 1998], the authors discuss probabilistic approaches for determining what the users are doing and for using this information in deciding whether to assist them and how to do it. This goal is shared by those agents based on plan recognition, such as [Rich et al., 2001]. Plan recognition is the identification of a user’s plans or goals from the available evidence, which comprises his actions as well as information about his preferences [Carberry, 1990]. A prerequisite for the recognition of plans is knowledge of a user’s possible actions and the combination of these actions in complex action sequences, which describe typical user behavior. Although plan recognition enables an agent to reason about what the user might do next and to determine how to assist him, it does not provide information about how he wants to interact with the agent and how he wants the assistance to be provided.

1.2. Problem Overview

Regarding user-agent interaction, in [André et al., 1999] the personality of an interface agent is considered. The personality of an agent is defined by several features known as the OCEAN model [McCrae and Costa, 1996]: openness, conscientiousness, extroversion, agreeableness and neuroticism. An agent behaves according to a particular combination of these features. For example, as regards extroversion, an agent can be extrovert, neutral or introvert. Regarding agreeableness, an agent can be agreeable, neutral or disagreeable. Each user can select the personality he wants for his interface agent. The audiovisual appearance of interface agents has also been taken into account in [André and Rist, 2002], where the goal is representing interface agents with lifelike characters since this seems to be a more natural way of interacting with users. Each user can choose a different character as a representation for his agent, such as a parrot, a genie or a wizard.

Although the previous issues are important to improve and personalize the interaction between a user and his agent, there are other major issues that have to be considered. Empirical studies [Schiaffino and Amandi, 2004] showed us that each user interacts with his interface agent in a distinct and personal way, in the same way as he interacts in a distinct and personal way with a human secretary. These studies enabled us to discover a set of user-agent interaction issues that need to be personalized to assist users as they expect. These interaction issues are: considering the particular assistance requirements users have in different contexts; discovering when to interrupt the user; analyzing users’ tolerance of agents’ errors; discovering what type of assistant each user wants (for example, submissive, autonomous); providing the means for simple but useful explicit user feedback, and to capture as much implicit feedback as possible; providing the means to control and inspect agent behavior; and discovering how much control the user wants to delegate to the agent.

As our studies have demonstrated, these user-agent interaction issues have an impact on the competence of an interface agent and they can make the interaction between user and agent a success or a failure. This is the concern of a recent research area within Human-Computer Interaction that studies the “etiquette” of human-computer relationships [Miller, 2004, Nass, 2004, Bickmore, 2004]. We agree with the researchers in this area in that the ability to adapt to the way in which the user wants to interact with the agent is almost as important as the ability to learn the user’s preferences in a particular domain.

In this work, we will address two of the personalization problems mentioned above - which we considered were the most urgent to solve: learning how to best assist the user, that is discovering what type of assistance the user expects in a
1.2. Problem Overview
given context; and learning when (and if) to interrupt the user. The agent has to
learn when to offer its services to the user, how to best help the user, when to let
the user decide how to solve a problem and when to ask the user for information.
Given a problem situation or situation of interest for the user, the agent has to
determine which of the various possible assistance actions it should execute, and if
it is advisable to execute it or not. For every problem or interesting situation that
may arise in a certain application, an interface agent must learn when to warn the
user about it, when to propose a solution to the problem, and when to solve the
problem on behalf of the user. The agent has to discover which situations or items
in a given application domain are relevant to the user, which are urgent and which
are irrelevant. The agent should only interrupt the user if the situation is urgent
and relevant to him. The following subsections describe in detail the two problems
we are tackling.

1.2.1 Users’ Assistance Requirements

When an interface agent detects a situation in which the user might need assistance,
it has to decide whether to help him in that particular situation, and if he does, how
to do it. The agent has to choose among various possible assistance actions: warning\(^5\)
the user about the situation and let him decide how to deal with it; suggesting the
user how to handle the situation; executing an action to manage the situation on
the user’s behalf; or doing nothing.

The agent’s decision will be mainly influenced by the knowledge the agent has to
make a suggestion or to execute an autonomous action, since if it does not know what
to suggest or what to do, it will merely warn the user about the problem. However,
although the agent probably knows what to suggest or what to do, it has to learn
if the user wants it to suggest him something or not in that particular situation, or
if he does not even want to be warned. In order to take the best decision the agent
must learn how exactly the user wants to be assisted in a given context. The agent
has to analyze each feature describing a particular (problem) situation, as well as
the user’s needs and goals, since the same problem occurring in different contexts
may require a different agent action.

Consider, for example, an interface agent that assists a user with his calendar
management. Suppose that the user receives an email inviting him to take part in
an event, and that this event takes place in a location the user does not like. The

\(^5\)By warning we mean not only telling the user about a possible problem, but also notifying him
about an interesting situation. We include alerts and notifications in the definition of a warning.
1.2. Problem Overview

agent can warn the user about this situation; it can also suggest the user how to solve it, that is suggest an alternative place; it can reject the invitation on behalf of the user; or it can ignore the problem. However, the user does not always handle this kind of situations in the same way: he wants the agent’s assistance in some contexts (for example, business meetings), but he wants to handle some other problems by himself (for example, family parties). Thus, an interface agent should learn how the user wants to be assisted in order to personalize its interaction with him and not to hinder the user’s work with unwanted help.

1.2.2 User’s Interruption Preferences

As pointed out in [McCrickard and Chewar, 2003], one of the problems of agents is their incorrect estimates of the user’s task priorities, which makes information to be introduced at inappropriate times and with unsuitable presentation choices. McFarlane [McFarlane, 1997] defines an interruption as “methods by which a person shifts his or her focus of consciousness from one processing stream to another”. Research has found that interruptions are harmful. They are disruptive to the primary computing task and they decrease user’s performance. However, interruptions are necessary in interface agent technology since agents need to communicate important and urgent information to users.

To solve this problem, when the agent detects a (problem) situation relevant to the user it has to correctly decide if it will send him a notification without interrupting the user’s work, or if it will interrupt him. On the one hand, the user can choose between paying attention to a notification or not, and he can continue to work in the latter case. On the other hand, he is forced to pay attention to what the agent wants to tell him if it interrupts him abruptly.

Not to disturb the user, the agent has to base its decision on: the relevance and the urgency the situation has for the user; the relationship between the situation to be notified or the assistance to be provided and the user’s goals; the relevance the situation underlying the interruption has to the current user tasks; how tolerant the user is of interruptions; and when he does not want to be interrupted no matter how important the message is.

For example, consider an interface agent that assists a user with his calendar management. If a notification about an urgent meeting organized by the user’s boss is relevant to the user, then an interruption notifying this event is worthy, no matter which the primary user task is. On the other hand, a notification or reminder about a party a work-mate is organizing because of his birthday can be irrelevant to the user, even if he is scheduling a new event for that day. Thus, an interruption will
not be welcome. The interface agent will have to learn which situations are relevant and which are irrelevant so that no unwanted interruptions occur.

The following section briefly describes our proposed solution to tackle the two problems presented before.

1.3 Proposed Solution

In this work we propose to personalize not only the assistance provided to users but also the interaction between interface agents and users. Our approach proposes a new definition for a user profile, since our interface agents have to store information about the user’s assistance requirements and interruption preferences. The contents of a user profile in our approach are described in Section 1.3.1. Then, Section 1.3.2 presents our proposed profiling algorithm, which is used to obtain the new components of the user profile.

1.3.1 User Interaction Profile

In order to provide personalized assistance to computer users, interface agents need knowledge about them. This knowledge is contained in the user profile. A profile is a description of someone containing the most important or interesting facts about him or her. In our context, a user profile contains all the information an interface agent needs to provide personalized assistance to a user. A user profile contains both application dependent and application independent information about a user. Application independent information includes mainly personal information about the user, such as name, age, job, and hobbies. Application dependent information includes a user’s interests, preferences, goals, working habits, behavioral patterns, knowledge, needs, priorities and commitments regarding a particular application. Its contents vary from one application domain to another. For example, in an online newspaper domain, the user profile contains the types of news (topics) the user likes to read, the types of news (topics) the user does not like to read, the newspapers he usually reads, and the user’s reading habits and patterns. In a calendar management domain the user profile contains information about the dates and times when the user usually schedules each type of activity in which he is involved, the priorities each activity feature has for the user, the relevance of each user contact and the user’s scheduling and rescheduling habits.

Although the information considered thus far in a user profile seems to be enough to provide personalized assistance to users, information about user-agent interaction
Figure 1.2: User profile

is not taken into account. Thus, we will add to the user profile information about the user’s interaction habits, about his assistance requirements and about his reactions towards some agent assistance actions such as warnings, suggestions, actions on the user’s behalf, and interruptions.

Figure 1.2 shows the different components of a user profile in our approach. The components at the top of the figure represent a generalization of those considered by the different interface agents developed thus far. We have represented these components in a general form through a set of preferences with associated confidence values, the relations among some of these preferences and the exceptions that may arise to some of them. We have also modeled temporary preferences and preferences that show some kind of seasonality.

The components regarding user-agent interaction consist of a set of profile items representing the situations when the user requires a suggestion to deal with a problem, the situations when the user needs only a warning about a problem, the situations in which the user expects an agent action on his behalf, the circumstances under which the user accepts an interruption from the agent, and the situations in which the user needs a notification and does not want an interruption.

1.3.2 User-Agent Interaction Profiling Algorithm

In our approach, an interface agent observes the user’s behavior and the interaction between the user and the agent to obtain information about a user’s assistance requirements and interruption preferences. The agent records each interaction between the user and the agent, and the feedback (implicit or explicit) the user provides
after the agent’s assistance actions. We refer to each interaction as an assistance experience or interaction experience. An interaction experience is described: by the (problem) situation originating the interaction; the assistance action the agent executed to deal with the situation; the modality of the assistance, that is whether the agent interrupted the user or not; the task the user was carrying out; the relevance the situation originating the interaction has for this task; the user feedback obtained after assisting the user; an evaluation of the assistance experience (success or failure); and the date when the interaction took place.

All this information is then used by our user-agent interaction profiling algorithm to build the user interaction profile. An interface agent can use this profile to decide which assistance action to execute to assist the user in a given problem situation, and when and how (that is, interrupting or not interrupting the user) to provide this assistance.

Our profiling approach takes as input the set of user-agent interaction experiences and two algorithms process them. The first algorithm, named WATSON (for Warning, AcTion, Suggestion or No action), is in charge of learning when the user requires a warning, when he needs an action on his behalf, when he requires a suggestion and when he wants no assistance at all. The goal of the second algorithm, named IONWI (for Interruption Or Notification Without Interruption), is discovering when the agent can interrupt the user’s work to provide him assistance and when it can only send him a notification without interrupting him. The output of the algorithm are the components of the user interaction profile. Figure 1.3 presents an overview of our proposed approach.

Figure 1.3: Proposal overview
1.4 Thesis Overview

The rest of this thesis report is organized as follows. Chapter 2 describes some problems and open issues in personalization within interface agent technology. We describe an experiment we carried out to study these personalization issues regarding user - interface agent interaction. We present the results from this experiment and we discuss the lessons we have learned.

Chapter 3 presents an overview of our proposed solution to two of the personalization problems discussed in Chapter 2. Our proposal involves a new definition for a user profile together with two profiling algorithms that build this profile.

Chapter 4 presents the WATSON algorithm, which enables an interface agent to learn the assistance action a user requires in a given problem situation: warning, action, suggestion or no assistance at all. The output of this algorithm is a set of user assistance requirements, which are part of the user interaction profile.

Chapter 5 describes the IONWI algorithm, which enables an interface agent to determine whether it should interrupt the user’s work to tell him something or not in a particular situation. The output of this algorithm is a set of user interruption requirements, which are part of the user interaction profile.

Chapter 6 describes the design and implementation of our user interaction profiling approach. It also describes two applications we have developed to evaluate and further study our proposed approach.

Chapter 7 describes the experiments we carried out to set the different parameters of the WATSON and IONWI algorithms.

Chapter 8 presents the results of our experimental evaluation with the WATSON and the IONWI algorithms.

Finally Chapter 9 presents our conclusions, which describe the contributions of our work, the limitations we found and some future work.

Appendix A describes the main characteristics of interface agents. Readers who are not familiarized with this technology will also find a survey of the different domains in which these agents have been applied and how they provide personalized assistance to users.

Appendix B contains the questionnaire we used to interview users in the study presented in Chapter 2.

Appendix C describes the format of the input files for WATSON and IONWI.

Appendix D describes in detail some concepts on association rule mining, a technique our profiling algorithm uses.

Appendix E shows the descriptions of the user profiles we used to make the experiments with IONWI and WATSON.
Chapter 2

User- Interface Agent Interaction: Open Personalization Issues

We have empirically studied a set of user-agent interaction issues we presumed might require personalization [Schiaffino and Amandi, 2004]. In this Chapter we discuss the results we have obtained. Some of the personalization issues we have studied are: discovering the type of assistant a user wants, learning when (and if) to interrupt the user, discovering how the user wants to be assisted in different contexts. An interface agent should take these issues into account in order to personalize its interaction with a user and meet his assistance requirements.

2.1 Introduction

We consider that interface agent developers have paid little attention to two key issues when developing these agents: how to best interact with each user and how to provide them assistance of the right sort at the right time. We have focused our attention on studying the interaction between a user and an interface agent, as well as on the way in which a user wants to be assisted.

It has been demonstrated that users tend to interact with intelligent systems in the same ways as they interact with humans [Reeves and Nass, 1996]. The way in which users interact with other humans - human assistants for example - is potentially different for each user. Thus, each user interacts in a personal way (different from the way of other users) with an interface agent. To prove our hypothesis, we carried out an experiment with real users interacting with different interface agents, and we discovered several user-agent interaction issues that require personalization. The goals of our experiment were the ones listed below.

- First, we studied what kind of software assistant different users want. Given
that different people require different types of human secretaries, it is likely
that they want different types of interface agents. For example, some users
might prefer a collaborator but others might want a submissive agent that
only executes their orders. Thus, one of the goals of our study was discovering
if these differences in users’ preferences appear when users interact and work
with interface agents.

- Second, we studied users’ preferences and assistance requirements concerning
  three assistance actions: warnings, suggestions and actions on the user’s be-
  half. The purpose was studying the assistance actions users expect in different
  situations or contexts. We analyzed whether users require different assistance
  actions to deal with a given situation in different contexts, and which these
  situation-action pairs are. Particularly, we studied the differences among the
  preferences and requirements of various users.

- Third, we analyzed users’ reactions towards interruptions, that is whether
  users object to being interrupted by their agents or not. We studied how dif-
  ferent users react to interruptions in different contexts (relevant and irrelevant
  situations) and how users behave when agents interrupt their work.

- Fourth, we considered users’ tolerance of agents’ errors, since some users are
  more tolerant than others. We studied users’ tolerance to errors involving
  warnings, suggestions and actions. We analyzed variations in users’ tolerance
  depending on the type of error, on the assistance actions involved and on the
  “stage” in user-agent interaction\textsuperscript{1}. We focused our attention on the differences
  that exist among different users’ reactions to agents’ errors.

- Fifth, we investigated how users feel about providing explicit feedback to train
  their agents. Not all users are willing to give explicit feedback, and their will-
  ingness may vary as they interact with the agent. We studied users’ reactions
  towards providing feedback in different stages of the user-agent interaction and
  to different types of feedback (simple versus complicated).

- Finally, we analyzed users’ opinions regarding agents’ ability to perform ac-
  tions on behalf of users and the possibility of giving agents more autonomy.
  Our goal was to discover if human beings are ready to delegate more tasks to
  interface agents or if they still want to be in complete control.

\textsuperscript{1}By stage we mean whether the user and agent have interacted for a short or a long period. We
can talk about the beginning of an interaction, for example.
2.2. Personalization Issues: Empirical Study

![Pie chart showing the levels of expertise of the participants]

Figure 2.1: Levels of expertise of the participants

As a result of our experiment we found that different users have different preferences and requirements regarding the interaction issues mentioned above. This is an indication that a personalized assistance is required if interface agents want to assist users as they expect.

The rest of this chapter is organized as follows. Section 2.2 describes in detail the experiment we carried out. Section 2.3 summarizes the lessons we have learned.

2.2 Personalization Issues: Empirical Study

2.2.1 Experiment Participants

Forty-two users ranging in age from 21 to 50 years old, twenty-nine male and thirteen female, participated in the experiment. Some participants had developed and actively used interface agents in several application domains. Other participants had occasionally used some interface agents in various domains, and others had only interacted with those assistants that we can find in MS Word or MS Excel. We categorized the first group of participants as expert users (12%), the second group as intermediate users (64%), and the last group of participants as inexpert users (24%) regarding their experience at interacting with interface agents. Figure 2.1 shows the expertise levels of the participants.

As we have said before, expert participants had developed interface agents in various application domains, which makes them highly experienced regarding the experiment context. Intermediate users and most of the expert users had interacted with a subset of these interface agents, namely: an agent assisting users with calendar management [Schiavino and Amandi, 2002], an agent assisting users in Web searching [Godoy and Amandi, 2000], an agent that generates personalized newspapers [Cordero et al., 1999], and an agent that assists database users [Schiavino and Amandi, 2003]. These agents can provide different types of assistance, such as warning or alerting the user about relevant or interesting situations, making suggestions and automating some activities. The way in which the assistance is provided
2.2. Personalization Issues: Empirical Study

varies from one agent to another. For example, some agents notify the user about a suggestion they can make to the user and then, upon user request, they show the suggestion to the user. Other agents show the suggestions directly, without any
previous notification. This variety of assistance actions and user interfaces allowed users to have a big picture about how interface agents can assist them and how they can interact and work with them.

2.2.2 Experiment Procedure

All participants were given a survey with a set of questions regarding user-agent interaction and user assistance. The users were asked to answer the questionnaire providing as much information as possible. Those users who had interacted with interface agents developed within our research group answered the questions in terms of their personal experience with these agents. Those users who had only had interaction with MS Office assistants answered the questions according not only to their experience with these assistants, but also to their expectations about an interface agent’s behavior in those cases where they had no practical experience. We consider that both kinds of answers are equally important and valuable for our study since we want to know how the different users feel about interacting with software assistants.

2.2.3 Experiment Results

The following sections describe in detail the results we have obtained from our experiment. Each section addresses a particular issue regarding personalization in user-agent interaction, or a combination of different issues.

2.2.3.1 Do users prefer submissive or authoritative agents?

People having personal assistants work and interact with them in certain ways. For example, some people delegate most of their tasks to their assistants and trust them completely. Others probably do not like to delegate their tasks to their secretaries and they only let them do what they ask them to do. Therefore, it is natural to think that the same occurs with interface agents: different users are likely to prefer different types of software assistants. The type of relationship between a user and an interface agent can lead to different user-agent interaction styles, and probably can originate different user stereotypes, such as authoritative or dominant user (one who wants a submissive agent), interactive user (one who wants an interactive and collaborative agent), delegator (a user who delegates everything to his agent), among
2.2. Personalization Issues: Empirical Study

![Pie chart showing user-agent interaction styles](image)

Figure 2.2: User-agent interaction styles

others. An interface agent must discover which type of assistant the user wants as a first step towards personalization.

Our experiment revealed that 12% of the users prefer an agent who can collaborate and actively interact with them. The 40% of the users expect their agents to live "in the background", becoming active only when the user requests the agent’s assistance. The 48% of the users want their agents to take part only in certain situations, probably pre-defined by users, in which they would interact and collaborate with their agents. Figure 2.2 shows the preferences of the different users.

As regards the different types of users, we found that 20% of the expert users want a submissive agent, 40% of this type of users want a collaborator, and 40% prefer an agent that can act autonomously only in certain occasions. Concerning intermediate users, we discovered that 11% of the intermediate users want a submissive agent, 30% want a collaborative agent, and 59% prefer an agent that can act autonomously only in certain occasions. Finally, the 70% of inexpert users want their agents to live in the background, none of them want a collaborative agent and the 30% of these users want an agent that takes part only in certain situations. These last figures reflect the fear or disappointment (caused by Microsoft assistants) of inexpert users towards having an autonomous assistant. Most of these users want to be in control, calling their agents just when they need them. Figure 2.3 shows these results.

The type of assistant a user wants is closely tied to the types of actions the user allows the agent to execute. For example, a user who delegates most of his tasks to his agent will let him make warnings, make suggestions and mainly perform tasks on his behalf. On the other hand, a user that only allows his agent to execute his commands will not let it make warnings or suggestions and he will never tolerate autonomous actions. Hypothetically, an intermediate user will allow his agent to make warnings and suggestions, but he will probably not enable the agent to execute autonomous actions unless he asks it to do that.

As shown in Figure 2.4, most users (49%) would allow their agents to make warnings, alerts and suggestions, but they would not allow them to perform autonomous
tasks. About 27% of the users would let their agents make only warnings in an autonomous fashion, and the rest of the assistance actions only upon request. Only 22% of the users would allow their agents to execute every type of assistance actions, and 2% would only let the agents act when they are told.

2.2.3.2 Allowing actions on the user’s behalf

One of the most controversial aspects of agents is their ability to perform tasks on users’ behalf in an autonomous fashion. Moreover, the possibility that the user could not completely control the agent is considered as negative from a human-computer interaction (HCI) point of view [Whittaker and Sidner, 1996]. Thus far, some interface agents have dealt with this problem by enabling users to control agents’ behavior by setting some parameters and thresholds [Kozierok and Maes, 1993, Maes, 1994, Fleming and Cohen, 1999]. In other cases, these thresholds are set by the developers.

Our experiment has confirmed users’ fear towards having a completely automated
2.2. Personalization Issues: Empirical Study

![Pie chart showing user acceptance of autonomous actions on their behalf]

Figure 2.5: Users' acceptance of agents' actions on their behalf

agent. The results regarding this issue are shown in Figure 2.5. The 4% of the users would never allow their agents to execute tasks autonomously without any kind of control. The rest of the users would enable their agents to act autonomously provided that: users know exactly what the agent is going to do (45%); only if they ask their agents to perform the tasks (36%); only in situations that do not compromise them (15%). Most users agreed in that they "do not want to lose control of their computers" and that autonomous actions are "dangerous". On the contrary, some users find useful harmful actions such as the automatic filling of certain fields in calendar management for example.

As regards the different types of users, we found that none of the expert users would let their agents execute autonomous actions without any kind of control, 40% of the expert users would let agents perform autonomous tasks only if they are sure about their actions, 20% of these users would let agents act only in certain occasions, and 40% want their agents to act only upon users’ request. Concerning intermediate users, 48% of them would let agents act on their behalf only when they are sure about what agents are doing, 41% want agents to act only when they are told, and 11% of these users want their agents to take part only in certain situations. As regards inexpert users, 20% of them would never allow agents to perform actions on their behalf without any kind of control; 30% would enable agent to act autonomously when users know exactly what they will do; 30% of the inexpert users would enable agents to act when they request their assistance and the remainder 20% would allow agents to perform autonomous actions only in certain predefined situations. Figure 2.6 shows these results.

2.2.3.3 Does the user want suggestions or just warnings?

If an interface agent wants to be competent and wants to be accepted by the user it is assisting, it has to discover what exactly the user needs in a given context or
situation. For example, consider an agent assisting a user with his calendar management, and suppose that the user is scheduling a meeting with several participants for the following Saturday in a free time slot. The agent detects that one of the participants will surely disagree with the meeting date because he never attends meetings on Saturdays. The agent can merely warn the user about this problem, it can suggest the user another meeting date considering all participants’ preferences and priorities or it can do just nothing. Our hypothesis is that a certain user probably prefers a warning about the inconvenient, but another user surely prefers a suggestion of another meeting date in that situation. Thus, the agent will have to discover what each user prefers in each situation or context.

Our experiments revealed us that the 60% of the users could identify some situations in which they clearly prefer a particular type of assistance action rather than the others, but 40% of the users could not identify these kind of situations. In the case of users belonging to the first group, it is clear that the agent has to learn when to provide each type of assistance because each user has different preferences. Most of the users who could not identify situations in which they would prefer a particular type of assistance were intermediate and inexpert users. Probably, their inexperience did not let them identify particular situations, but this does not mean that such situations do not exist. The agent will have to discover them. Figure 2.7 shows users’ preferences regarding warnings and suggestions.

Concerning the choice between warnings and suggestions, we also studied how users prefer to be notified when the agent has something to suggest. The 67% of the users wants to be notified first about the potential suggestions and then obtain the suggestions. The rest of the users (33%) wants to get the suggestions directly,
without any previous notification. We studied this issue by providing two different user interfaces in one of our agents, and asking users which of them they found more convenient or practical. Figure 2.8 shows users’ preferences regarding this issue.

In summary, we found that different users have different preferences regarding the three assistance actions we are considering (that is warnings, suggestions and actions on the user’s behalf) and that even a single user prefers different actions to deal with a given problem depending on the context. Thus, personalization is required in order to assist each user properly.

2.2.3.4 Do users tolerate agents’ errors?

Tolerance of agents’ errors is a key point in user-agent interaction. Some experiments have demonstrated that users tolerate their agent’s errors as far as they achieve an acceptable performance [Ruvini and Gabriel, 2002]. In our experiment, we focused our attention on errors regarding the different types of assistance actions, that is warnings, suggestions and actions on the user’s behalf.

We discovered that only 7% of the users do not get angry if their agents provide them the wrong type of assistance and that they would fix the situation in the case of an error. The rest of the users would be disappointed if their agents provided them the wrong type of assistance. The 14% of the users would be highly disappointed, 50% said that their disappointment depends on the magnitude of the mistake, 27% would try to train the agent by means of explicit feedback so that these mistakes could decrease in the future, and the 2% would fix the situation by themselves
Fig. 2.9: Users’ tolerance to agents’ errors

Fig. 2.10: Users’ tolerance to different types of errors

without interacting with the agent (that is without providing explicit feedback). Figure 2.9 shows these results.

Many users get really angry if the type of assistance involved in an error is an action or if the mistake is severe (85%), but they can tolerate a suggestion instead of a warning. The 15% of users are equally tolerant (or intolerant) of errors involving the different types of assistance actions. Figure 2.10 shows these proportions.

As shown in Figure 2.11, 90% of the users are more tolerant of agent’s errors when they start the interaction with their agents, but they do not tolerate mistakes once they have interacted with their agents for a long time. The 10% of users do not tolerate mistakes at any stage of the user-agent relationship.
2.2.3.5 Do users object to providing explicit feedback?

One of the main sources of learning in interface agent technology is user feedback. This feedback may be explicit, when users explicitly evaluate an agent’s actions through a user interface provided for that purpose, or implicit, when the agent observes a user’s actions after assisting him to detect some implicit evaluation of its assistance. The explicit feedback can be simple or complex. It is simple when, for example, the user is required to evaluate the agent’s assistance according to a quantitative or a qualitative scale (for example 0 to 10, relevant or irrelevant) or to just press a dislike/like button. However, it becomes more complicated when the user is required to provide big amounts of information in various steps.

Mostly, an interface agent has to learn from implicit feedback since the explicit feedback is not always available. The reason is that not all users are willing to provide explicit feedback, mainly if this demands them a lot of time and effort.

We discovered that 35% of the users do not object to providing explicit feedback when they start their interaction with an agent, because they consider that they have to train it in early stages. However, they find annoying to provide feedback when they have interacted with the agent for some time. The 19% of the users do not object to giving explicit feedback provided that the feedback mechanisms are simple and that they do not have to spend a lot of time and effort evaluating the agent’s behavior. The 24% of the users answered that they do not complain about giving feedback if the two previous conditions are met. In the extreme cases, only 5% of the users finds completely bothersome to provide explicit feedback, and the 17% do not complain about giving feedback at any stage of the interaction because they believe it is necessary for the agent to learn about them. Figure 2.12 shows users’ reactions towards providing explicit feedback.

2.2.3.6 Do users accept agents’ interruptions?

One of the key decisions agents must take not to hinder the user’s work is discovering whether the user objects to being interrupted or not, and in which contexts this occurs. Some studies have demonstrated that notifications can be disruptive, both frustrating users and decreasing the efficiency with which they perform ongoing tasks [Cutrell et al., 2001]. Some of the consequences for an agent showing a disruptive behavior were discussed in Section 1.1.

The experiment we carried out indicates that 10% of the participants do not object to being interrupted by their agents at all. On the contrary, 14% of them do not tolerate being interrupted under no circumstances and 76% of the users do not
2.3 Lessons Learned

The experiment we carried out revealed us some interesting aspects of user-agent interaction and confirmed some assumptions about users’ reactions towards agent technology. As yet, interface agents lack the ability to personalize their interaction with users according to the user’s interaction preferences and requirements.

We discovered that there are several issues interface agent developers have to address if they want to personalize and improve the interaction between interface agents and users:

- discovering the type of assistant each user wants
2.3. Lessons Learned

- considering the particular assistance requirements users have in different contexts
- analyzing users’ tolerance to agents’ errors
- discovering when (context awareness), and when not, to interrupt the user
- providing the means to provide simple (but useful) explicit user feedback
- providing the means to capture as much implicit feedback as possible
- discovering how much control the user wants to delegate to the agent
- providing the means to control and inspect agent behavior

To personalize its interaction with the user, an interface agent has to discover what type of assistant a user wants. Some users want just a secretary that executes their orders without any autonomous intervention. Other users enable their assistants to make some suggestions and warnings, while others want an assistant capable of actively collaborating with them and of performing tasks on their behalf. The agent has to observe and analyze the user’s reactions towards the different assistance actions and discover which are those he mostly accepts. The agent has to pay also especial attention to those actions the user explicitly requests his agent to execute. This information will enable the agent to discover which the user’s interaction style is, that is authoritative, collaborator or delegator.

Second, the agent has to learn which type of assistance the user wants in the different situations that may arise. Different users require different agent actions in the same context, and even a user may require different agent actions in different instances of a given problem situation. The agent must discover when the user wants it to propose a solution to a problem or to deal with a given situation, when the user wants the agent to solve the problem on the user’s behalf, when he wants it to simply warn him about the situation and when he wants the agent to do just nothing.

Third, once the agent has learned what type of assistance the user needs, it has to learn how to provide it, that is interrupting the user’s work or not. Most users agreed in that they tolerate interruptions if the underlying problem or situation of interest is relevant to them. Thus, the agent has to analyze the relevance the situation to be notified has for the user, probably depending on the task the user is carrying out and on the different contexts.

Fourth, the agent has to discover how tolerant the user is of its assistance mistakes. Two users can react differently to the same agent mistake. For example, one
of our users hates being interrupted when he is working, even by a "new email" message. He told us that his reaction towards useless interruptions would be uninstalling the agent. On the contrary, other users are more tolerant and patient. In the former case, the agent has to be very confident when providing assistance, but in the latter cases it can attempt to make some guesses regarding the user’s preferences since these users are more tolerant.

As regards agents’ autonomy, HCI people have criticized agent-based methodologies that seem to produce systems not easily accepted by the user: one of the main reasons is the autonomy of the agents that can cause a loss of control by the user [D’Aloisi et al., 1997, Scerri et al., 2002]. We agree with these authors in that one of the reasons for this complaint is the limited research dedicated to examining the role of interface agents as a mixed-initiative interaction system, opposite to a completely automated system. An agent should be endowed with the capability of acting and autonomously proposing solutions according to the current problem, but the user must have the possibility of controlling and inspecting the agent’s decisions.

Concerning agent design, a set of users told us through the survey that they would like to configure their agents’ behaviors. For example, some users would like to be able to set the type of actions the agent can execute, or set some threshold values to control agents’ behavior, or to tell the agent what is relevant or irrelevant. Thus, the interface agent developer has to provide the facilities to allow users to configure certain parameters, which can in turn be used by the agent learning module to personalize its interaction with the user.

Another important aspect that has to be taken into account when designing an interface agent is how users will provide feedback. It is well known that most users do not like to provide explicit feedback. However, they would probably provide some if this does not require a lot of time and effort. Thus, the designer must carefully choose the appropriate interaction metaphor and the feedback interfaces in order to capture as much explicit feedback as possible. Many users would like the request for feedback to diminish as they interact with the agent. However, this situation can originate the following problem: in case of changing interests, the agent might not be able to perceive them if it cannot obtain user feedback. The developer has to find a balance between these two conflicting issues.

2.4 Summary

In this Chapter we have discussed a set of issues that interface agent developers have to take into account in order to personalize the interaction between an interface agent
2.4. Summary

and a user. We have presented the results of an experiment we carried out to study these personalization issues and we have exposed the lessons we have learned. Our study contributes to the understanding of user - interface agent interaction. The results of this study have implications on how to design interface agents in order to personalize their interaction with users. In the following Chapters we present our proposal to deal with some of the problems we have discussed.
Chapter 3

Personalizing User-Agent Interaction

In the previous Chapter we presented the results of an experiment we carried out to study the interaction between users and interface agents. As a result of our experiment we found that different users have different preferences and requirements with respect to this interaction. This result signaled the need of personalizing not only the assistance given to users but also the interaction between users and interface agents.

In this Chapter we present our proposal to deal with this personalization problem. We propose a definition for a user interaction profile that models a user’s interaction preferences and assistance requirements. The user interaction profile can be used by interface agents to enhance and personalize their interaction with users, and to adapt their behavior to each user’s assistance and interaction requirements. In turn, we propose two profiling algorithms to build the two components of a user interaction profile. One of the algorithms learns a user’s assistance requirements in different contexts, while the other learns when the user can be (or should be) interrupted and when not.

3.1 Introduction

When an agent is about to assist a user, it has to decide which of several possible actions to execute. In this work we are considering the following assistance actions: warnings, notifications, alerts and reminders of situations of interest and problems\(^1\); suggestions about how to deal with these situations; and actions performed on the user’s behalf. The situation originating an assistance action may be a problem situation (for example in calendar management, two overlapping events), situations in

\(^1\)For simplicity, from now on we call all these types of assistance actions “warnings”.
3.1. Introduction

which the user might need assistance or in which the agent might give the user a piece of advice (for example, the user is scheduling a meeting with various participants), and situations of interest (for example, a new email advertising an event relevant to the user).

Some algorithms have been proposed to decide which action an agent should execute next. These algorithms adopt mainly one of two approaches: some use decision and utility theory [Horvitz, 1999b, Fleming and Cohen, 2001a, Brown et al., 1998], and others use confidence values attached to different actions [Maes, 1994, Kozierok and Maes, 1993, Fleming and Cohen, 1999]. However, these works do not consider a user’s assistance needs and interaction requirements, the possibility of providing different types of assistance (for example warnings, suggestions), or the particularities of the situation at hand.

For example, the work reported in [Horvitz, 1999a] and [Horvitz, 1999b] presents the LookOut system, which helps users that use Microsoft Outlook messaging and scheduling system. The system identifies a user’s goals automatically by considering the content of messages being reviewed. Depending on the inferred probability about a user’s goals and on an assessment of the expected costs and benefits of action (expected utility), the system decides to: do nothing but simply wait for manual invocation of LookOut, to engage the user in a dialog about his intentions with regards to providing a service, or to go ahead and attempt to provide its service.

The work presented in [Fleming and Cohen, 2001b] and [Fleming and Cohen, 2001a] describes a utility-based decision making process to determine when to take the initiative to interact with a user to request further assistance from him. In this context, the system is working on a problem and it is at a point where it might benefit from asking the user a question. Thus, it must decide whether to interact or not. The decision making is based on a calculation of the expected utility of several courses of action and the likelihood that the user will be an effective contributor of information, if an interaction were initiated.

In these utility theory based works, utilities are assigned default values or they are specified by users. No study of the user’s preferences is done to estimate them. We consider that a decision theoretic approach is not feasible to be applied to our problem because, as these works demonstrated, it is difficult to assess the cost and benefits of the different assistance actions. We should obtain utility models that match each user’s interaction preferences and assistance requirements, instead of letting the developer or the user set them arbitrarily. We will discard this approach because we do not consider it viable to solve our problems. From now on, we will concentrate on the confidence-based approach.
This approach, used in [Maes, 1994, Fleming and Cohen, 1999, Kozierok and Maes, 1993], is based on the confidence on an agent action. The confidence on an action indicates how sure the agent is about executing that action, and it is computed according to the agent’s experience at assisting a user. For example, in [Maes, 1994] the agent computes the confidence on the prediction of an action to the current situation taking into account how many similar situations the agent has memorized, whether or not all the nearest neighbors of the situation recommend the same action, and how close or distant these nearest neighbors are.

In the confidence-based approach, interface agents have generally three possibilities when they want to assist a user: executing a task autonomously, suggesting the user what to do, and doing nothing. These agents use two threshold values to take decisions, which are established by the user to control the agent’s behavior: do-it threshold and tell-me threshold. If the confidence value associated with an agent action is smaller than the tell-me threshold the agent does nothing; if the confidence value is greater than the tell-me threshold but smaller than the do-it threshold, the agent tells the user what it thought he would do and it waits for confirmation to automate the action; and if the confidence value is greater than the do-it threshold the agent executes the task autonomously on the user’s behalf, sending him a report. The do-it threshold is higher than the tell-me threshold. If the agent does not execute an action, then the user has to deal with the situation at hand, and the agent observes his behavior to learn from it.

Algorithm 1 shows a high level decision making algorithm, which is an abstraction of the algorithms used by the interface agents built under the confidence-based approach.

**Algorithm 1** High level decision making algorithm

**Input:** A problem situation $Sit$ to deal with

**Output:** The agent has executed an action to deal with $Sit$

1. Select action $A$ via learning techniques to deal with $Sit$
2. Compute confidence value $C$ for $A$
3. if $C \geq \text{do-it threshold}$ then
   4. Perform action $A$
5. else if $C \geq \text{tell-me threshold}$ then
   6. Suggest action $A$
7. else
   8. Do nothing
9. end if

The confidence-based approach has several problems. The main one is that confidence values do not consider the way in which the user wants to interact and
work with his agent. These agents do not take into account when the user wants each type of assistance action. They do not take into account when the user wants a suggestion, when he only wants a warning about a problem, when he wants the agent to execute an action on his behalf, or when the user does not want any assistance at all. The decision making algorithm should select the assistance action the user expects and will accept. Thus, despite the agent finds a good solution to a given problem it also has to analyze whether the user wants to be informed about it or not.

Moreover, threshold values are generally set by the user and they have fixed values. Thus, if the agent wants to modify them according to the user's behavior, it cannot do it. Finally, the same thresholds are used for every agent action and every problem situation or situation of interest\textsuperscript{2}. This generalization can lead to inappropriate agent behavior since the agent is not supposed to react in the same way to every situation. For example, a calendar agent will not use the same threshold values for suggesting the user a meeting date than for suggesting an answer to an invitation, since it has to be more accurate in this last situation.

The interface agent has to consider not only the confidence on the assistance to be provided - what it is done - but also on the type of the assistance. For example, a calendar agent might know the place and time a user would schedule a meeting with his co-workers, but probably the user does not want it to make the suggestion or to schedule the meeting on his behalf. On the other hand, despite the agent probably is not confident enough to make a suggestion, the user might prefer it instead of no assistance at all.

In summary, the problem with the current action selection or decision making algorithms is that they do not take into account how the user wants to be assisted and how he prefers to interact with the agent in different contexts. User assistance and user-agent interaction should be personalized and contextualized in order to assist users as they expect. In consequence, the relationship between users and agents will be enhanced.

In this work we propose a new solution to decide what an agent should do next. Our approach takes into account not only the agent confidence on the various assistance actions, but also the user's requirements and preferences regarding these assistance actions. Thus, when the agent has to decide among various actions it will consider how the user wants to be assisted in the particular situation the agent is dealing with. Section 3.2 presents an overview of our proposed action selection

\textsuperscript{2}From now on we will use situation for: a problem situation, a situation of interest, and a situation in which the agent believes the user might need assistance.
3.2 Proposed approach: Overview

When the user asks the agent for help or when the agent detects a situation in which the user might need assistance, it has to decide what to do in that situation in order to assist the user according to his assistance requirements and interruption preferences. Algorithm 2 shows the different steps of our proposed decision making algorithm. The aim of this section is giving an overview of our proposal. More details about this algorithm are given in Section 3.5.

As mentioned in Chapter 1, the user profile contains the information the agent needs to select the type of assistance the user expects, to determine what to do to assist the user and how to provide this assistance, that is interrupting the user or not. Given a problem situation that occurs while the user is working with the computer, the agent looks in the user profile for a profile item containing the situation and, preferably, the current user task at hand. If no such item exists, then we use the general confidence-based algorithm described in Algorithm 1. This general algorithm is used because the agent does not have information about the user’s assistance requirements or interruption preferences for the situation it is dealing with. Thus, it has to act as interface agents have acted thus far.

On the other hand, if such a profile item is found the agent retrieves from this item the type of the assistance action (warning, action, suggestion), the assistance to be provided (what to do, what to warn, or what to suggest), and the assistance modality (interrupting the user or not) associated with the situation and the current user task. Then, the confidence on the action is computed. The confidence tells us how sure the agent is about executing a certain assistance action to help the user in a given situation. The confidence is compared against the corresponding threshold value, one for each action type. If the confidence value is greater than the threshold value, then the assistance action is correct, and it will be executed. However, if the confidence value is smaller than the threshold value, the final decision is a tradeoff between the type of action the user expects and the confidence on the assistance to be provided. The alternatives are prioritizing the user’s requirements or prioritizing the confidence on the agent’s actions. In the first case, the action the user will execute is the one suggested by the user profile despite it has a low confidence value. The problem with this choice is that the trust in the agent can diminish because it can make mistakes. In the second case, a different action type has to be selected since the agent is not confident enough to execute the type of action the user expects.
Thus, if the type of assistance is an action on the user’s behalf and its confidence is low, the agent will make a suggestion. Similarly, if the action is a suggestion with a low confidence value, the agent will only make a warning.

The second alternative is the one we have chosen. However, to take the best decision the agent should learn what the user prefers. Thus, the agent will inform the user about the low confidence problem and let him decide what to do. In this way, the agent will learn whether the user prefers a suggestion with low confidence or a just a warning about a problem.

Algorithm 2 Our high level decision making algorithm

Input: A problem situation Sit to deal with, the user profile, the current user task Task

Output: The agent has made a decision Dec that involves a type of assistance action, the assistance action itself, and its modality

1: profitem ← retrieve from the user profile an item containing Sit and, if available, also containing Task
2: if profitem ≠ ∅ then
3: Dec.action ← profitem.action
4: Dec.modality ← profitem.modality
5: Dec.type ← profitem.type
6: conf ← compute confidence on action for Sit and Task
7: if conf > threshold value then
8: return Dec
9: else
10: Dec.type ← tradeoff between action expected and confidence
11: return Dec
12: end if
13: else
14: call Algorithm 1 {no profile item found for Sit}
15: end if

To make our decision making algorithm work, the agent needs information about a user’s assistance requirements and interruption preferences. This information is contained in our proposed user profile, which is described in Section 3.3. Then, Section 3.4 presents an overview of our proposed profiling algorithm, which is used to build the user profile.

### 3.3 Proposed user profile

In order to provide personalized assistance to computer users, interface agents need knowledge about their needs, habits and preferences. This knowledge is contained
in the user profile. A profile is a description of someone containing all the most important or interesting facts about him or her.

As discussed in Chapter 1, a user profile contains application-dependent and application-independent user information. Application-independent information includes mainly personal information about the user. Application-dependent information includes a user’s interests, preferences, goals, working habits, behavioral patterns, knowledge, needs, priorities, and commitments regarding a particular domain. To abstract the elements modeled in a user profile, we can represent all these items in a general form through a set of preferences with associated confidence values, the relations among some of these preferences, and the exceptions that may arise for some of them. We can also model temporal preferences and preferences that may show some kind of seasonality. We name this profile “standard user profile”, that is the profiles used thus far by interface agents, to distinguish it for our definition of a user profile. We are not concerned about how the standard user profile is generated. To build this profile, the agent can use one of the many approaches reported in the literature [Billsus and Pazzani, 1999, Kozirok and Maes, 1993, Lieberman, 1995, Maes, 1994]. At the top of Figure 3.1 we can observe the generalized components of a standard user profile.

In our approach, a user profile should contain all the information concerning user-agent interaction that an interface agent needs to provide personalized assistance to a user. We define a user profile as:

\[
\text{user profile ::= standard user profile + user interaction profile}
\]

A user interaction profile is composed of a set of user assistance requirements

---

**Figure 3.1: Proposed User Profile**
3.3. Proposed user profile

and a set of user interruption preferences. Assistance requirements are defined as a set of situations with the required assistance actions, and a parameter (named certainty, to differentiate it from the confidence on an action) indicating how sure the agent is about the user wanting it to execute that action to assist him in that particular situation. In a BNF-like notation:

\[
\text{user interaction profile ::= user assistance requirement}^* + \text{user interruption preference}^*
\]

\[
\text{user assistance requirement ::= situation + agent action + certainty}
\]

The situation describes the problem situation, situation of interest or assistance opportunity underlying the interaction between the user and the agent. Examples of situations in the calendar management domain are: there are two overlapping events scheduled by the user; the user receives an email inviting him to take part in an event; the user is scheduling a new event; the user is scheduling a business event for a holiday; the user is scheduling an event next to another event, but the associated places are far apart. Each situation is described by a set of attributes or features, and each attribute can take a predefined set of values. Formally, a situation is defined as \( \text{situation ::= (feature}_i, \text{value}_j) \), where \( \text{value}_j \) is the \( j-th \) value feature \( i \) can take. For example, if two events overlap the situation will be described by the information of the two events involved, such as date, time, type of event, organizer, participants, place, priority, and topic.

The agent action associated with a given situation may be a warning, a suggestion, an action on the user's behalf, or no assistance action at all. The certainty degree can take a value between 0 and 1.

\[
\text{agent action ::= warning|suggestion|action|no action.}
\]

\[
\text{certainty ::= [0..1]}
\]

Interruption preferences are defined as a set of situations with the preferred assistance mode or modality, and optionally the primary user task, the relevance the situation being notified has to the current task, and the type of assistance action to be executed. The modality associated with the assistance action indicates whether the agent should interrupt the user or not. The user task represents the task the user was carrying out when the assistance was provided, scheduling a new event in our example. Profile items can also indicate the relationship between the situation originating the assistance and the user task. A situation can be relevant or irrelevant to the primary user task, and in this latter case, it can be related or unrelated to it. A certainty value indicates how certain the agent is about this user preference.

\[
\text{user interruption preference ::= situation + [user task] + [task relevance] +}
\]
3.4. Building User Interaction Profiles

[agent action] + assistance modality + certainty

assistance modality ::= interruption | notification

task relevance ::= relevance + relationship

relevance ::= relevant | irrelevant

relationship ::= related | unrelated

The user task is represented as feature-value pairs in the same way as situations. These features are domain dependent. Formally, user task ::= (feature, value) + , where value is the j-th value feature i can take. The user task may coincide with the situation when, for example, the user is scheduling a new event and the agent suggests him some of the event features.

Two examples of user interaction profile components in a calendar management application are shown below. The first example indicates that when two events overlap and one of them is a meeting organized by the user, the user requires a warning about the problem. The features of the second event are not relevant. The second example expresses that when an email asking the user to attend a meeting organized by the user’s boss arrives, the user wants to be interrupted. No information is given about the current user task and the relevance of the situation to the task.

- assistance-requirement (situation (type(event overlapping), features(event-one( (event-type, meeting), (host, user)), action(warning), certainty(0.7))

- interruption-preference (situation (type(new email), features(sender(boss), topic(meeting))), action(warning), modality(interruption), certainty(0.8))

3.4 Building User Interaction Profiles

In the previous section we presented our definition of a user interaction profile. In this section we describe how we build this user interaction profile. Subsection 3.4.1 describes our interaction profiling approach. Subsection 3.4.2 describes the data the agent obtains by observing the user’s behavior, which is used to build the user interaction profile. Finally, subsection 3.4.3 presents an overview of our two learning algorithms.

3.4.1 Proposed Interaction Profiling Approach

The typical information sources in the interface agent paradigm are: the observation of the user’s behavior, the information explicitly provided by the user, the user’s feedback, and other agents’ help [Maes, 1994]. Thus far, agents observe a user’s behavior with respect to a computer application in order to obtain his preferences and
habits. The information users can optionally provide also consists of his preferences with respect to the application. The feedback the agent gathers help it to adjust the information about the user’s preferences. In our approach, an agent also has to obtain information about the user’s interaction preferences. Thus, the agent will observe not only the user’s behavior while he is working with a computer application, but also his behavior when he interacts with the agent. In turn, the agent has to gather feedback with respect to this interaction. It has to obtain feedback for the type of assistance provided and for the modality of this assistance, and not only for what the agent did or suggested. Our focus is not on the user’s actions with the computer application as in the typical interface agent paradigm, but on the user-agent interaction.

Figure 3.2 illustrates our profiling approach. This graphic shows how an interface agent can build a user interaction profile by recording information about the actions a user requests from his agent, the assistance actions the agent performs, and the feedback the user provides to the agent’s assistance actions. This feedback can be explicit (direct) or implicit (indirect), as well as positive or negative. It is explicit when the user explicitly evaluates the agent’s actions using some mechanisms provided by a user interface. It is implicit when the agent has to observe the user’s actions after it has assisted him in order to obtain the feedback. In turn, the user can also explicitly state some interaction preferences. However, asking the user when it is convenient to interrupt him or when he needs a suggestion instead of a warning may be not a viable solution, since he probably does not know what to answer and he may not want to spend time providing this information.

In this work we will not consider that the agent can obtain information about the user from other agents. Although two users may have similar interests, they would certainly not interact with their agents in the same way. We separated the two kinds of user profiles in the figure to make the difference between them clear, although they both compose the user profile as shown in Figure 3.1.

### 3.4.2 Data Model

In our approach an interface agent records every interaction between the user and the interface agent. There are several aspects of the interaction that we need to record: the situation or context in which the interaction takes place, the assistance action the agent executes, the task the user was performing when he was assisted, the relationship between the current user task and the situation, the user feedback (explicit and implicit), how the assistance was provided (with or without an interruption), a conclusion or evaluation of the interaction (that is if it was a success
3.4. Building User Interaction Profiles

Figure 3.2: Proposed profiling approach

or a failure in terms of user assistance), and the date when the interaction took place. All these items form a user-agent interaction experience, formally defined as 
\[
\text{experience} ::= \text{situation} + \text{agent action} + \text{modality} + \text{user task} + \text{task relevance} + \text{user feedback} + \text{evaluation} + \text{date},
\]
where the brackets mean that the items are optional. The following paragraphs describe each of these items, most of which were defined in Section 3.3.

- **Situation description**: This item describes the problem situation, situation of interest or assistance opportunity underlying the interaction between the user and the agent. It is described by feature-value pairs, which depend on the application domain.

- **Agent action**: This item represents the action the agent executes to deal with the situation at hand. The agent action has a type and a content. The action type can be a warning, a suggestion, or an action on the user’s behalf. The content expresses what the agent suggested in the case of a suggestion, or what the agent did in the case of an action on the user’s behalf. This information is domain dependent and it is beyond the scope of our work. We only consider the action type as part of the experience\(^3\).

- **Assistance Modality**: It indicates how the agent provided assistance to the user, that is interrupting his work, without interrupting his work, using sounds,

\(^3\)The agent obtains what to suggest or what to do from the standard user profile. This is not our focus. We have to obtain the appropriate assistance action type and its modality, and we do not need to worry about the assistance content. We assume that we have this information.
using colors, using movement, blinking, among other display options. In this work we will only consider whether the agent interrupted the user or not to provide him assistance.

- **User Task**: It represents the primary task the user is engaged in when the interaction between him and the agent takes place. This item is useful when analyzing whether to interrupt the user or not, as explained in Chapter 5.

- **User Task Relevance**: It indicates the relationship between the primary user task and the situation motivating the interaction between the user and the agent. A situation can be relevant or irrelevant to the primary user task, and in this latter case, it can be related or unrelated to it. This item is useful when analyzing whether to interrupt the user or not, as explained in Chapter 5.

- **User feedback**: The user can explicitly evaluate the agent’s performance by giving feedback through a user interface provided for that purpose. This type of feedback is optional, since many users are unwilling to provide explicit feedback and others consider it intrusive. Thus, the agent also observes the actions the user executes to obtain an indirect or implicit feedback. The user feedback is composed of different parts: feedback regarding the type of assistance provided, feedback regarding the modality of this assistance, and feedback regarding the content of the assistance. This item is domain dependent, since it can adopt different forms according to the particular application under consideration.

- **Conclusion or evaluation of the interaction**: Once the agent has gathered feedback for its assistance, it has to evaluate if the assistance interaction was a success, a failure or undetermined, that is neither a success nor a failure. 
  
  \[
  \text{evaluation ::= success| failure| undefined.}
  \]

- **Date**: It records the date (day, month, year) when the interaction between the user and the agent takes place. This item is considered since our algorithms may use a fading function to forget old and obsolete interactions.

For example, if we consider an agent assisting a user of a calendar management system, an assistance experience could be the following: the user is scheduling a new event, a meeting to discuss the evolution of project A with his employees Johnson, Taylor and Dean. The event is being scheduled for Friday at 5 p.m. at the user’s office. The agent has learned by observing the user’s actions and schedules that Mr. Dean will probably disagree on the meeting date and time because he never
3.4. Building User Interaction Profiles

schedules meetings on Friday evenings. Thus, it decides to warn the user about this problem. In reply to this warning, the user asks the agent to suggest him another date for the event. In this example, the different parts of the assistance experience are:

- Situation = \{(type, new-event), (event-type, business meeting), (organizer, user), (participants, [Johnson, Taylor, Dean]), (topic, project A evolution), (date, Friday), (time, 5p.m.), (place, user’s office)\}

- Action = \{(type, warning), (content, Johnson does not like meetings on Friday evenings)\}

- Modality = notification

- Task = \{(type, new-event), (event-type, business meeting), (organizer, user), (participants, [Johnson, Taylor, Dean]), (topic, project A evolution), (date, Friday), (time, 5p.m.), (place, user’s office)\}

- Relevance = relevant

- User Feedback = \{(type, explicit), (action, asks for a suggestion)\}

- Evaluation = failure (suggestion instead of warning)

- Date = 18/02/04

3.4.3 Proposed learning algorithms overview

The user interaction profile contains two components. Thus, we propose two profiling algorithms to obtain these components: WATSON and IONWI. The WATSON (acronym for Warning, A|C|T|ion, Suggestion or No action) algorithm is in charge of obtaining a user’s assistance requirements. It has to learn when the user requires a suggestion to deal with a situation, when he needs a warning and when the user wants the agent to perform a task on his behalf. The goal of the IONWI (acronym for Interruption Or Notification Without Interruption) algorithm is obtaining a user’s interruption preferences. It learns when the agent can interrupt the user’s work to provide him assistance, and when it only can send him a notification without interrupting him.

Figure 3.3 shows the relationships among the different profiling algorithms an interface agent uses to build a user profile under our approach. The figure components in grey are those under the scope of our work. As we have mentioned, in our approach an interface agent stores data about a user’s interaction with the computer application as well as about the interaction between the agent and the user. The first database is used by the standard profiling algorithms to learn about a user’s
preferences, habits, interests and priorities regarding a given computer application. We grouped these items in the standard user profile. The second database, a set of user-agent interaction experiences each described by the items enumerated in the previous section, is used by our profiling algorithms to learn a user’s assistance requirements and interruption preferences.

The outputs of our algorithms constitute the user interaction profile, which is used by the interface agent to decide how to assist a user in a given situation. The decision making (or agent action selection) algorithm uses the information contained in the user profile to decide which assistance action to execute and how to execute it to provide assistance to the user. The contents of the assistance action (for example, the suggestion made or the action performed) are obtained from the standard user profile, as it has been done thus far.

The two algorithms are incremental since they keep learning as new user-agent interaction experiences arrive. In this way, they can also detect changes in a user’s interaction preferences.

Our algorithms use association rules as machine learning technique. Association rules enable us to obtain relationships between items in a domain [Agrawal and Srikant, 1994]. In this work, we use association rules with two purposes: to discover the existing relationships between problem situations or situations of interest and the assistance actions a user requires to deal with them, and to determine the relationships between situations, the primary user task and the modality of the assistance required.

Our objective was obtaining knowledge about a user’s assistance requirements and interruption preferences from the information recorded while observing and
interacting with the agent. Although this knowledge can be used to predict the user’s future actions, our goal was understanding a user’s behavior when he interacts with his agent in order to assist him better. Thus, we considered that our task was more descriptive than predictive. Consequently, we chose association rules to implement our algorithms, and we discarded classification and inductive techniques\(^4\). As a future work, we will compare our first proposed technique against others to evaluate their performance and, eventually, find a better solution to the problem we are dealing with.

Algorithm 3 shows the main steps our algorithms involve when using association rules as machine learning technique. The association rules generated from the user-agent interaction experiences are automatically post-processed in order to derive the user’s assistance requirements and interruption preferences. Post-processing includes detecting the most interesting rules according to our goals, eliminating redundant and insignificant rules, pruning out contradictory weak rules, and summarizing the resultant information. The rules surviving the filtering steps constitute hypotheses about a user’s preferences. Once a hypothesis is formulated, the algorithm looks for positive evidence supporting the hypothesis and negative evidence rejecting it in order to validate it. The certainty degree of the hypothesis is computed taking into account both the positive and the negative evidence. Finally, facts (user assistance requirements and interruption preferences) are generated from the set of highly supported hypotheses.

The steps shown in Algorithm 3 are common to JONWI and WATSON. However, they differ in how each step is performed within the two algorithms.

### 3.5 Detailed decision making algorithm

In Section 3.2 we presented an overview of our decision making algorithm. Now, we will explain in detail the different steps this algorithm involves. Algorithm 4 shows our detailed decision making algorithm.

The inputs for this algorithm are the situation the agent must handle, the current user task, and the user profile. The output of the algorithm is a decision that consists of the type of the assistance action (warning, suggestion, action on the user’s behalf), the content of the assistance action (what to do or what to suggest), and the modality of the assistance action (interruption or notification).

\(^4\)If one wants to adopt a predictive approach, some changes would have to be made to the recorded data and to the problem formulation. Then, classification techniques such a Bayesian classifiers and decision trees could be used.
3.5. Detailed decision making algorithm

**Algorithm 3** WATSON and IONWI Overview

**Input:** A set $Ex$ of user-agent interaction experiences $Ex_i = <$ 
$Sit_i, Act_i, UF_i, E_i, Mod_i, Task_i, Rel_i, date_i >$

**Output:** A set $F$ of facts representing the user’s assistance requirements and interruption preferences

1. $F \leftarrow \emptyset$
2. $H \leftarrow \emptyset$
3. $AR \leftarrow $ Call association rule mining algorithm with $Ex$
4. $AR_1 \leftarrow $ Filter out uninteresting rules from $AR$
5. $AR_2 \leftarrow $ Eliminate redundant and insignificant rules from $AR_1$
6. $AR_3 \leftarrow $ Eliminate contradictory weak rules from $AR_2$
7. $AR_4 \leftarrow $ Summarize the discovered rules contained in $AR_3$
8. $H \leftarrow $ Transform rules in $AR_4$ into hypotheses
9. **for** $i = 1$ to size of $H$ **do**
10. Find evidence for $(E^+)$ and against $(E^-)H_i$
11. $Cer(H_i) \leftarrow $ compute certainty degree of $H$ considering $(E^+)$ and $(E^-)$
12. **if** $Cer(H_i) \geq \delta$ **then**
13. $F \leftarrow F \cup H$
14. **end if**
15. **end for**

To make a decision, the agent first searches the standard user profile looking for a profile item involving the situation at hand. If no profile item is found for this situation, then the agent cannot provide assistance to the user because it does not have information to do it. Thus, no assistance action is executed. Another alternative could be warning the user about the problem, but only in those cases where the confidence on the problem situation is high enough.

If a profile item is found, then the agent has information to assist the user. Thus, it has to decide the type of the assistance to be provided. To achieve this goal, the agent looks into the user interaction profile for an assistance requirement containing the situation at hand or a similar situation. If such an assistance requirement is not found, then the confidence-based decision making algorithm is used. This algorithm has to be used because the agent does not know the user’s assistance requirements or interruption preferences for the situation it is dealing with. Thus, the algorithm used currently by interface agent has to be used.

On the other hand, if such an assistance requirement is found the agent retrieves from this item the type of the assistance action, the assistance to be provided, and the assistance modality associated with the situation and the current user task. Then, the confidence on the action is computed. The confidence is compared against the corresponding threshold value, one for each action type. If the confidence value is greater than the threshold value, then the assistance action is correct and it will
Algorithm 4 Detailed decision making algorithm

Input: A problem situation \( S_i \) to deal with; the user profile composed by the standard user profile \( SUP \), the user interruption preferences \( UIP \) and the user assistance requirements \( UAR \); the current user task \( Task \)

Output: The agent has chosen a type of action to deal with \( S_i \): warning \( W \), suggestion \( S \), action \( A \) or no action \( N \); a modality: interruption \( I \) or no interruption \( NI \); and a content. These three items constitute the decision \( Dec \)

1. \( Dec \leftarrow \emptyset \)
2. compute \( Conf(S_i) \)
3. if \( Conf(S_i) \geq \tau_1 \) then
4. \( Dec.content \leftarrow \) select from \( SUP \) a profile item containing \( S_i \)
5. if \( Dec.content \neq \emptyset \) then
6. \( AR \leftarrow \) select from \( UAR \) a profile item containing \( S_i \) (or \( S_{i1} \) similar to \( S_i \))
7. if \( AR \neq \emptyset \) then
8. \( type \leftarrow AR.agent.action \)
9. if risky user then
10. \( Dec.action \leftarrow type \)
11. else
12. if \( type = action \) and \( Conf(Dec.content) \geq \tau_2 \) then
13. \( Dec.action \leftarrow A \)
14. else if \( (Conf(Dec.content) < \tau_2) \) or \( (type = suggestion \) and \( Conf(Dec.content) \geq \tau_3) \) then
15. \( Dec.action \leftarrow S \)
16. else if \( (type = warning) \) or \( (Conf(Dec.content) < \tau_3) \) then
17. \( Dec.action \leftarrow W \)
18. else
19. \( Dec.action \leftarrow N \) \{\( type = no \) action\}
20. end if
21. end if
22. else
23. \( Dec.action \leftarrow \) Call confidence-based decision making algorithm
24. end if
25. \( IP \leftarrow \) select from \( UIP \) a profile item containing \( S_i \) and \( Task \)
26. if \( IP \neq \emptyset \) then
27. \( IP \leftarrow \) select from \( UIP \) a profile item containing \( S_i \)
28. end if
29. if \( IP \neq \emptyset \) then
30. \( Dec.modality \leftarrow IP.modality \)
31. else
32. \( Dec.modality \leftarrow \) analyze relationship between \( Task \) and \( S_i \)
33. end if
34. else
35. \( Dec.action \leftarrow N \) or \( W \)
36. end if
37. else
38. \( Dec.action \leftarrow N \) \{the agent considers the situation is not worth handling\}
39. end if
40. return \( Dec \)
be executed. However, if the confidence value is smaller than the threshold value, then we have to analyze what the user wants. If the user is risky, that is if he wants the action despite it has a low confidence value, then the action contained in the user assistance requirement is performed. Otherwise, a different action is executed. If the action is an action on the user’s behalf but the confidence on the action is smaller than a threshold value \( \tau_2 \), then the agent will make a suggestion. If the action contained in the assistance requirement is a suggestion and its confidence value is smaller than \( \tau_3 \), the agent will only make a warning.

An important issue interface agents developed thus far do not consider in their action selection algorithms is how to provide assistance to the user. Once the agent has selected which action to perform it has to decide whether to interrupt the user or not, in order to provide him assistance. Thus, it looks for an interruption preference containing the situation and, preferably, also containing the current user task. If such a preference exists, the agent retrieves the modality associated with the situation-task pair or with the situation only. If no interruption preference exists, we can use the following approximation. If the situation is rated as related or relevant to the user and to the current user task, the agent interrupts the user. Otherwise, it provides him assistance without interrupting his work, letting him decide when to pay attention to that assistance. More details about how to decide between an interruption or a notification are given in Chapter 5.

### 3.5.1 Characteristics of the proposed algorithm

#### 3.5.1.1 Domain dependence

The algorithm is applicable to those situations in which there are more than one possible assistance actions the agent can execute. If there is only one possibility, such as a warning, then there is no need to use this algorithm. The algorithm based on confidence values will suffice.

The definitions of situations and user tasks is also domain dependent. Consequently, the comparison of situations and the comparison between situations and user tasks (to determine their relationship) is also domain dependent. The feedback the user can provide also depends on the application domain the agent is dealing with.

#### 3.5.1.2 Threshold values

Our algorithm considers three threshold values, namely \( \tau_1 \), \( \tau_2 \) and \( \tau_3 \). The \( \tau_1 \) threshold determines when a situation is worth handling and, thus, a warning can be
made. The $\tau_2$ threshold is equivalent to the do-it threshold and $\tau_3$ is equivalent to the tell-me threshold. We have not considered in the algorithm threshold values neither confidence values for the type of assistance action and the modality of the assistance, since they are implicitly considered in the user interaction profile. The profile items are those that have a confidence value higher than a given threshold, as we will explain in detail in the following Chapters.

Thus far, we have considered the same threshold values for each type of assistance action. When the developer has information about the application domain, then he can consider different values for each problem situation, for each suggestion and for each action. We suggest the consideration of different threshold values for the different situation-action pairs.

3.5.1.3 Comparison of situations

As shown in line 6 in our decision making algorithm, if the agent does not find an assistance requirement with the situation at hand, it looks for a similar one. We use a similarity metric to determine if two situations are similar or not. This metric computes a similarity function that gives as result a score representing how similar the situations are. If this score is higher than the threshold value $\psi$ specified by the metric, then the situations are similar. Equation 3.1 shows the function we use to compute the similarity between situations $Sit_i$ and $Sit_j$. In that function, $w_k$ is the weight of feature $k$, $sim_k$ is the similarity function for feature $k$, and $f_{ik}$ and $f_{jk}$ are the values for features $k$ in $Sit_i$ and $Sit_j$, respectively. The value $p$ is the minimum of $p_i$ and $p_j$, which are the number of features in the first situation and the number of features in the second situation, respectively. We assume that features $k$ in both representations are comparable, i.e. they refer to the same attribute.

$$similarity(Sit_i, Sit_j) = \sum_{k=1}^{p} \frac{sim_k(f_{ik}, f_{jk}) \cdot w_k}{p}$$

(3.1)

As we have said, the comparison of situations is domain dependent. Thus, the agent designer will have to provide the similarity functions for the different types of situations the agent will deal with.

3.5.1.4 How to compute the confidence on an agent action

Thus far, different authors have considered different factors to compute the confidence on an agent action. We will adopt a variation of the calculus used in [Maes, 1994], since we are interested in the confidence on the type of assistance. We will consider the amount of past similar situations, the similarity between the current
3.6. Summary

The confidence on a problem situation or the confidence on a warning about this situation, \( Conf(Sit) \), is computed as the ratio of the amount of times the agent made a warning about situation \( Sit \) (or a similar one considering the formula in 3.1) to the amount of times the problem \( Sit \) occurred. Equation 3.2 shows this formula.

\[
Conf(Sit) = Conf(Sit, W) = \frac{\text{number of }(Sit\text{-warning}) \text{ pairs}}{\text{number of times } Sit \text{ occurred}} \tag{3.2}
\]

The confidence on a suggestion, that is on a solution proposed to deal with a given problem situation, is computed as the ratio of the amount of times the agent proposed the solution to the amount of times the situation occurred. The formula is shown in Equation 3.3.

\[
Conf(Sit, S) = \frac{\text{number of }(Sit\text{-S}) \text{ pairs}}{\text{number of times } Sit \text{ occurred}} \tag{3.3}
\]

Similarly, as shown in Equation 3.4 the confidence on an agent action on the user’s behalf can be computed considering the amount of times the agent executing an action on the user’s behalf and the total amount of times the agent dealt with the situation.

\[
Conf(Sit, A) = \frac{\text{number of }(Sit\text{-A}) \text{ pairs}}{\text{number of times } Sit \text{ occurred}} \tag{3.4}
\]

3.6 Summary

In this Chapter we have presented our proposal to deal with two of the problems interface agents have to solve in order to personalize their interaction with users according to users’ assistance requirements. The following chapters describe in detail the two profiling algorithms in charge of building a user interaction profile.
Chapter 4

The \textit{WATSON} Algorithm: Does the user want a \textit{Warning}, an \textit{ActIon}, a \textit{Suggestion} or \textit{Nothing}?

One of the problems interface agents have to solve to personalize their interaction with users is discovering what type of assistance each user requires in the different situations that may arise when a user works with a given computer application. Particularly, an interface agent has to learn when the user wants a suggestion to deal with a situation, when he requires only a warning about it, when the user wants the agent to perform a task on his behalf, and when he wants the agent to do just nothing.

4.1 Introduction

In this chapter we present the \textit{WATSON} learning algorithm (acronym for \textit{Warning, ActIon, Suggestion Or Nothing}), which enables an interface agent to learn a user’s assistance requirements in different contexts. The goal of our algorithm is learning which assistance action the user expects from an interface agent in each problem situation or situation of interest that may arise when he interacts with a certain application. Our algorithm is based on the observation of a user’s actions and a user’s interaction with the agent, particularly on a user’s reactions to the agent’s assistance actions. As shown in Figure 4.1, the information obtained from observation is stored as user-agent interaction experiences, which are the inputs for our algorithm. The output of the \textit{WATSON} algorithm is a set of user assistance requirements that are part of the user interaction profile. When an interface agent has to decide among various assistance actions to deal with a problem, the user interaction profile enables
it to choose the one that is the most acceptable to the user in that particular instance of a given situation.

The rest of this Chapter is organized as follows. Section 4.2 describes in detail the inputs and outputs of our algorithm. Section 4.3 describes the different steps our algorithm involves. Section 4.4 describes how the knowledge obtained with WATSON is updated as new interaction experiences are recorded. Finally, in Section 4.5 we develop a complete example of the utilization of WATSON.

4.2 WATSON Inputs and Outputs

The input for our learning algorithm is a set of user-agent interaction experiences that take place when an agent assists a user with regards to a given computer application. An interaction experience starts when one of the participants, user or agent, initiates an interaction which involves an agent’s assistance action, and it ends when the agent gets some feedback from the assistance it has provided.

For example, in calendar management, an interaction experience occurs when the agent detects that the user is scheduling a new event and it realizes that it could assist him. Thus, it suggests the user the event place and time. The user accepts the agent’s suggestion but he changes the suggested time. From this interaction we can conclude that the user appreciates suggestions when scheduling a new event. We can also conclude that the agent has to acquire more knowledge to make a better suggestion the next time.

We will recall some definitions given in Chapters 1 and 3. An interaction ex-
perience $Ex$ is described by five arguments\(^1\) $<Sit, Act, UF, E, date>$: a situation $Sit$; the assistance action $Act$ the agent executes to deal with the situation; the user feedback $UF$ obtained after assisting the user; an evaluation $E$ of the assistance experience (success, failure or undefined); and the $date$ when the interaction experience was recorded.

$Sit$ describes the situation that originates the interaction between the user and the agent. The interaction can be initiated by the user to request the agent’s help, or by the agent to proactively assist the user. In a calendar management application a situation may be one of the following: there is an overlapping between two events scheduled by the user; the user receives an email inviting him to take part in an event; the user is scheduling a new event and the agent can suggest date, time and/or place; the user is scheduling a business event for a holiday; the user is scheduling an event in a time slot next to another event’s time slot, but the places where they take place are far apart. Each situation $Sit$ is described by a set of features and the values these features take, $Sit = \{(feature_i, value_i)\}$. The set of features and feature values vary from one problem situation to another. For example, if two events overlap the situation will be described by the information of the two events involved, such as date, time, type of event, organizer, participants, place, priority, and topic. However, if the underlying situation is a new event, the characteristics of this event will describe the situation.

An assistance action may be a suggestion, a warning or an action on the user’s behalf. The user feedback may be explicit, if the user explicitly evaluates the agent’s actions, or implicit if the agent has to obtain it from the user’s actions. In turn, the user feedback can be positive or negative. It is positive if the user accepts the assistance provided by the agent, that is if the assistance action executed by the agent was the one the user expected. Otherwise, the feedback is negative. The implicit feedback can adopt different forms such as the agent executing an action different from the one suggested by the agent, the user executing the task the agent suggested him, asking the agent for another solution, or not dealing with the problem at hand.

Once the agent has gathered feedback for its assistance, it has to evaluate if the assistance experience was a success, a failure or neither a success nor a failure, i.e. it is undefined. An interaction experience is evaluated as a success if the type of assistance the agent provided was the one the user expected. An interaction experience is evaluated as a failure if the assistance provided was of the wrong type.

\(^1\)In this Chapter we only consider the attributes relevant to the $WATSON$ algorithm.
Finally, if the agent cannot tell whether the assistance experience is a success or a failure, because the agent could not gather implicit feedback and the user did not provide explicit feedback, the interaction is rated as undefined. In this latter case, the agent might have distinguished which the expected action was but with very low certainty.

For example, if we consider an agent assisting a user of a calendar management system, an assistance experience could be the following. The user is scheduling a new event: a meeting to discuss the evolution of project A with his employees Johnson, Taylor and Dean. The event is being scheduled for Friday at 5 p.m. at the user’s office. The agent has learned by observing the user’s actions and schedules that Mr. Dean will probably disagree about the meeting date and time because he never schedules meetings on Friday evenings. Thus, it decides to warn the user about this problem. In reply to this warning, the user asks the agent to suggest him another date for the event. In this example, the different parts of the assistance experience are:

- Sit = {(type, new-event), (event-type, business meeting), (organizer, user), (participants, [Johnson, Taylor, Dean]), (topic, project A evolution), (date, Friday), (time, 5 p.m.), (place, user’s office)}
- Act = {(type, warning), (message, Johnson does not like meetings on Friday evenings)}
- UF = {(type, explicit), (action, asks for a suggestion)}
- E = {(type, failure), (certainty, 1.00)} (suggestion instead of warning)

Figure 4.2 shows part of a file containing a set of user-agent interaction experiences in the calendar management domain. This file constitutes an input to the WATSON algorithm\(^2\).

The output of our algorithm is a set of facts representing the agent’s beliefs about the assistance action a user requires in a given situation. These facts may adopt one of the following forms: “in problem situation Sit the user requires a warning W”, “in situation Sit the user requires a suggestion S”, “in situation Sit the user wants the agent to execute action A” or “the user does not want assistance (in situation Sit)”. Each fact F is accompanied by a certainty degree Cer(F) which indicates how certain the agent is about this fact. Facts constitute part of the user interaction profile which was introduced in Chapter 3. The following sections describe how we obtain these facts from the set of user-agent interaction experiences.

\(^2\)The format of the file, arff, is quite known in the machine learning community since it is the one used by WEKA, a widely used machine learning tool (http://www.cs.waikato.ac.nz/~ml/weka). See Appendix B.
4.3. WATSON Overview

As shown in Figure 4.3, the WATSON algorithm uses the information contained in a set of user-agent interaction experiences to obtain a user’s assistance requirements. Thus, when an interface agent has to decide among various assistance actions to deal with a problem situation, the agent uses the knowledge it has acquired about a user’s assistance requirements to choose the action it supposes the user expects in that particular instance of a given situation. Once the assistance is provided, the agent obtains (explicit and/or implicit) user feedback. This new interaction is recorded as an assistance experience, which will be used in the future to incrementally update the knowledge the agent has about the user.

In order to obtain a user’s assistance requirements from a set of user-agent interaction experiences the WATSON algorithm first generates a set of hypotheses.
Figure 4.4: WATSON main steps

A hypothesis expresses the agent’s belief that the user requires a certain type of assistance in a given situation. A hypothesis $H$ expresses that whenever situation $Sit$ occurs, the user will require an assistance action $Act$ with a certainty degree of $Cer(H)$. To validate a hypothesis, the algorithm gathers positive evidence supporting it and negative evidence rejecting it. Both types of evidence are used to compute the certainty degree of the hypothesis. If this certainty degree is greater than a threshold $\delta$, then the hypothesis is considered as valid and it is turned into a fact representing a user assistance requirement. Otherwise, it is discarded. These steps are shown in Figure 4.4.

The WATSON algorithm uses association rules to generate hypotheses about a user’s assistance requirements. Association rules imply an association relationship among a set of items in a given domain. Association rule mining finds interesting association or correlation relationships among a (large) set of data items. In this work, an interface agent uses association rules to discover the existing relationships between problem situations or situations of interest and the assistance actions a user requires to deal with them.

The association rules generated from the user-agent interaction experiences are automatically post-processed in order to derive useful hypotheses from them. Post-processing includes detecting the most interesting rules according to our goals, eliminating redundant and insignificant rules, pruning out contradictory weak rules, and
summarizing the information in order to formulate the hypotheses more easily. As we have said before, once a hypothesis is formulated, the algorithm looks for positive evidence supporting the hypothesis and negative evidence rejecting it in order to validate it. The certainty degree of the hypothesis is computed taking into account both the positive and the negative evidence. This calculus is done by using metrics from association rule discovery, as we will explain later. Finally, facts are generated from the set of highly supported hypotheses. Algorithm 5 shows the main steps our algorithm involves when using association rules as machine learning technique. The following sections explain in detail how we perform each of them.

Algorithm 5 WATSON Overview

Input: A set $Ex$ of user-agent interaction experiences $Ex_i =<Sit_i, Act_i, UF_i, E_i, date_i>$ (where $Sit$: situation, $Act$: assistance action, $UF$: user feedback, $E$: evaluation, $date$: date of the experience)

Output: A set $F$ of facts and a set $H$ of hypotheses representing the user’s assistance requirements

1. $F \leftarrow \varnothing$
2. $H \leftarrow \varnothing$
3. $AR \leftarrow$ Call association rule mining algorithm with $Ex$
4. $AR_1 \leftarrow$ Filter out uninteresting rules from $AR$
5. $AR_2 \leftarrow$ Eliminate redundant and insignificant rules from $AR_1$
6. $AR_3 \leftarrow$ Eliminate contradictory weak rules from $AR_2$
7. $AR_4 \leftarrow$ Summarize the discovered rules contained in $AR_3$
8. $H \leftarrow$ Transform rules in $AR_4$ into hypotheses
9. for $i = 1$ to size of $H$ do
10. Find evidence for $(E^+)$ and against $(E^-)$ $H_i$
11. $Cer(H_i) \leftarrow$ compute certainty degree of $H$ considering $(E^+)$ and $(E^-)$
12. if $Cer(H_i) \geq \delta$ then
13. $F \leftarrow F \cup H$
14. end if
15. end for

4.3.1 Mining Association Rules from User-Agent Interaction Experiences

The WATSON algorithm uses association rules to discover knowledge about a user’s assistance requirements from a set of user-agent interaction experiences. Association rules enable us to rapidly discover associations between different situations and the expected agent actions. An association rule is a rule which implies certain association relationship among a set of objects in a given domain, such as they occur together or one implies the other. Association discovery finds rules about items that
appear together in an event, called transactions, such as a purchase transaction or a user-agent interaction experience. Association rule mining is commonly stated as follows [Agrawal and Srikant, 1994]: Let \( I = i_1, \ldots, i_n \) be a set of items and \( D \) be a set of data cases, such as the file shown in Figure 4.2. Each data case or transaction consists of a subset \( X \) of items in \( I \). An association rule is an implication of the form \( X \rightarrow Y \), where \( X \subseteq I \), \( Y \subseteq I \) and \( X \cap Y = \emptyset \). \( X \) is the antecedent of the rule and \( Y \) is the consequent. The support of a rule \( X \rightarrow Y \) is the probability of attribute sets \( X \) and \( Y \) occurring together in the same transaction.

Given an itemset \( X \) (that is, \( X \subseteq I \)) and a transaction \( T \), we say that \( T \) contains \( X \) if and only if \( X \subseteq T \). The support count of an itemset \( X \) is defined to be \( \sigma_x \), which is the number of transactions in \( D \) that contain \( X \). We say that an itemset \( X \) is large, with respect to a support threshold of \( z\% \), if \( \sigma_x \geq |D| \cdot z\% \) where \( |D| \) is the total number of transactions in the database \( D \). A rule \( X \rightarrow Y \) has support \( s \) in \( D \) if \( s\% \) of the data cases in \( D \) contains \( X \cap Y \), that is \( \sigma_{X \cup Y} = |D| \cdot s\% \). If there are \( n \) total transactions in the database, and \( X \) and \( Y \) occur together in \( m \) of them, then the support of the rule \( X \rightarrow Y \) is \( m/n \). The rule \( X \rightarrow Y \) holds in \( D \) with confidence \( c \) if \( c\% \) of data cases in \( D \) that contain \( X \) also contain \( Y \). The confidence of rule \( X \rightarrow Y \) is defined as the probability of occurrence of \( X \) and \( Y \) together in all transactions in which \( X \) already occurs. If there are \( s \) transactions in which \( X \) occurs, and in exactly \( t \) of them \( X \) and \( Y \) occur together, then the confidence of the rule is \( t/s \).

An example of an association rule is: “30% of transactions in a supermarket that contain beer also contain diapers; 2% of all transactions contain both items.” In this example, 30% is the confidence of the rule and 2% the support of the rule.

Given a transaction database \( D \), the problem of mining association rules is to find all association rules that satisfy:

1. minimum support (called \( \text{minsup} \)) and,

2. minimum confidence (called \( \text{minconf} \)).

\( \text{minsup} \) is an input parameter to the algorithm for generating association rules. It defines the support threshold, and rules that have greater support than \( \text{minsup} \) are the only ones that are generated. If a rule has low support then it might have arisen by chance. There is not enough evidence to draw out a conclusion. \( \text{minconf} \) is an input parameter that defines the minimum level of confidence that a rule must possess. If a rule has low confidence then it is likely that there is no relationship between the antecedent and the consequent.
4.3. WATSON Overview

The association rule mining problem can be reduced to the problem of finding all large itemsets for the \( \minsup \) threshold and generating then the association rules considering the \( \minconf \) threshold [Agrawal et al., 1993]. Thus, if \( s\% \) is the given support threshold, the mining problem is reduced to the problem of finding the set 
\[
L = \{ X | X \subseteq I \land \sigma_x \geq |D| \ast s\% \}. \]
We call an itemset that contains exactly \( k \) items a \( k \)-itemset and use the symbol \( L_k \) to denote the set of all \( k \)-itemsets in \( L \).

Association rules have been used for data mining purposes in many application domains. These applications include discovering affinities for market basket analysis and cross-marketing, catalog design, loss-leader analysis, health insurance, telecommunications and store layout and customer segmentation based on buying patterns. There has been a lot of research in the area of association rules and, as a result, there are various algorithms for discovering association rules in a database. The most popular is the Apriori algorithm [Agrawal and Srikant, 1994] - and its variations - which is the one we use to find our association rules.

In this work we are not concerned about how association rules are generated since we merely call the rule mining algorithm within our WATSON algorithm. Thus, we will only go briefly through the Apriori algorithm in order to understand some vital concepts that will be used later\(^3\). The Apriori algorithm finds out the large itemsets iteratively. In the \( k \)-th iteration, it finds out \( L_k \), that is the set of all large itemsets of size \( k \). To do this, in each iteration the algorithm first generates a set of candidate itemsets of size \( k \) denoted by \( C_k \). For the first iteration, \( C_1 \) contains all the 1-itemsets. For the subsequent iterations \( C_k \) is generated by applying the \( \text{apriori-gen} \) function [Agrawal and Srikant, 1994] on the set \( L_{k-1} \), i.e. the set of all large (k-1)-itemsets, which have been found in the previous iteration. The \( \text{apriori-gen} \) function generates all those \( k \)-itemsets \( X \) satisfying the condition that all (k-1) subsets of \( X \) be large (that is, \( X \in L_{k-1} \)). Since this is a necessary condition for a \( k \)-itemset to be large, the \( \text{apriori-gen} \) function guarantees that all large \( k \)-itemsets are included in \( C_k \), that is \( C_k \supseteq L_k \). Having found the set of candidates \( C_k \), Apriori scans the database \( D \) in order to obtain the support counts \( \sigma_x \) for all \( X \in C_k \). Next, all candidates \( X \in C_k \) with support count \( \sigma_x \geq |D| \ast s\% \) are added to the set \( L_k \).

This completes one iteration. The iterations go on until \( L_j \) is empty for some \( j \). The set of all large itemsets \( L \) is then the union \( \bigcup_{1 \leq k \leq j} L_k \).

The WATSON algorithm uses a subset of the association rules generated by the Apriori algorithm, since we are interested only in some particular association

\(^3\)If readers want more information about association rule mining they will find a brief description in Appendix C, or they can read [Agrawal and Srikant, 1994], [Srikant et al., 1997a], [Bayardo and Agrawal, 1999].
rules generated from a set of user-agent interaction experiences. The rules we are interested in are those association rules of the form "problem description, assistance action—user feedback, evaluation" having appropriate support and confidence values. In general, the values of the confidence and support thresholds are specified by the developer or the user of the association rule mining algorithm. In this work, determining the most appropriate values of minconf and minsup is part of the WATSON algorithm.

4.3.2 Analyzing the interestingness of rules

Association rule mining algorithms tend to produce a huge number of rules, most of which are not relevant to the user. Due to the large number of rules it is very difficult for the user to analyze them manually in order to identify those truly interesting ones. The interestingness of a rule can be measured using objective measures and subjective measures. Objective measures involve analyzing the structure of the rule, the predictive performance and the statistical significance. However, as it is noted in [Piatetsky-Shapiro and Matheus, 1994], objective measures are insufficient for determining the interestingness of a discovered rule and subjective measures are needed.

Many works have dealt with the problem of finding objective interesting rules, such as [Liu et al., 2000], [Liu and Hsu, 1996], [Bayardo and Agrawal, 1999] and [Hilderman and Hamilton, 1999]. These works have defined a variety of metrics to determine the objective interestingness of a rule. Among them are confidence and support [Agrawal and Srikant, 1994], gain [Agrawal et al., 1996], chi-squared value [Morishita, 1998], entropy gain [Morimoto et al., 1998], gini [Morimoto et al., 1998], laplace [Webb, 1995] [Clark and Boswell, 1991], lift (also known as interest [Brin et al., 1997] or strength [Dhar and Tuzhilin, 1993]), and conviction [Brin et al., 1997].

However, although objective measures provide us important information about the generated rules, sometimes people is interested in other aspects of rules. For example, one may only want rules that contain a specific item or rules that contain one item from a subset of items, no matter what objective metrics tell him. Subjective measures often require some domain knowledge that must be provided by the user or a domain expert. Generally, a combination of the two types of metrics is desired.

In our work, we are interested in those association rules of the form "problem description, assistance action—user feedback, evaluation", having appropriate support and confidence values. Other combinations of items are irrelevant since we are
trying to discover the relationships between a problem situation and the assistance actions that received a positive user feedback and between problem situations and assistance actions that received negative user feedback. These types of rules reveal, directly in the former case and indirectly in the latter case, the user’s assistance requirements.

We can use an intuitive approach [Klemmentinen et al., 1994] to select those rules we are interested in. Relevant (and also irrelevant) classes of rules can be specified with templates. Templates describe a set of rules by specifying which attributes occur in the antecedent and which attributes occur in the consequent of a rule.

We (or a domain expert) can first classify attributes into an is-a hierarchy or a taxonomy, since we might want rules containing attributes of a given class. In our example domain of calendar management, the events can be divided into different types: meetings, dinners, parties, classes or courses, appointments with the doctor, and others. In turn, we can have different types of meetings, such as business meetings or school meetings, for example. Thus, we have the following generalizations:

- Business, School $\subseteq$ Meeting $\subseteq$ EventType
- Friends, Family $\subseteq$ Party $\subseteq$ EventType

We can also express these generalizations by using taxonomies, which are often used to describe the items in a database and can be used to specify the structure of the desired association rules. An example of a taxonomy is shown in Figure 4.5. This taxonomy states that a meeting with Mr. Jones is a business meeting, a business meeting is a meeting, and a meeting is a type of event. When taxonomies are present, users are usually interested in generating rules that span different levels of the taxonomy. Users may want those rules containing descendants of a given item, for example.

A template is an expression of the form:

$$A_1, \ldots, A_k \rightarrow A_{k+1}, \ldots, A_n$$

where each $A_i$ is either an attribute name, a class name, or an expression $C+$ and $C*$, which correspond to one or more and zero or more instances of the class C,
respectively. A rule $B_1, ..., B_h \rightarrow B_{h+1}, ..., B_m$ matches the pattern if the rule can be considered to be an instance of the pattern.

Our agents will be interested in those rules where the Problem Description and the Assistance Action are on the left side and the User Feedback and the Evaluation are on the right side. All of them are classes, that is non-leaf nodes in the taxonomy.

The template can be simply expressed as: Problem Description$^+$, Assistance Action$^+$ $\rightarrow$ User Feedback$^*$, Evaluation$^+$. Although we use the symbol $+$ we will not have more than one problem description, assistance action or evaluation in a rule. The same occurs with the symbol $^*$, we will have one or zero user feedback item per rule. The different components of the template may have taxonomies describing them. For example, each situation or problem description may have several taxonomies for the different attributes they involve.

In the previous paragraphs we saw that we can apply constraints on items as a post-processing step after generating association rules. However, integrating them into the mining algorithm can dramatically reduce the execution time [Srikant et al., 1997b]. According to this approach we can build constraints over the items and include these constraints into the association discovery algorithm. Constraints are boolean expressions over the presence or absence of items in the rules. These boolean expressions can also be expressed as templates.

### 4.3.3 Pruning the discovered rules

Once we have filtered out those rules that are not interesting or relevant for us, we will still have many rules to process, some of them redundant or insignificant. Many discovered associations are redundant or minor variations of others. Thus, those spurious and insignificant rules should be removed. For example, consider the following rules:

**R1:** Sit($(Event\ Type = doctor))$, $(Act = warning)$ $\rightarrow$(UF = ask for suggestion), $(Ev = failure)$ [sup: 0.40, conf: 0.825]

**R2:** Sit($(Event\ Type = doctor), (Event\ Priority = high))$, $(Act = warning)$ $\rightarrow$(UF = ask for suggestion), $(Ev = failure)$ [sup: 0.40, conf: 0.775]

If we know R1, then R2 is insignificant because it gives little extra information. Thus, it should be pruned. R1 is more general and simple. General and simple rules are preferred.

In this work we will follow the approach proposed by [Shah et al., 1999] to prune out redundant rules. We will consider the following set of pruning rules, which are a small subset of the rules defined by [Shah et al., 1999] (pruning rules 1 and 4 of those defined in this paper). Consider two rules $R1$ and $R2$: 
4.3. **WATSON** Overview

- if $R_1 = A \rightarrow C; R_2 = A, B \rightarrow C$, and either both rules are positive or negative with similar strength, then $R_2$ is redundant

- if $R_1 = A \rightarrow B; R_2 = A, B, C$, then $R_1$ is redundant

where a positive rule, denoted by $A \rightarrow^+ B$, is a rule where the presence of attribute $A$ is found to increase the probability of $B$’s occurrence significantly. Formally, this means that for a given user-defined coefficient $P > 1$, $Pr(B|A) > P * Pr(B)$ should be satisfied. A negative rule, denoted by $A \rightarrow^- B$, is a rule where for a user-defined coefficient $N > 1$, $Pr(B) > N * Pr(B|A)$. The strength of an association rule is the confidence of the rule, that is $Pr(B|A)$. Two rules are of similar strength if for a small predefined value $1 > \varepsilon_1 > 0$, $|strength(rule_1) - strength(rule_2)| < \varepsilon_1$, that is $|confidence(rule_1) - confidence(rule_2)| < \varepsilon_1$.

The rest of the rules defined in [Shah et al., 1999] are not applicable to our work, since they involve transitive dependencies, i.e. the consequent of a rule is the antecedent of another rule, or other relationships that cannot arise in our rule sets.

An example of the first pruning rule is the one we have presented before in this section with $\varepsilon_1 = 0.06$, $N = 2$, $P = 2$, and 100 rules. The following is an example of the second pruning rule, where the redundant rule is $R_1$:

- $R_1$: $Sit((EventType = doctor)), (Act = warning) \rightarrow (UF = ok)$ [sup: 0.50, conf: 0.86]

- $R_2$: $Sit((EventType = doctor)), (Act = warning) \rightarrow (UF = ok), (Ev = success)$ [sup: 0.52, conf: 0.84]

In addition, we have to analyze certain combinations of attributes in our particular domain in order to determine if two rules are telling us the same thing. The following list shows some examples of redundancy in rules found in our domain, provided that the rules involved refer to the same problem situation:

- warning, failure, OK and ask for solution $\equiv$ suggestion, success

- suggestion, failure, solve on user’s behalf $\equiv$ action, success

- action, failure, just suggest next time $\equiv$ suggestion, success

- action, failure, just warn next time $\equiv$ warning, success

The first equivalence, for instance, expresses that a rule indicating that for a given problem situation an agent warning received a negative user feedback because the user expected a solution, is equivalent to a rule indicating that, for the same situation, a suggestion received a positive feedback. If these two rules have similar
4.3. WATSON Overview

confidence values, we can eliminate the first rule and keep the second one. Two rules are of similar confidence if for a small predefined value \(1 > \varepsilon_1 > 0\), \(|\text{confidence}(R1) - \text{confidence}(R2)| < \varepsilon_1\).

Similarly, the last equivalence indicates that a rule expressing that for a certain situation the user rejected an agent action arguing that a suggestion would be enough is equivalent to a rule indicating that a suggestion about how to deal with the situation was a success. In most cases, we need the user feedback and not just the evaluation to determine if two rules are equivalent. For example, if we do not have the user feedback in the two last equivalences, we will not be able to tell if a negative evaluation for an agent action means that the user wanted a warning or if he wanted a suggestion.

As regards redundancy analysis, if two rules \(R1\) and \(R2\) provide the same information but their confidence values are very different, \(|\text{confidence}(R1) - \text{confidence}(R2)| > \varepsilon_1\), we have to keep the two rules.

As well as analyzing redundant rules, we have to check if there are any contradictory rules. We define that two rules are contradictory if, for the same situation, they express that the user wants two different assistance actions. The following list shows some examples of contradictory rules in our domain, provided that they refer to the same problem situation:

- suggestion, success \(\equiv\) warning, success
- warning, failure, OK and ask for solution \(\equiv\) suggestion, failure
- suggestion, failure, do something \(\equiv\) action, failure
- action, just a suggestion, failure \(\equiv\) suggestion, failure
- action, just a warning, failure \(\equiv\) warning, failure

If two contradictory rules \(R1\) and \(R2\) have very different confidence values, for example \(R1\) has a confidence of 80% and \(R2\) has a confidence value of 20%, then we will eliminate \(R2\) and keep \(R1\). More formally, we will eliminate \(R1\) if for a predefined value \(1 > \varepsilon_2 > 0\), \(|\text{confidence}(R1) - \text{confidence}(R2)| > \varepsilon_2\), and \(\text{confidence}(R1) < \text{confidence}(R2)\). However, if the confidence values are similar, that is \(|\text{confidence}(R1) - \text{confidence}(R2)| < \varepsilon_2\), we will keep the two rules.

Finally, to avoid considering obsolete assistance experiences, our algorithm deletes old transactions from the database. We consider that an interaction experience is obsolete when its date is older than \(\theta\) months, for a given threshold \(\theta\) currently set to six.
4.3. **WATSON** Overview

![Diagram of WATSON](image.png)

**Figure 4.6:** Formulating hypotheses from user-agent interaction experiences

The rules that survive all the pruning processes described before are those **WATSON** considers to build the hypotheses. Figure 4.6 shows, as an example, how a set of hypotheses is derived from a set of user-agent interactions.

### 4.3.4 Building facts from hypotheses

The association rules that have survived the pruning processes described in the previous section are those the **WATSON** algorithm uses to build hypotheses about a user’s assistance requirements. Each single hypothesis is obtained from a set of association rules that are related because they refer to the same problem situation but are somewhat different: a “main” association rule; some redundant association rules with regards to the main rule, which could not be pruned out because they did not fulfill the similar confidence restriction; and a set of contradictory rules, which could be not pruned away because they did not meet the different confidence requirement. The main rule is chosen by selecting from the rule set the rule that has the greatest support value, whose antecedent is the most general and whose consequent is the most specific.

For example, consider the following set of rules:
4.3. WATSON Overview

\[ R1: \text{Sit}(\text{EventType} = \text{doctor}), (\text{Act} = \text{warning}) \rightarrow (\text{UF} = \text{OK}) \text{ [sup: 0.30, conf: 0.70]} \]

\[ R2: \text{Sit}(\text{EventType} = \text{doctor}), (\text{EventDay} = \text{Monday}), (\text{Act} = \text{warning}) \rightarrow (\text{UF} = \text{OK}) \text{ [sup: 0.40, conf: 0.25]} \]

\[ R3: \text{Sit}(\text{EventType} = \text{doctor}), (\text{Act} = \text{warning}) \rightarrow (\text{UF} = \text{wrong}), (\text{Ev} = \text{failure}) \text{ [sup: 0.20, conf: 0.50]} \]

\[ R4: \text{Sit}(\text{EventType} = \text{doctor}), (\text{Act} = \text{warning}) \rightarrow (\text{UF} = \text{OK}), (\text{Ev} = \text{success}) \text{ [sup: 0.40, conf: 0.72]} \]

In this rule set rule R4 is the main rule, rules R1 and R2 are redundant rules that could not be eliminated, and R3 is a contradictory rule that could not be eliminated, considering \( N = 2, P = 2, e_2 = 0.8, \) and \( e_1 = 0.06. \) The hypothesis will be built with the main rule and the other rules are taken into account to compute the certainty degree of the hypothesis.

Once the WATSON algorithm has formulated a set of hypotheses it has to validate them. Our algorithm tries to prove a hypothesis by analyzing the evidence for and against it. The evidence for and against a given hypothesis can be obtained by analyzing the association rules belonging to the rule set of the rule originating the hypothesis. Given an association rule that originates a hypothesis, we define as positive evidence those association rules that were considered as redundant during the pruning steps but were not eliminated. Formally, given an association rule \( \text{Sit}_1, \text{Act} \rightarrow Y, \) a rule \( \text{Sit}_2, \text{Act} \rightarrow Y \) where \( \text{Sit}_2 \supset \text{Sit}_1 \) is considered a positive evidence. Similarly, given an association rule \( \text{Sit}, \text{Act} \rightarrow \text{Ev}, \text{UF}, \) a rule \( \text{Sit}, \text{Act} \rightarrow \text{Ev} \) is considered a positive evidence. Combinations of the two previous situations can be also considered as positive evidence.

We define as negative evidence those association rules that were considered as contradictory rules during the pruning steps but were not eliminated. Given an association rule \( \text{Sit}, \text{Act} \rightarrow Y, \) a rule \( \text{Sit}, \text{Act} \rightarrow Z, \) where \( Y \neq Z, \) is considered a negative evidence. Similarly, an association rule \( \text{Sit}, \text{Act}_1 \rightarrow Z \) is considered as a negative evidence for the rule \( \text{Sit}, \text{Act}_2 \rightarrow Z, \) given that \( \text{Act}_1 \neq \text{Act}_2. \) Finally, given the rule \( \text{Sit}, \text{Act} \rightarrow Z, \) the rule \( \text{Sit}, \text{Act} \rightarrow Y \) where \( Z \cap Y = \emptyset \) is a negative evidence.

The certainty degree of a hypothesis \( H \) is computed as a function of the supports of the rule originating the hypothesis and the rules considered as positive and negative evidence of \( H. \) We chose this method to compute the certainty degree of a hypothesis because the support of a rule gives us an indication of the usefulness of the rule, that is the amount of interaction experiences that contain the items involved in the rule. Thus, we take into account not only the number of experiences
that contain the items in the main rule, but also the experiences containing the positive and the negative evidence. The positive evidence contains those similar situations with the same associated assistance action. Therefore, we add the support of these rules to the support of the main rule to have a picture of the usefulness of the hypothesis. Then, we subtract the support of the negative evidence because we want to reflect that those rules indicate that in those situations our hypothesis is not useful.

The function we use to compute certainty degrees is shown in Equation 4.1, where $\alpha$, $\beta$ and $\gamma$ are the weights of the terms in the equation, $Sup(AR)$ is the support of the rule originating $H$, $Sup(E^+)$ is the support of the rules being positive evidence, $Sup(E^-)$ is the support of the rules being negative evidence, $Sup(E)$ is the support value of an association rule taken as evidence (positive or negative), $r$ is the amount of positive evidence and $t$ is the amount of negative evidence. We set $\alpha=0.7$, $\beta=0.15$ and $\gamma=0.15$ because we consider that the main rule is more important in this calculus than the negative and positive evidence, and we consider that both types of evidence are equally important.

$$Cer(H) = \alpha Sup(AR) + \beta \frac{\sum_{k=1}^{r} Sup(E^+)}{\sum_{k=1}^{r+t} Sup(E)} - \gamma \frac{\sum_{k=1}^{t} Sup(E^-)}{\sum_{k=1}^{r+t} Sup(E)} \quad (4.1)$$

If the certainty degree of a hypothesis is greater than a threshold value $\delta$ the hypothesis is turned into a fact, otherwise it is discarded.

For example, for the rule set presented before, the certainty degree of the hypothesis is computed as follows:

$$Cer(H) = 0.7 \times 0.4 + 0.15 \times \frac{(0.30+0.40)}{(0.30+0.20+0.40)} - 0.15 \times \frac{0.20}{(0.30+0.20+0.40)} = 0.28 + 0.1167 - 0.033 = 0.3637$$

To determine the action in a hypothesis, we have to analyze the evaluation and the user feedback of the main rule. If the evaluation is known and it is a success, then the hypotheses is built straightforward by associating the situation with the corresponding assistance action. If the evaluation is known and it is a failure, then the algorithm can determine from the user feedback which the correct assistance action is. We prefer “positive” hypotheses, that is hypotheses implying success. Finally, if the evaluation is undefined, it is most likely that the evaluation is actually a failure but the agent cannot tell which the required action is because it does not have enough information. It can assume some action with a certainty degree, and confirm this when more information is available. Algorithm 6 shows the different steps our WATSON algorithm involves.
Algorithm 6 \textit{WATSON} learning algorithm

\textbf{Input:} A set $E_x$ of user-agent interaction experiences $E_{x_i} = <Sit_i, Act_i, UF_i, E_i, date_i>$ (where $Sit$: problem description, $Act$: assistance action, $UF$: user feedback, $E$: evaluation, $date$: interaction date)

\textbf{Output:} A set $F$ of facts and a set $H$ of hypotheses about the user’s assistance requirements

1: minsup ← Obtain value for minsup (explained later in Chapter 7)
2: minconf ← 0.8
3: Call association rule mining algorithm (Apriori) with $\text{minconf}, \text{minsup}$ and $E_x$
   \{Post-process association rules\}
4: $AR_1$ ← Select from $AR$ those association rules $X \rightarrow Y$ where $X = <Sit, Act>$ and $Y = <UF, E >$ or $Y = <E >$  
5: $AR_2$ ← Filter out uninteresting rules from $AR_1$
6: $AR_3$ ← Eliminate redundant and insignificant rules from $AR_2$
7: $AR_4$ ← Eliminate contradictory rules from $AR_3$
   \{Generate hypotheses\}
8: $ARSet_i$ ← Group similar rules in $AR_4$ (a rule and those redundant and contradictory rules that were not eliminated)
9: $ARSet ← \cup ARSet_i$
10: $n ← \text{size of } ARSet$
11: \textbf{for} $i = 1$ to $n$ \textbf{do}
12: \hspace{1em} Generate hypothesis $H_i$ from $ARSet_i$
13: \hspace{1em} $H ← H \cup H_i$
14: \textbf{end for}
   \{Validate hypotheses\}
15: \textbf{for} $i = 1$ to $n$ \textbf{do}
16: \hspace{1em} $E^+$ ← Obtain positive evidence for $H_i$ from $ARSet_i$
17: \hspace{1em} $E^-$ ← Obtain negative evidence for $H_i$ from $ARSet_i$
18: \hspace{1em} Compute certainty $\text{Cer}(H_i)$ taking into account $E^+$ and $E^-$
19: \hspace{1em} \textbf{if} $\text{Cer}(H_i) \geq \delta$ \textbf{then}
20: \hspace{2em} $F ← F \cup H_i$
21: \hspace{1em} \textbf{end if}
22: \textbf{end for}
4.4 Incremental Learning and Profiling

In this section we will concentrate on the problem of maintaining the discovered knowledge about a user’s assistance requirements. The interaction experience database is not a static database, because updates are constantly being applied to it. On the one hand, new interaction experiences are added since the agent keeps observing a user’s behavior. On the other hand, old experiences are deleted from the database to save storage space and because they are out of interest or they become obsolete. Because of these update activities, the user-agent interaction experience database keeps on changing. In consequence, new hypotheses about a user’s behavior may appear and some of the learned hypotheses may become invalid. Thus, the maintenance of the knowledge the agent has acquired is an important problem.

One possible approach to the update problem is to re-run the WATSON algorithm on the whole updated database. This approach, though simple, has some obvious disadvantages. All the computation done initially at finding a user’s assistance requirements is wasted, and they have to be obtained from the scratch.

Since our agent’s hypotheses about a user’s assistance requirements are obtained from the association rules extracted from the interaction experiences, we will address the maintenance problem from the association rule point of view. As the database changes, new association rules may appear and at the same time, some existing association rules may become invalid. The problem of maintaining the discovered association rules has been deeply studied and several algorithms have been proposed [Veloso et al., 2001, Veloso et al., 2002, Ng and Lam, 1999, Ayan et al., 1999, Cheung et al., 1996, Cheung et al., 1997, Thomas et al., 1997]. From these algorithms we have chosen the FUP2 algorithm [Cheung et al., 1997] to update association rules within the WATSON algorithm. This algorithm can update the discovered association rules when new transactions are added to, deleted from, or modified in the database. We have selected this algorithm because its framework is similar to the Apriori algorithm, while the other algorithms use other association rule mining algorithms. A disadvantage of the FUP2 algorithm is that it requires $O(k)$ database scans, where $k$ is the size of the large frequent itemset. However, in our domain $k$ is a number between 2 and 20.

4.4.1 Update of association rules

Assuming that the two thresholds, minimum support and minimum confidence, do not change, there are several important characteristics in the update problem [Cheung et al., 1996].
4.4. Incremental Learning and Profiling

Figure 4.7: Definitions of $D$, $D'$, $D-$, $\triangle-$ and $\triangle+$

1. The update problem can be reduced to finding the new set of large itemsets. After that, the new association rules can be computed from the new large itemsets.

2. An old large itemset has the potential to become small in the updated database.

3. Similarly, an old small itemset could become large in the new database.

4. In order to find the new large itemsets, all the records in the updated database including those from the original database, have to be checked against every candidate set.

A naive approach to compute the new set of association rules would be to re-execute the Apriori algorithm on the updated database. However, this process is not efficient since it is memoryless (that is, it ignores the already discovered knowledge), essentially duplicating part of the work that has already been done.

After some activities, old transactions are deleted from the database $D$ and new transactions are added. Let $\triangle-$ be the set of deleted transactions and $\triangle+$ the set of newly added transactions. We assume that $\triangle- \subseteq D$. We denote the updated database $D'$. Note that $D' = (D - \triangle-) \cup \triangle+$. We denote the set of unchanged transactions by $D- = D - \triangle- = D' - \triangle+$. The relationships between these data sets are illustrated in Figure 4.7.

As defined in Section 4.3.1, we use $\sigma_x$ to denote the support count of an itemset $X$ in the original database $D$. The set of large itemsets in $D$ is $L$ and $L_k$ is the set of k-itemsets in $L$. The authors in [Cheung et al., 1997] define $\sigma'_x$ to be the new support count of an itemset $X$ in the updated database $D'$, and $L'$ to be the set of large itemsets in $D'$. $L'_k$ is the set of k-itemsets in $L'$. They further defined $\delta_X^+$ to be the support count of itemset $X$ in the database $\triangle+$ and $\delta_X^-$ to be that of $\triangle-$. These definitions are summarized in Table 4.1. The change of support count of itemset $X$ as a result of the update activities is defined as $\delta_X = \delta_X^+ - \delta_X^-$. Thus, we have: $\sigma'_x = \sigma_x + \delta_X^+ - \delta_X^- = \sigma_x + \delta_X$.

For the update problem, $L$ and $\sigma_x \forall X \in L$ are data available as the result of a previous mining operation done on the old database $D$. Thus, the update problem
### 4.4. Incremental Learning and Profiling

<table>
<thead>
<tr>
<th>database</th>
<th>support count of itemset $X$</th>
<th>large k-itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta^+$</td>
<td>$\delta_X^+$</td>
<td>-</td>
</tr>
<tr>
<td>$D^+$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$\Delta^-$</td>
<td>$\delta_X^-$</td>
<td>-</td>
</tr>
<tr>
<td>$D = \Delta^- \cup D^+$</td>
<td>$\sigma_x$</td>
<td>$L_k$</td>
</tr>
<tr>
<td>$D' = D^- \cup \Delta^+$</td>
<td>$\sigma'_x$</td>
<td>$L'_k$</td>
</tr>
</tbody>
</table>

Table 4.1: Definitions of several symbols [Cheung et al., 1997]

is to find $L'$ and $\sigma'_x \forall X \in L'$ efficiently, given the knowledge of $D$, $D'$, $\Delta^+$, $\Delta^-$, $D^+$, $L$ and $\sigma_x \forall X \in L$.

We will go briefly through the basic FUP2 algorithm [Cheung et al., 1997] in order to understand how our incremental WATSON algorithm works\(^4\). Suppose that the association rules of an existing database have already been mined, and that after the mining, some updates have been performed on the database. The FUP2 algorithm, like Apriori, generates the large itemsets iteratively. In the $k$-th iteration, it first generates a set of candidate itemsets $C_k$ for the updated database $D'$. For the first iteration, $C_1$ is the set of all 1-itemsets. In the subsequent iterations, $C_k$ is generated by applying \textit{apriori-gen} (see Appendix C) on $L'_{k-1}$, the new large itemsets found in the previous iteration. The properties of \textit{apriori-gen} guarantees that $C_k \supseteq L'_k$.

Next, $C_k$ is divided into two partitions: $P_k = C_k \cap L_k$ and $Q_k = C_k - P_k$. In words, $P_k$ is the set of candidate itemsets that are previously large with regards to $D$. Conversely, $Q_k$ is the set of candidate itemsets that are previously small with regards to $D$. Our goal is to select those itemsets that are currently large regarding $D'$.

The candidates in the two partitions of $C_k$ are handled differently. For $X \in P_k \subseteq L_k$, we know the old support count $\sigma_x$ from the old mining results. So, we can scan $\Delta^+$ and $\Delta^-$ to find out $\delta_X$ and hence calculate the new support count $\sigma'_x = \sigma_x + \delta_X$. If this is greater than or equal to $|D'| \times s\%$ then $X$ is large, and hence it is added to the set $L'_k$. For each candidate $X \in Q_k$ we do not know its old support count $\sigma_x$, but we know that $\sigma_x < |D'| \times s\%$, since they were not large itemsets in the original database $D$. Therefore, such a candidate can be large only if $\delta_X \leq (|\Delta^+| - |\Delta^-|) \times s\%$, according to Lemma 4 in [Cheung et al., 1997]. Hence, in the scan of $\Delta^+$ and $\Delta^-$, we can obtain also the counts $\delta_X$ for the candidates $X \in Q_k$. For the remaining candidates $X \in Q_k$, we scan the unchanged transactions $D^-$ to find out their counts there, and then add this count to $\delta_X^+$ to get $\sigma'_x$. Those

\(^4\) Readers are referred to the cited paper for detailed information.
with \( \sigma' \geq |D'| \times \% \) are added to \( L'_k \). As a result, all the candidates in \( C_k = P_k \cup Q_k \) can be handled, and the candidate that are large in \( D' \) are added to \( L'_k \). Hence an iteration is completed, and the algorithm proceeds to the next iterations unless \( L'_k \) is empty. Some more detailed information about the FUP2 algorithm can be found in Appendix C.

### 4.4.2 When to update?

As pointed out in [Lee and Cheung, 1997], one of the main problems to maintain association rules is determining whether we update the mined association rules or not. We can apply FUP2 each time a new interaction experience is added to or deleted from the database. However, the overhead of such a run of FUP2 is very high. The total long-term overhead will be large because the algorithm is applied too frequently. On the other hand, if we always wait until a large amount of updates has accumulated before applying FUP2, we will not be able to discover the newest association rules quickly. This is not desirable, especially if we want to learn a user’s new assistance requirements or the changes in these requirements. Therefore, applying the FUP2 algorithm too infrequently we would not be able to discover the new association rules in time. To have the best results, the FUP2 algorithm should be run at suitable times.

We will adopt the approach proposed by [Lee and Cheung, 1997] to determine when to update our association rules. These authors have proposed an algorithm to estimate the difference between the association rules in a database before and after it is updated. The estimated difference can be used to determine whether we update the mined association rules or not. If the estimated difference is large, then it is time to update the discovered association rules in order to learn the new rules and discard the old ones. If the estimated difference is small, then the rules in the original database are still a good approximation for those in the updated database. Consequently, the user assistance requirements obtained from the association rules extracted from the original database are still a good approximation of the current user assistance requirements. Thus, we can accumulate more updates before actually updating the rules.

The DELI (acronym for Difference Estimation for Large Itemsets) algorithm proposed in [Lee and Cheung, 1997] uses a sampling technique to estimate the difference between the old and new association rules. This estimate is used as an indicator for whether the FUP2 algorithm should be applied to the database to accurately find out the new association rules. If the estimated difference is large enough (with respect to some user specified threshold), the algorithm signals the need of an up-
date operation, which can be accomplished by using the FUP2 algorithm. If the estimated difference is small, then we do not run FUP2 immediately and we can take the old rules as an approximation of the new rules. Hence, we wait until more changes are made to the database and then re-apply the DELI algorithm.

The difference between the old large itemsets $L$ and the new large itemsets $L'$ can be measured by the set symmetric difference between them. We use the notation $L \oplus L'$ to denote the symmetric difference between $L$ and $L'$. Note that

$$L \oplus L' = L' \ominus L = (L' - L) \cup (L - L')$$

Depending on the similarity between $L$ and $L'$, the size of $L \ominus L'$ can vary between 0 and $|L| + |L'|$. It is 0 when $L = L'$ and it is $|L| + |L'|$ when $L$ and $L'$ are disjoint. The smaller the size of $L \ominus L'$, the greater the similarity between $L$ and $L'$.

The ratio $\frac{|L \ominus L'|}{|L|}$ can be used as a relative measurement of the difference between $L$ and $L'$. Since in this estimation problem we do not know $L'$, we cannot calculate the above ratio. Thus, Lee and Cheung chose to use the ratio $\frac{|L \ominus L'|}{|L|}$ instead. We will use this ratio as a difference measure for the old and new large itemsets. Since $L$ is known to us from the results of the last mining, it remains to estimate the value of $L \ominus L'$. The estimation problem is to efficiently estimate the size $|L \ominus L'|$ without finding out $L'$, given the knowledge of $D$, $D'$, $\triangle +$, $\triangle -$, $D-$, $L$ and $\sigma_x \forall X \in L$.

The readers can find more details of the DELI algorithm in Appendix C. Algorithm 7 shows the main steps the incremental WATSON algorithm involves.

**Algorithm 7 Incremental WATSON**

**Input:** Original interaction experience database $DB$, deleted experiences $DB-$, new experiences $DB+$, current fact set $F$

**Output:** updated fact set $F_{new}$

1: updating $\leftarrow$ false
2: $F_{new} \leftarrow \emptyset$
3: updating $\leftarrow$ call DELI algorithm with $DB, DB-, DB+$ to determine whether to update or not
4: if updating then
5: $AR \leftarrow$ call FUP2 with $DB, DB-, DB+$
6: $F_{new} \leftarrow$ call WATSON post-processing steps with $AR$
7: else
8: $F_{new} \leftarrow F$
9: end if
10: return $F_{new}$
4.5 Example

In this section we will develop an example of the whole process the WATSON algorithm involves. For simplicity reasons, we will consider a small set of input data corresponding to the interaction between a user and the interface agent assisting him with his calendar management. The following paragraphs describe the characteristics of this user.

The user works in a software company as a project leader. He usually attends work meetings with his boss and with his subordinates to discuss the evolution of the projects he leads. These meetings take place generally on Monday mornings. The user goes to gym with a group of friends after 7 pm. The user generally goes out with his friends. They generally go to the cinema or have dinner together. The user also works as a teacher assistant of “Object Oriented Programming” at the University. He sometimes has meetings with the professor that teaches this subject. These meetings are usually on Friday afternoons.

Regarding the different types of assistance actions the agent provides, if there is an overlapping between a work meeting and another type of meeting, the user wants the agent to perform an action on the user’s behalf. Typically, it must cancel or reschedule this last meeting. If there is an overlapping between a meeting at home and a meeting taking place somewhere else, the agent should make a suggestion. If a meeting at the university overlaps with a meeting in some other place, the agent should solve the problem on the user’s behalf. Generally, it should cancel or reschedule this last event. If a work meeting overlaps with a meeting at the university, the agent should only make a warning.

Figure 4.8 shows the file containing the interaction experiences corresponding to the user we are analyzing. This file contains experiences in which the situation is an overlapping between two events. We are only considering 10 interaction experiences.

The first step of the WATSON algorithm consists of generating association rules from the user-agent interaction experiences. Figure 4.9 shows a subset of these association rules.

From all the association rules generated, the WATSON algorithms filters out those that are not relevant to its purposes according to the templates described in Section 4.3.3. The rules that survived the first filtering process are the following. The numbers between parentheses are used to differentiate the features of the two overlapping events (1 for the first event and 2 for the second event).
4.5. Example

Evaluation: overlapping

<table>
<thead>
<tr>
<th>Antecedent</th>
<th>Consequent</th>
</tr>
</thead>
<tbody>
<tr>
<td>(success, failure, addition)</td>
<td>(success, failure, addition)</td>
</tr>
<tr>
<td>(success, failure, addition)</td>
<td>(success, failure, addition)</td>
</tr>
</tbody>
</table>

Figure 4.8: Input file “overlapping”

<table>
<thead>
<tr>
<th>Antecedent</th>
<th>Consequent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. userreaction</td>
<td>2. userreaction</td>
</tr>
<tr>
<td>3. userreaction</td>
<td>4. userreaction</td>
</tr>
<tr>
<td>5. userreaction</td>
<td>6. userreaction</td>
</tr>
<tr>
<td>7. userreaction</td>
<td>8. userreaction</td>
</tr>
<tr>
<td>9. userreaction</td>
<td>10. userreaction</td>
</tr>
<tr>
<td>11. userreaction</td>
<td>12. userreaction</td>
</tr>
</tbody>
</table>

Figure 4.9: Association rules generated by Apriori
4.5. Example

1. topic (2) = other, participants (2) = friends, agent action = warning → user reaction = ok and solve, evaluation = failure; sup(0.4), conf(1)

2. event type (1) = work meeting, place (1) = office, agent action = warning → user reaction = ok and solve, evaluation = failure; sup(0.4), conf(1)

3. topic (2) = other, agent action = warning → user reaction = ok and solve, evaluation = failure; sup(0.4), conf(1)

4. topic (2) = other, participants (2) = friends, agent action = warning → evaluation = failure; sup(0.4), conf(1)

5. place (1) = office, agent action = warning → user reaction = ok and solve, evaluation = failure; sup(0.4), conf(1)

6. topic (1) = projects, agent action = warning → user reaction = ok and solve, evaluation = failure; sup(0.4), conf(1)

7. event type (1) = work meeting, agent action = warning → user reaction = ok and solve, evaluation = failure; sup(0.4), conf(1)

8. event type (1) = work meeting, place (1) = office, agent action = warning → evaluation = failure; sup(0.4), conf(1)

As the reader can observe, the amount of rules considered as interesting according to WATSON templates is small in relation to the total amount of rules generated (400 in this example). The next step is eliminating redundant rules according to Shah [Shah et al., 1999] pruning rules. In this example, all the rules survived this filtering step. The reasoning process is as follows. Initially, there are various pairs of rules that seemed to be redundant: rule 1 and rule 3; rule 1 and rule 4; rule 2 and rule 5; rule 2 and rule 7; and rule 2 and rule 8. However, when analyzing if the rules involved in each pair are either both negative or both positive, we find that none of them fulfills this requirement. Thus, no rule is considered redundant and no rule is deleted.

For example, consider the following pair of rules:

1. topic (2) = other, participants (2) = friends, agent action = warning → user reaction = ok and solve, evaluation = failure; sup(0.4), conf(1)

3. topic (2) = other, agent action = warning → user reaction = ok and solve, evaluation = failure; sup(0.4), conf(1)
4.5. Example

According to the notation used in Section 4.3.3, rule 1 can be written as \( A \rightarrow B \). The support of the consequent, \( Pr(B) \), is 0.8. The confidence of the rule, \( Pr(B/A) \), is 1. Since \( 1 > 0.8 \times 2 \) is not true, the rule is not positive. Similarly, \( 0.8 > 2 \times 1 \) is not true. Thus, the rule is not negative.

The same analysis is done with rule 3. For this rule, \( Pr(B) \) is 0.8 and \( Pr(B/A) \) is 1. Then, the calculus are the same than for rule 1 and the rule is neither negative nor positive. In consequence, the apparently redundant rule (rule 1) cannot be deleted.

Next, contradictory rules are searched. In this example, no contradictory rules are found. Thus, all the rules are considered to build the hypotheses. The components of each of the hypotheses generated by our algorithm are shown below.

1. **Main Rule:** topic (2) = other, agent action = warning → user reaction = ok and solve, evaluation = failure. **Positive Evidence:** topic (2) = other, participants (2) = friends, agent action = warning → user reaction = ok and solve, evaluation = failure; topic (2) = other, participants (2) = friends, agent action = warning → evaluation = failure. **Negative Evidence:** none. **Certainty:** 0.43

2. **Main Rule:** place (1) = office, agent action = warning → user reaction = ok and solve, evaluation = failure. **Positive Evidence:** event type (1) = work meeting, place (1) = office, agent action = warning → user reaction = ok and solve, evaluation = failure. **Negative Evidence:** none. **Certainty:** 0.43

3. **Main Rule:** event type (1) = work meeting, agent action = warning → user reaction = ok and solve, evaluation = failure. **Positive Evidence:** event type (1) = work meeting, place (1) = office, agent action = warning → user reaction = ok and solve, evaluation = failure. **Negative Evidence:** none. **Certainty:** 0.43

4. **Main Rule:** topic (1) = projects, agent action = warning → user reaction = ok and solve, evaluation = failure. **Positive Evidence:** none. **Negative Evidence:** none. **Certainty:** 0.28

For example, the certainty degree of hypothesis 1 is computed as follows.

\[
Cer(H) = 0.7 \times 0.4 + 0.15 \times \frac{(0.40+0.40)}{(0.40+0.40)} - 0.15 \times 0 = 0.28 + 0.15 = 0.43
\]

In this example, the certainty of the hypothesis is greater than the main rule support because the certainty calculus also includes the support of the positive evidence. This kind of adjustment is the purpose of including the positive and negative evidence in the certainty calculus.

The certainty value of hypothesis 4 is directly \( Cer(H4) = 0.7 \times 0.4 = 0.28 \), since there is no positive or negative evidence for or against it.
After summarizing the information contained in the hypotheses, the user profile is built. In this example, the type of situation is an overlapping between two events. Thus, the situation description in the profile items include the characteristics of one or two of the events involved. The user assistance requirements are the following:

- situation: topic = other; type of assistance: action
- situation: event type = work meeting; type of assistance: action
- situation: place = office; type of assistance: action
- situation: topic = projects; type of assistance: action

The second assistance requirement reflects directly one of the user preferences stated at the beginning of this section. The first assistance requirement does not give us much information, and it probably appears because of the association rule algorithm. Time will decide if this assistance requirement is valid or not. The third and fourth assistance requirements give us almost the same information as the second requirement.

Now, suppose that the agent has to decide how to assist the user in the following situation. The user is scheduling an event that overlaps with a work meeting taking place at the user’s office. The action the agent decides to execute according to this profile is an action on the user’s behalf. This will be done if the confidence on the action involved - the event rescheduling - is high enough. Otherwise, the agent will attempt to make a suggestion. Although the agent has only interacted 10 times with the user, he has determined the assistance action correctly, according to the user preferences stated at the beginning of this section.

Now, suppose that the user has interacted with his agent during some more time. Thus, the agent has to update the user profile according to this new information. The incremental algorithm has to be run. In this case we only have an increment database with the new user-agent interactions. We do not have experiences to delete. These interactions are shown in Figure 4.10.

First, the WATSON algorithms analyzes the need of an update. Some interactions in the increment also appeared in the original database, but there are many new interactions that change the information the agent has about the user. In consequence, the difference between the old and new transactions is significant. And so is the difference between the old large itemsets and the new large itemsets. Thus, it is necessary to compute the new association rules.

The new interesting association rules are the following. This set of rules is quite different from the original set of rules.
Figure 4.10: New user-agent interactions

1. topic (2) = other, participants (2) = friends, agent action = warning → evaluation = failure; sup(0.35), conf(1)

2. topic (2) = other, agent action = warning → evaluation = failure; sup(0.35), conf(1)

3. event type (1) = university meeting, place (1) = university, event type (2) = other, topic (2) = birthday, place (2) = home, agent action = warning → user reaction = ok, evaluation = success; sup(0.25), conf(1)

4. place (2) = university, event type (2) = other, topic (2) = birthday, place (2) = home, agent action = warning → user reaction = ok, evaluation = success; sup(0.25), conf(1)

5. event type (1) = university meeting, event type (2) = other, topic (2) = birthday, place (2) = home, agent action = warning → user reaction = ok, evaluation = success; sup(0.25), conf(1)

6. event type (1) = university meeting, place (1) = university, topic (2) = birthday, place (2) = home, agent action = warning → user reaction = ok, evaluation = success; sup(0.25), conf(1)

7. event type (1) = university meeting, place (1) = university, event type (2) = other, place (2) = home, agent action = warning → user reaction = ok, evaluation = success; sup(0.25), conf(1)
8. event type (1) = university meeting, place (1) = university, event type (2) = other, topic (2) = birthday, agent action = warning → user reaction = ok, evaluation = success; sup(0.25), conf(1)

During the second pruning step, rule 3 is eliminated since it is redundant with respect to rule 4. Rule 3 is positive since the support of the consequent is 0.4 (Pr(B)), the confidence of the rule is 1 (Pr(B/A)), and 1 > 0.4*2. The same occurs with rule 4.

The new set of hypotheses is listed below.

1. **Main Rule**: topic (2) = other, agent action = warning → evaluation = failure; sup(0.35), conf(1). **Positive Evidence**: topic (2) = other, participants (2) = friends, agent action = warning → evaluation = failure; sup(0.35), conf(1). **Negative Evidence**: none. **Certainty**: 0.39

2. **Main Rule**: place (1) = university, event type (2) = other, topic (2) = birthday, place (2) = home, agent action = warning → user reaction = ok, evaluation = success; sup(0.25), conf(1). **Positive Evidence**: none. **Negative Evidence**: none. **Certainty**: 0.175

3. **Main Rule**: event type (1) = university meeting, event type (2) = other, topic (2) = birthday, place (2) = home, agent action = warning → user reaction = ok, evaluation = success; sup(0.25), conf(1). **Positive Evidence**: none. **Negative Evidence**: none. **Certainty**: 0.175

4. **Main Rule**: event type (1) = university meeting, place (1) = university, topic (2) = birthday, place (2) = home, agent action = warning → user reaction = ok, evaluation = success; sup(0.25), conf(1). **Positive Evidence**: none. **Negative Evidence**: none. **Certainty**: 0.175

5. **Main Rule**: event type (1) = university meeting, place (1) = university, event type (2) = other, place (2) = home, agent action = warning → user reaction = ok, evaluation = success; sup(0.25), conf(1). **Positive Evidence**: none. **Negative Evidence**: none. **Certainty**: 0.175

6. **Main Rule**: event type (1) = university meeting, place (1) = university, event type (2) = other, topic (2) = birthday, agent action = warning → user reaction = ok, evaluation = success; sup(0.25), conf(1). **Positive Evidence**: none. **Negative Evidence**: none. **Certainty**: 0.175

Now, the user assistance requirements are the following:
4.6. Summary

In this chapter we have presented a solution to one of the problems interface agents have to face when they want to personalize the assistance they provide to their users. The WATSON algorithm enables an interface agent to learn when the user requires a warning, when he needs a suggestion, when he wants the agent to perform a task on his behalf and when the user does not need the agent’s assistance. The following chapter describes the IONWI algorithm, which learns when the user needs to be interrupted (and want to) by the agent to assist him.
Chapter 5

The *IONWI* algorithm: Does the user prefer an Interruption or a Notification Without Interruption?

In Chapter 4 we presented a solution to deal with one of the problems interface agents have to solve to personalize their interaction with users: discovering what type of assistance each user requires in different situations that may arise while he is working with a given application. In this Chapter we tackle another important problem interface agents have to deal with: discovering whether to interrupt the user or not to provide him assistance.

5.1 Introduction

As pointed out in [McCrickard and Chewar, 2003] and [Adamczyk and Bailey, 2004], one of the problems of agents is their incorrect estimates of a user’s task priorities, which makes information to be introduced at inappropriate times and with unsuitable presentation choices. Although agents are well-intentioned, they do not consider the impact an interruption has on a user. To solve this problem, when the agent detects a problem situation relevant to the user it has to correctly decide if it will send him a notification - with a warning or a suggestion - without interrupting his work, or if it will interrupt him. On the one hand, the user can choose between paying attention to the notification or not, and he can continue to work in this latter case. On the other hand, he has to pay attention to what the agent wants to tell him if it interrupts him abruptly. We consider that the agent has to base its decision on: the relevance and the urgency the situation has for the user; the user’s goals; the user’s current tasks; how tolerant the user is of interruptions; and when the user does not
5.1. Introduction

![Diagram of Learning and Decision Making Algorithms]

Figure 5.1: Learning and decision making algorithms

want to be interrupted no matter how important the message is.

In this Chapter we present the IONWI learning algorithm (acronym for Interruption Or Notification Without Interruption), which enables an interface agent to learn a user’s interruption needs and preferences in different contexts. The goal of our algorithm is learning whether to interrupt the user or not when different problem situations or situations of interest arise in a certain application the user is working with. Our algorithm is based on the observation of a user’s actions and a user’s interaction with the agent, particularly on a user’s reactions to the agent’s assistance actions. As shown in Figure 5.1, the information obtained from the observation of the user-agent interactions is stored as user-agent interaction experiences, which are the inputs for our algorithm. The output of the IONWI algorithm is a set of user interruption preferences, which are part of the user interaction profile. When an interface agent has to decide whether to interrupt the user or not, the user interaction profile enables it to make the right choice.

The rest of the Chapter is organized as follows. Section 5.2 describes what an interruption is and how interruptions can be categorized. Section 5.3 describes some general studies regarding interruptions, some of which will be taken into account in the IONWI algorithm. Section 5.4 describes our proposed learning algorithm. Section 5.5 describes how the knowledge obtained with IONWI is updated as new interaction experiences are observed. Section 5.6 describes how an interface agent uses IONWI to provide personalized assistance to the user. Finally, in Section 5.7 we develop an example of the utilization of IONWI.
5.2 Interruptions: Concept and Categories

McFarlane [McFarlane, 1997] defines an interruption as “methods by which a person shifts his or her focus of consciousness from one processing stream to another”. Research has found that interruptions are harmful [Czerwinski et al., 2000a, Czerwinski et al., 2000b, Bailey et al., 2000]. They are disruptive to the primary computing task (that is the task the user is carrying out when he is interrupted), they decrease a user’s performance and they affect a user’s emotional state [Bailey et al., 2001, Ziljastra et al., 1999]. However, interruptions are necessary, since people need to monitor for important information and to communicate with other individuals. With the advent of continuous availability enabling devices such as pagers and cell phones, instant messaging, semi-autonomous and autonomous monitoring services, intelligent agents, and technology supporting concurrent multitasking and collaboration, interruptions are part of the current and future job description.

Interruptions can be categorized according to their relevance to the primary user task. Some interruptions may be pertinent to the primary task and assist in completing this task, by contributing direct answers to subtasks of the primary task or providing needed or helpful information for the primary task. For example, in a calendar management application, if the user is scheduling a new event and the interface agent interrupts him to suggest a potential time for the event, then this interruption is relevant to the primary user task. On the other hand, irrelevant interruptions are not helpful in moving towards the completion of the primary task.

In turn, irrelevant interruptions can be divided into two categories. Interruptions may be related in content to the primary task, even though they do not actually contribute to the completion of this task. These interruptions are defined as irrelevant but related. For example, consider that a user is scheduling a new event and he receives a notification of an email inviting him to a meeting that overlaps with the event he is scheduling. In this case, the situation being notified does not help the user to finish his task, but it is related to this task since the user will have to take this situation into account to complete it. There are also other interruptions that neither contribute towards the primary task completion, nor are related in content to this task. These interruptions are categorized as both irrelevant and unrelated. For instance, considering the same situation of the previous example, if the user receives a notification of an incoming email not related with the user’s schedule, this situation is completely unrelated and irrelevant to the primary user task.

Previous research on interruptions has focused before on interruptions which were relevant to the task at hand versus irrelevant interruptions [Czerwinski et al., 2000b]. Their results are not surprising, since they found that interruptions relevant to the
5.2. Interruptions: Concept and Categories

primary task would result in less time spent on the interrupting task and faster time to resume the primary task. In [Czerwinski et al., 2000b], relevant interruptions were defined as interruptions which gave direct answers to a primary task subtask. Irrelevant interruptions were defined as those which gave related information. The authors considered predefined interrupting tasks and user tasks to performed their experiments, that is, they studied predefined relevant and irrelevant interruptions with respect to a set of tasks users had to perform as part of the experiment. No general method to determine the relevance of interruptions to primary user activities was given in this work.

At this point, we should mention related works in Plan Recognition. Works in this area aim at inferring a user’s goals and a portion of that person’s plan for achieving his goals, to improve the effectiveness of an interactive system [Carberry, 2001]. Knowing a user’s plans can help to determine when (within a task, for example) to interrupt him and when not.

Although the categorization of interruptions as relevant or irrelevant to the primary user task is important, it does not consider the user’s preferences and interests. An interruption relevant to the current task may be not relevant to the user. In turn, an irrelevant or unrelated interruption may be important for the user. In our approach, we will categorize interruptions with respect to their relevance to the user being interrupted. We do not want the agent to interrupt the user when he does not want to be interrupted, because he will reject the agent. For example, one of the problems of Clippit (MS Office Assistant) is that it does not consider the user’s preferences, and it interrupts the user when it considers he needs help with the task he is carrying out.

Some interruptions are relevant to the user because the problem or the situation originating the interruption is relevant to him. These interruptions can be related or not to the primary task. For example, a notification about an urgent meeting organized by the user’s boss is relevant to the user. Thus, an interruption notifying this event is worthy no matter which the primary user task is. Other interruptions are irrelevant to the user because the underlying problem or situation is irrelevant to him. Again, these interruptions can be related or unrelated to the primary user task. For example, a notification or reminder about a party a work-mate is organizing because of his birthday can be irrelevant to the user, even if he is scheduling a new event for that day. Thus, the interface agent will have to learn which situations are relevant or urgent for the user, and which irrelevant so that no irrelevant interruptions are generated.

There are two mechanisms the agent can use to learn whether an interruption
is relevant or irrelevant to the user: the user feedback (both explicit and implicit) and the examples the user explicitly provides. However, although users probably can state some situations in which they want to be interrupted and some others in which they do not want to be interrupted, the agent also has to learn from users’ actions which their requirements and preferences are. In this work, we propose an algorithm to learn a user’s interruption preferences from a set of user-agent interaction experiences, considering the different types of interruptions we have defined in this section. The user’s interruption preferences are part of the user interaction profile.

5.3 Working with interruptions: related research

There have been numerous studies exploring interruptions in a general way, mainly in the Human Computer Interaction (HCI) area. These studies revealed that the disruptiveness of an interruption is related to several factors, including complexity of the primary task and/or interrupting task, similarity of the two tasks [Gillie and Broadbent, 1989], whether the interruption is relevant to the primary task [Czerwinski et al., 2000a], stage of the primary task when the interruption occurs [Czerwinski et al., 2000b], management strategies for handling interruptions [McFarlane, 1999], and modalities of the primary task and the interruption [Arroyo et al., 2002, Latorella, 1998].

Other studies found that interruptions are not always harmful, and that they may vary in how they affect users’ performance. For example, one study [O’Conaill and Frochlich, 1995] found that 64% of interruptions (in which interruptions had a human initiator) result in the recipient receiving some benefit from the interaction taking place. In the case where the primary task is simple and non-challenging, interruptions require the recipient to focus more on the primary task and resulted better overall performance [Speier et al., 1997]. Having experience in handling a task with interruptions, such as via training, can also suppress some of the harmful effects of interruptions [Hess and Detweiler, 1994].

The following subsections briefly describe a set of works that have studied interruptions from different points of view.

5.3.1 Visual and Multimodal Strategies for Interruption

McFarlane [McFarlane, 1999] examined four strategies for deciding when to interrupt someone during multi-tasked computing. He explored several interruption policies, including immediate (requiring an immediate user response), negotiated (user
chooses when to attend), mediated (an intelligent agent might determine when it is best to interrupt), and scheduled (interruptions come at prearranged time intervals). He found that none of these methods was the single best strategy to interrupt users in tasks across all performance measures. He concluded that giving people the control to negotiate for the onset of interruptions resulted in good performance.

Several studies have shown that the nature of the display of the notification influences the performance on the primary computing task. The work presented in [Maglio et al., 2000] demonstrated that continuously scrolling displays were more distracting than discrete displays (those that start and stop) to ongoing word editing tasks. They found that all notification styles reduced word-editing performance in comparison to a no-notification condition. Arroyo [Arroyo et al., 2002] studied different modalities of interruptions to determine the disruptiveness associated with each of them. Their work included five interruption modalities: heat, smell, sound, vibration and light. The study showed that the least used modalities in computer interfaces (smell, vibration) have bigger disruptive effects.

### 5.3.2 Interruption Length, Complexity and Relationship with the Primary User Task

Gillie and Broadbent [Gillie and Broadbent, 1989] presented a series of experiments aimed at elucidating features of interruptions that make them more or less disruptive to an ongoing computer task. They manipulated interruption length, similarity to the ongoing task, and the complexity of the interruption. They showed that being able to rehearse one’s position in the main task does not protect one from the disruptive effects of an interruption. In addition, they discovered that interruptions with similar content could be quite disruptive even if they are extremely short.

People at Microsoft Research has deeply studied the effects of instant messaging (IM) in users, mainly on ongoing computing tasks [Czerwinski et al., 2000a, Czerwinski et al., 2000b, Cutrell et al., 2001]. These authors found that IM that were relevant to ongoing tasks were less disruptive than those that were irrelevant. This influence of relevance was found to hold for both notifications viewing and task resumption times, suggesting that notifications that were unrelated to ongoing tasks took longer to process.

### 5.3.3 Temporal Strategies for Interruption

Some studies [Czerwinski et al., 2000b, Cutrell et al., 2001] place moments for interruption towards the beginning, middle, or end of a task. This kind of strategy
5.3. Working with interruptions: related research

relates most to [Miyata and Norman, 1986]. The authors explain that task execution occurs in three phases: planning, execution and evaluation. In [Czerwinski et al., 2000b] the authors demonstrated that the delays associated with an IM disruption depends on the point in a computing task that a user is presented with it. They found that a good time for notifications is early in the task, before the user has become deeply engaged in the task goal, and that notifications arriving during the evaluation, planning and executions phases of a task were harmful. Although there are clearly effects to interrupting during the various phases [Zijlstra et al., 1999], associating rough temporal placement (beginning, middle, end) might be an oversimplification of task execution.

Other studies place interruptions between instances of repetitive sequences or, more generally, at breakpoints in a task sequence [Miller, 2002b, Monk et al., 2002]. The choice of these points is more intuitive but the reasoning behind these locations remains ill defined. Studies of this type sometimes produce internally inconsistent results [Miller, 2002b].

In [Adamczyk and Bailey, 2004] the authors measure the effects of interrupting a user at different moments within task execution in terms of task performance, emotional state, and social attribution. Their results show that different interruption moments have different impacts on user emotional state and positive social attribution, and suggest that a system could enable a user to maintain a high level of awareness while mitigating the disruptive effects of interruption. They used task models based on event perception to predict the better and worse moments for interruption.

5.3.4 Cost and Utility of Interruptions

In [Horvitz et al., 1999] the authors use decision-theoretic principles to control alerting in computing and communication systems. Alerts (incoming mail, tips about application usage) provide potentially valuable information at the cost of interruption. They present an attention-sensitive approach for computing the expected value of alerts. They frame the discussion within the task of relaying notifications about incoming email messages. Their approach centers on developing the means for automatically assessing the expected utility of messages and for making inferences about a user’s focus of attention by monitoring multiple sources of information. They analyze the expected cost of interruptions, the expected cost of deferring alerts and the criticality of email messages.

As we have already said, these studies come from different research areas in
which interface agents are not included. However, the results of these studies can be taken into account by interface agents to provide assistance to users without affecting users’ performance in a negative way and diminishing the disruptiveness of interruptions. Particularly, interface agents have to consider those studies that tell them when to interrupt the user with respect to the task execution and those that tell us to analyze the relationship between the interruption and the current user task. However, none of the related works we have discussed has considered the relevance of interruptions to users, or the relevance the situation originating the interruption has for the user. This issue and the relevance of interruptions to user tasks are two aspects of interruptions our learning algorithm considers. The following section presents our solution to deal with the problem of interruptions within interface agent technology.

5.4 The IONWI learning algorithm

In this work, we propose an approach to determine when an interface agent can (or should) interrupt a user and when not. Our approach differs with the related works described in the previous section since it takes into account the user’s preferences regarding interruptions and notifications. We propose an algorithm, named IONWI, capable of learning when to interrupt a user and when not from the observation of the user’s interaction with a computer application and with the agent. The algorithm learns when a situation that may originate an interruption is relevant to the user’s needs, preferences and goals, and when it is irrelevant. In addition, this algorithm also considers the relationship and relevance the situation originating the interaction has to the user’s current task.

The outputs of the IONWI algorithm are a set of user interruption preferences, which are part of the user interaction profile. As shown in Figure 5.2, the IONWI algorithm uses the information contained in a set of user-agent interaction experiences to obtain a user’s interruption preferences. Thus, when an interface agent has to decide whether to interrupt the user or not given a certain problem situation, the agent uses the knowledge it has acquired about a user’s interruption preferences to choose the assistance modality it supposes the user expects in that particular instance of a given situation. Once the assistance has been provided, the agent obtains explicit and/or implicit user feedback. This new interaction is recorded as an assistance experience, which will be used in the future to incrementally update the knowledge the agent has about the user.

In order to obtain a user’s interruption preferences from a set of user-agent in-
5.4. The IONWI learning algorithm

![IONWI Diagram]

Figure 5.2: IONWI Overview

interaction experiences the IONWI algorithm first generates a set of hypotheses. A hypothesis expresses the agent’s belief that the user requires a certain assistance modality (interruption of notification without interruption) in a given situation. A hypothesis \( H \) expresses that whenever situation \( \text{Sit} \) occurs the user will require an interruption (or a notification) with a certainty degree of \( \text{Cer}(H) \). In turn, a hypothesis can also indicate that when situation \( \text{Sit} \) occurs and the user is performing task \( \text{Task} \), the user requires an interruption (notification) with a given certainty. This type of hypothesis considers that the relationship between the interrupting situation and the current user task may be an important factor to decide whether to interrupt the user or not.

To validate a hypothesis, the algorithm gathers positive evidence supporting it and negative evidence rejecting it. Both types of evidence are used to compute the certainty degree of the hypothesis. If this certainty degree is greater than a threshold \( \delta \), then the hypothesis is considered as valid and it is turned into a fact that represents a user’s interruption preference. Otherwise, it is discarded. These steps are shown in Figure 5.3.

5.4.1 Algorithm Inputs and Outputs

As described in Chapter 3, the input for our learning algorithm is a set of user-agent interaction experiences. An interaction experience \( \text{Ex} \) is described by seven arguments\(^1\) \(<\text{Sit}, \text{Mod}, \text{Task}, \text{Rel}, \text{UF}, E, \text{date}>\): a problem situation or situation of interest \( \text{Sit} \) is described by a set of features and the values these features take, \( \text{Sit} = \{(\text{feature}_i, \text{value}_i)\} \); the modality \( \text{Mod} \) that indicates whether the agent

\(^1\)From all the parameters recorded in an interaction experience, we consider only those relevant to the IONWI algorithm.
interrupted the user or not to provide him assistance; the Task the user was executing when he was interrupted or notified, which is described by a set of features and the values these features take \( \text{Task} = \{(\text{feature}_i, \text{value}_i)\} \); the relevance \( \text{Rel} \) the interruption has for the Task; the user feedback \( \text{UF} \) (regarding the assistance modality) obtained after assisting the user; an evaluation \( E \) of the assistance experience (success, failure or undefined); and the date when the interaction experience was recorded.

A key point when processing a user-agent interaction experience is user feedback analysis. If the agent has interrupted the user’s work, then it has to discover if the user appreciates this interruption because the situation at hand is really relevant for him or if the interruption hinders him because the problem is less important than the task the user was working on. If the user continues with his previous task, then the interruption has been useless, but if the user leaves his previous task and attends the problem that has been notified, then the interruption has been not in vain. The user feedback processing is inherently domain dependent.

On the other hand, if the agent has sent a notification, the user can pay attention to it immediately or delay it some time. In order to evaluate the urgency the situation has for the user, the algorithm will have to analyze the consequences of this delay. If it is too late when the user pays attention to the notification, it is a signal that an interruption would have been more appropriate. Again, this analysis
is domain dependent.

For example, consider that the user is scheduling a meeting with several participants and he is interrupted by his agent to remind him about a business meeting that will take place the next day. The user does not pay attention to the message being notified and presses a button to tell the agent not to interrupt him in these occasions. From this experience the agents learns that reminders of this kind of meetings are not relevant to the user, and it will send him a notification in the future without interrupting him. In this example, the different parts of the assistance experience are:

- **Sit** = \{(type, event reminder), (event-type, business meeting), (organizer, boss), (participants, [Johnson, Taylor, Dean]), (topic, project A evolution), (date, Friday), (time, 5p.m.), (place, user’s office)\}
- **Mod** = interruption
- **Task** = \{(application, calendar management system), (task, new event), (event type, meeting), (priority, high), ……”\}
- **Rel** = irrelevant, unrelated
- **UF** = \{(type, explicit), (action, do not interrupt)\}
- **E** = \{(type, failure), (certainty, 1.00)\} (interruption instead of notification)
- **Date** = \{(day, 18), (month, April), (year, 2003)\}

Figure 5.4 shows part of a file containing a set of user-agent interaction experiences in the calendar management domain. This file constitutes an input to the IONWI algorithm\(^2\). The situation originating the interruption is, in this example, a notification about an urgent meeting.

The output of our algorithm is a set of facts representing the user’s interruptions preferences. Each fact indicates whether the user needs an interruption or a notification when a given situation occurs in the system. Facts constitute part of the user interaction profile which was introduced in Chapter 3. These facts may adopt one of the following forms: “in problem situation Sit the user should be interrupted”, “in situation Sit the user should not be interrupted”, “in situation Sit and if the user is performing the task Task, he should not be interrupted”, “in situation Sit and if the user is performing the task Task, the agent can interrupt him”. Each fact \( F \) is accompanied by a certainty degree \( Cer(F) \) which indicates how certain the agent is

\(^2\)The format of the file is the one used by WEKA (http://www.cs.waikato.ac.nz/~ml/weka). See Appendix B.
about this fact. The following sections describe how we obtain facts from the set of user-agent interaction experiences.

5.4.2 Learning users’ interruption preferences with IONWI

The IONWI algorithm uses association rules to obtain the existing relationships between situations, current user tasks and assistance modalities. The association rules generated from the user-agent interaction experiences are automatically post-processed in order to derive useful hypotheses from them. Post-processing includes detecting the most interesting rules according to our goals, eliminating redundant and insignificant rules, pruning out contradictory weak rules, and summarizing the information in order to formulate the hypotheses more easily. As we have said before, once a hypothesis is formulated, the algorithm looks for positive evidence supporting the hypothesis and negative evidence rejecting it in order to validate it. The certainty degree of the hypothesis is computed taking into account both the positive and the negative evidence. This calculus is done by using metrics from association rule discovery. Finally, facts are generated from the set of highly supported hypotheses.

Algorithm 8 shows the main steps our algorithm involves when using association rules as machine learning technique. The following subsections describe how to determine interesting rules, how to filter out redundant and insignificant rules and
5.4. The \textit{IONWI} learning algorithm

how to find positive and negative evidence for a hypothesis within \textit{IONWI}.

\textbf{Algorithm 8} \textit{IONWI} Overview

\textbf{Input:} A set $Ex$ of user-agent interaction experiences $Ex_i = <St_i, Task_i, Rel_i, Mod_i, UF_i, E_i, date_i>$.  
\textbf{Output:} A set $F$ of facts and a set $H$ of hypotheses representing the user’s interruption preferences

1: $F \leftarrow \emptyset$
2: $H \leftarrow \emptyset$
3: $AR \leftarrow$ Call association rule mining algorithm with $Ex$
4: $AR_1 \leftarrow$ Filter out uninteresting rules from $AR$
5: $AR_2 \leftarrow$ Eliminate redundant and insignificant rules from $AR_1$
6: $AR_3 \leftarrow$ Eliminate contradictory weak rules from $AR_2$
7: $AR_4 \leftarrow$ Summarize the discovered rules contained in $AR_3$
8: $H \leftarrow$ Transform rules in $AR_4$ into hypotheses
9: for $i = 1$ to size of $H$ do
10: \hspace{1em} Find evidence for $(E^+)$ and against $(E^-)$ $H_i$
11: \hspace{1em} $Cer(H_i) \leftarrow$ compute certainty degree of $H$ considering $(E^+)$ and $(E^-)$
12: if $Cer(H_i) \geq \delta$ then
13: \hspace{2em} $F \leftarrow F \cup H$
14: \hspace{1em} end if
15: end for

5.4.3 Analyzing the interestingness of rules

In this work, we are interested in those association rules of the form "\textit{problem description, modality, task}$\rightarrow$\textit{user feedback, evaluation}", "\textit{problem description, modality}$\rightarrow$\textit{user feedback, evaluation}", "\textit{problem description, modality, relevance}$\rightarrow$\textit{user feedback, evaluation}", and "\textit{problem description, modality, task, relevance}$\rightarrow$\textit{user feedback, evaluation}", having appropriate support and confidence values. We are interested in these rules since they provide us information about the relationships between a situation or problem description and the modality of assistance the user prefers that received a positive (negative) evaluation. They also relate a situation and the current user task with an assistance modality, as well as a situation, the current user task and the relevance of the situation to the task with a certain assistance modality. Other combinations of items are irrelevant since we are trying to discover successful and not successful associations between a problem situation, a primary user task, and the modality of the assistance actions. These types of rules reveal, directly in the former case and indirectly in the latter case, the user’s interruption preferences.

To describe the selection of interesting rules, we should recall the definition of a
template given in Chapter 4. A template is an expression of the form:

\[ A_1, \ldots, A_k \rightarrow A_{k+1}, \ldots, A_n \]

where each \( A_i \) is either an attribute name, a class name, or an expression \( C^+ \) and \( C^* \), which correspond to one or more and zero or more instances of the class \( C \), respectively. A rule \( B_1, \ldots, B_h \rightarrow B_{h+1}, \ldots, B_m \) matches the pattern if the rule can be considered to be an instance of the pattern. The template the IONWI algorithm uses are: Problem Description\(^+\), Task\(^*\), Relevance\(^*\), Modality\(^+\) → User Feedback\(^*\), Evaluation\(^+\). Although using the \(+\) symbol, we will not have more than one problem description or more than one modality or evaluation in the rules. The same occurs with the \(*\) symbol, we will not have more than one task, task relevance or user feedback item.

### 5.4.4 Pruning the discovered rules

As we have mentioned in Chapter 4, once we have filtered out those rules that are not interesting or relevant to us, we will still have many rules to process, some of them redundant or insignificant. Many discovered associations are redundant or minor variations of others. Thus, those spurious and insignificant rules should be removed.

We will recall the definition of the set of pruning rules we use, which are a small subset of the rules defined by [Shah et al., 1999] (pruning rules 1 and 4 of those defined in this paper). Consider two rules \( R_1 \) and \( R_2 \):

- if \( R_1 = A \rightarrow C; R_2 = A, B \rightarrow C \), and either both rules are positive or negative with similar strength, then \( R_2 \) is redundant
- if \( R_1 = A \rightarrow B; R_2 = A \rightarrow B, C \), then \( R_1 \) is redundant

where a positive rule, denoted by \( A \rightarrow^+ B \), is a rule where the presence of attribute \( A \) is found to increase the probability of \( B \)’s occurrence significantly. Formally, this means that for a given user-defined coefficient \( P > 1 \), \( Pr(B/A) > P \times Pr(B) \) should be satisfied. A negative rule, denoted by \( A \rightarrow^- B \), is a rule where for a user-defined coefficient \( N > 1 \), \( Pr(B) > N \times Pr(B/A) \). The strength of an association rule is the confidence of the rule, that is, \( Pr(B/A) \). Two rules are of similar strength if for a small predefined value \( 1 > \varepsilon_3 > 0 \), \( |\text{strength}(\text{rule}_1) - \text{strength}(\text{rule}_2)| < \varepsilon_3 \), that is, \( |\text{confidence}(\text{rule}_1) - \text{confidence}(\text{rule}_2)| < \varepsilon_3 \).

An example of the first pruning rule is the following, with \( N = 2, P = 2, \varepsilon_3 = 0.06 \):

\( R_1: \) \text{Sit}((\text{Type, Event Reminder})(\text{Event-Type = doctor})), (\text{Task=View Calendar}), (\text{Mod = interruption}) \rightarrow (\text{UF = do not interrupt}, \text{Ev = failure}) \) [sup: 40/100, conf: 33/40]
5.4. The IONWI learning algorithm

R2: \( \text{Sit}((\text{Type}, \text{Event Reminder})(\text{Event-Type} = \text{doctor}), (\text{Task} = \text{View Calendar}), (\text{Event-Priority} = \text{high})), (\text{Mod} = \text{interruption}) \rightarrow (\text{UF} = \text{do not interrupt}), (\text{Ev} = \text{failure}) \) [sup: 40/100, conf: 31/40]

The difference between their confidence values is: \(|\text{confidence}(\text{rule}_1) - \text{confidence}(\text{rule}_2)| = |0.825 - 0.775| = 0.05 < 0.06\). The first rule says that when the user is viewing his calendar and the agent interrupts him to remind him about an appointment with the doctor, the user rejects the interruption. The second rule expresses that when the user is viewing his calendar and the agent interrupts him to remind him about an appointment with the doctor - event considered of high priority -, the user rejects the interruption. The second rule provides us little extra information with respect to the second rule. Thus, it should be eliminated.

The following is an example of the second pruning rule, where the redundant rule is R1:

R1: \( \text{Sit}((\text{Type}, \text{Event Reminder})(\text{Event-Type} = \text{meeting})), (\text{Task} = \text{View Calendar}), (\text{Mod} = \text{interruption}) \rightarrow (\text{Ev} = \text{success}) \) [sup: 50/100, conf: 43/50]

R2: \( \text{Sit}((\text{Type}, \text{Event Reminder})(\text{Event-Type} = \text{meeting})), (\text{Task} = \text{View Calendar}), (\text{Mod} = \text{interruption}) \rightarrow (\text{UF} = \text{OK}), (\text{Ev} = \text{success}) \) [sup: 52/100, conf: 44/52]

The difference between their confidence values is: \(|\text{confidence}(\text{rule}_1) - \text{confidence}(\text{rule}_2)| = |0.86 - 0.84| = 0.02 < 0.06\). The first rule indicates that when the user is viewing his calendar and the agent interrupts him to remind him about a meeting, the user accepts the interruption. The second rule gives also information about the explicit feedback the user provided. Thus, we prefer this rule since the consequent is more specific. The first rule is eliminated.

In addition to the pruning rules described above, we have to analyze certain combinations of attributes in our domain to determine if two rules are telling us the same thing. The following list shows some examples of redundancy in rules found in our domain, provided that the rules involved refer to the same problem situation (and eventually the same user task):

- interruption, failure \(\equiv\) notification, success
- interruption, success \(\equiv\) notification, failure

The first equivalence, for instance, expresses that a rule indicating that an interruption for a certain combination of situation and user task was evaluated as a failure, is equivalent to a rule indicating that for the same situation-task pair a notification was evaluated as a success. If these two rules have similar confidence values, we can eliminate the first rule and keep the second one. However, if two rules R1
and $R2$ provide the same information but their confidence values are very different,
\[|\text{confidence}(R1) - \text{confidence}(R2)| > \varepsilon_3,\] we have to keep the two rules.

As well as analyzing redundant rules, we have to check if there are any contradictory rules. We define that two rules are contradictory if, for the same situation and, eventually, for the same user task, they express that the user wants both an interruption and a notification without interruption. The following list shows some examples of contradictory rules in our domain, provided that they refer to the same problem situation:

- interruption, failure \equiv interruption, success
- interruption, success \equiv notification, success

If two contradictory rules $R1$ and $R2$ have very different confidence values, for example $R1$ has a confidence of 80\% and $R2$ has a confidence value of 20\%, then we will eliminate $R2$ and keep $R1$. More formally, we will eliminate $R1$ if for a small predefined value $1 > \varepsilon_4 > 0$, \[|\text{confidence}(R1) - \text{confidence}(R2)| > \varepsilon_4,\] and \[\text{confidence}(R1) < \text{confidence}(R2).\] However, if the confidence values are similar, i.e. \[|\text{confidence}(R1) - \text{confidence}(R2)| < \varepsilon_4,\] we will keep the two rules.

Finally, to avoid considering obsolete assistance experiences, our algorithm deletes old transactions from the database. We consider that an interaction experience is obsolete when its date is older than $\theta$ months.

The rules that survive all the pruning processes described before are those $IONWI$ considers to build the hypotheses.

### 5.4.5 Building facts from hypotheses

The association rules that have survived the pruning processes described in the previous section are those the $IONWI$ algorithm uses to build hypotheses about a user’s interruption preferences. Each single hypothesis is obtained from a set of association rules that are related because they refer to the same problem situation but are somewhat different: a “main” association rule; some redundant association rules with regards to the main rule, which could not be pruned out because they did not fulfill the similar confidence restriction; and some contradictory rules with regards to the main rule, which could be not pruned away because they did not meet the different confidence requirement. The main rule is chosen by selecting from the rule set the rule that has the greatest support value, whose antecedent is the most general and whose consequent is the most specific.
Similarly to WATSON, when the IONWI algorithm has formulated a set of hypotheses it has to validate them. This algorithm tries to prove a hypothesis by analyzing the evidence for and against it. The evidence for and against a given hypothesis is obtained by analyzing the association rules belonging to the rule set of the rule originating the hypothesis. Given an association rule that originates a hypothesis, we define as positive evidence those association rules that were considered as redundant during the pruning steps but were not eliminated. Formally, given an association rule $Sit_1, Mod_1, [Task], [Rel] \rightarrow Y$, a rule $Sit_2, Mod_2, [Task], [Rel] \rightarrow Y$ where $Sit_2 \supset Sit_1$ is considered a positive evidence (the brackets indicate that the attribute is optional).

We define as negative evidence those association rules that were considered as contradictory rules during the pruning steps but were not eliminated. Given an association rule $Sit, Mod, [Task], [Rel] \rightarrow Y$, a rule $Sit, Mod, [Task], [Rel] \rightarrow Z$, where $Y \neq Z$ and $Y$ and $Z$ are evaluations, is considered a negative evidence. Similarly, given a rule $Sit, Mod_1, [Task], [Rel] \rightarrow Y$, a rule $Sit, Mod_2, [Task], [Rel] \rightarrow Y$ where $Mod_1 \neq Mod_2$ is considered as a negative evidence.

As we explained in Chapter 4, the certainty degree of a hypothesis $H$ is computed as a function of the supports of the rule originating the hypothesis and the rules considered as positive and negative evidence of $H$. The function we use to compute certainty degrees is shown in Equation 5.1, where $\alpha$, $\beta$ and $\gamma$ are the weights of the terms in the equation (we use $\alpha=0.7$, $\beta=0.15$ and $\gamma=0.15$), $\text{Sup}(AR)$ is the support of the rule originating $H$, $\text{Sup}(E^+)$ is the support of the rules being positive evidence, $\text{Sup}(E^-)$ is the support of the rules being negative evidence, $\text{Sup}(E)$ is the support value of an association rule taken as evidence (positive or negative), $r$ is the amount of positive evidence and $t$ is the amount of negative evidence.

$$\text{Cer}(H) = \alpha \text{Sup}(AR) + \beta \frac{\sum_{k=1}^{r} \text{Sup}(E^+)}{\sum_{k=1}^{t} \text{Sup}(E)} - \gamma \frac{\sum_{k=1}^{t} \text{Sup}(E^-)}{\sum_{k=1}^{t} \text{Sup}(E)} \quad (5.1)$$

If the certainty degree of a hypothesis is greater than a threshold value $\delta$, the hypothesis becomes , otherwise it is discarded.

We have not discussed yet how we build a hypothesis from an association rule regarding the type of information it contains. If the evaluation is known and it is a success, then the hypotheses is built straightforward by associating the situation with the corresponding assistance modality. If the evaluation is known and it is a failure, then the algorithm can determine from the user feedback which the correct assistance modality is. Finally, if the evaluation is undefined, it is most likely that the evaluation is actually a failure but the algorithm can not tell which the required
action is because it does not have enough information.

Algorithm 9 shows the different steps our IONWI algorithm involves.

**Algorithm 9** IONWI learning algorithm

**Input:** A set $Ex$ of user-agent interaction experiences $Ex_i = <$
\[Sit_i, Mod_i, Task_i, Rel_i, UF_i, E_i, date_i\] > (where $Sit$: problem description,
$Mod$: modality of the assistance action, $Task$: user primary task, $Rel$: relevance
of $Sit$ to $Task$, $UF$: user feedback, $E$: evaluation, $date$: interaction date)

**Output:** A set $F$ of facts and a set $H$ of hypotheses about the user’s assistance
requirements

1: minsup ← Obtain value for minsup (explained later in Chapter 7)
2: minconf ← 0.8
3: Call association rule mining algorithm (Apriori) with $minconf$, $minsup$ and $Ex$
   {Post-process association rules}
4: $AR_1 ← Select$ from $AR$ those association rules $X → Y$ where $X = <$
   $Sit, Mod, [Task], [Rel]$ > and $Y = <$ $UF, E$ $>$
5: $AR_2 ← Filter$ out uninteresting rules from $AR_1$
6: $AR_3 ← Eliminate$ redundant and insignificant rules from $AR_2$
7: $AR_4 ← Eliminate$ contradictory rules from $AR_3$
   {Generate hypotheses}
8: $ARSet_i ← Group$ similar rules in $AR_4$ (a rule and those redundant and contra-
dictory rules that were not eliminated)
9: $ARSet ← ∪ARSet_i$
10: $n ← size$ of $ARSet$
11: **for** $i = 1$ to $n$ **do**
12: Generate hypothesis $H_i$ from $ARSet_i$
13: $H ← H ∪ H_i$
14: **end for**
   {Validate hypotheses}
15: **for** $i = 1$ to $n$ **do**
16: $E^+ ← Obtain$ positive evidence for $H_i$ from $ARSet_i$
17: $E^- ← Obtain$ negative evidence for $H_i$ from $ARSet_i$
18: Compute certainty $Cer(H_i)$ taking into account $E^+$ and $E^-$
19: **if** $Cer(H_i) ≥ δ$ **then**
20: $F ← F ∪ H_i$
21: **end if**
22: **end for**

### 5.5 Incremental Learning

As we said in Chapter 4, the database containing interaction experiences is not static, because updates are constantly being applied to it. On the one hand, new
interaction experiences are added since the agent keeps observing a user’s behavior.
On the other hand, old experiences are deleted from the database to save storage
5.5. Incremental Learning

space, and/or because they are out of interest or because they become obsolete. In consequence, new hypotheses about a user’s interruption preferences may appear and some of the learned hypotheses may become invalid.

To deal with this problem, we will adopt the same approach we adopted with WATSON. We will address the maintenance problem from the association rule point of view, that is, as the database changes new association rules may appear and at the same time, some existing association rules may become invalid.

IONWI uses the FUP2 algorithm [Cheung et al., 1997] to update the association rules and the DELI algorithm [Lee and Cheung, 1997] to determine when it is necessary to update the rules. The FUP2 algorithm can update the discovered association rules when new transactions are added to, deleted from, or modified in the database. Its framework is similar to the Apriori algorithm. The DELI algorithm uses a sampling technique to estimate the difference between the old and new association rules. This estimate is used as an indicator for whether the FUP2 algorithm should be applied to the database to accurately find out the new association rules. If the estimated difference is large enough (with respect to some user specified thresholds), the algorithm signals the need of an update operation, which can be accomplished by using the FUP2 algorithm. If the estimated difference is small, then we do not run FUP2 immediately and we can take the old rules as an approximation of the new rules. Hence, we wait until more changes are made to the database and then re-apply the DELI algorithm. Algorithm 10 shows the main steps of the incremental IONWI algorithm.

Algorithm 10 Incremental IONWI

Input: Original interaction experience database DB, deleted experiences DB−, new experiences DB+, actual fact set F

Output: updated fact set \( F_{new} \)

1: \( F_{new} \leftarrow \emptyset \)
2: updating \( \leftarrow \) false
3: updating \( \leftarrow \) call DELI algorithm with \( DB, DB-, DB+ \) to determine whether to update or not
4: if updating then
   5: \( AR \leftarrow \) call FUP2 with \( DB, DB-, DB+ \)
   6: \( F_{new} \leftarrow \) call IONWI post-processing steps with \( AR \)
7: else
8: \( F_{new} \leftarrow F \)
9: end if
10: return \( F_{new} \)
5.6 Assisting the user with \textit{IONWI}

Once we have constructed a fact base containing the user’s interruption requirements, the agent uses it to take decisions when it is about to assist a user. If there is a profile item involving both the situation that can originate an interruption and the task the user is carrying out, then the agent executes the assistance action with the modality the profile item indicates.

If there is a fact involving only the situation at hand in the user interaction profile, then the agent uses it to decide what to do. If the profile item indicates that the user do not want to be interrupted, then the agent sends him a notification. If the profile item indicates that the user can be interrupted, then the agent interrupts him.

In the case the agent can not find an interruption preference involving the problem at hand the agent will take the decision analyzing the relevance of the situation to be notified to the primary user task, and the relationships between them. If the situation is relevant to the task, then the agent will interrupt the user. If the situation is irrelevant but related to the current task, the agent can interrupt the user. However, if the situation is irrelevant and unrelated the agent will not interrupt the user. In this way, we are considering the results obtained by some studies that suggested that interruptions that are relevant to ongoing tasks are less disruptive than those that are irrelevant.

Algorithm 11 shows the algorithm used to decide whether to interrupt the user or not. This algorithm is an extension of the decision making algorithm that was introduced in Chapter 3. A possible alternative step is the following. If there is a profile item involving both the situation that can originate an interruption and the relevance of the task the user is carrying out, the agent can compute the relevance of the situation to the task to verify if the relevances match. If they do, the agent takes the decision suggested by the profile item. However, the agent cannot make a decision if the relevances do not match, and it should go to step 9.

To determine the relevance of interruptions to user primary tasks and the relationship between them, we propose the following method. First, to determine if the situation to be notified at hand is related to the primary user task, the agent has to analyze if the situation occurs in the same application the user is working with or in a different application. A simplification to this problem is considering that if the applications are different the interruption is irrelevant to the task; or the domain expert can establish relationships between applications, such as between a
Algorithm 11 Decision making with IONWI

Input: A problem situation \( \text{Sit} \) to deal with; the user profile composed by the regular profile \( \text{URP} \), the user interruption preferences \( \text{UPI} \) and the user assistance requirements \( \text{UAR} \); the current user task \( \text{Task} \)

Output: The agent has chosen a type of action to deal with \( \text{Sit} \); warning \( W \), suggestion \( S \), action \( A \) or no action \( N \); a modality: interruption \( I \) or no interruption \( NI \); and a content. These three items constitute the decision \( \text{Dec} \)

1: compute \( \text{Conf}(\text{Sit}) \)
2: if \( \text{Conf}(\text{Sit}) \geq \tau_1 \) then
   3: \{the steps for selecting the action contents and type\}
4: \( \text{IP} \leftarrow \text{select from UIP} \) an item associated with \( \text{Sit} \) and \( \text{Task} \)
5: if \( \text{IP} \neq \emptyset \) then
6: \( \text{Dec.modality} \leftarrow \text{IP.modality} \)
7: else
8: \( \text{IP} \leftarrow \text{select from UIP} \) an item associated with \( \text{Sit} \)
9: if \( \text{IP} \neq \emptyset \) then
10: \( \text{Dec.modality} \leftarrow \text{IP.modality} \)
11: else
12: \( \text{sim} \leftarrow \text{similarity}(\text{Sit, Task}) \)
13: \( \text{rel} \leftarrow \text{relevance}(\text{Sit, Task}) \)
14: if \( \text{rel} = \text{irrelevant} \) and \( \text{sim} = \text{related} \) then
15: \( \text{Dec.modality} \leftarrow \text{NI} \) (or \( I \))
16: else if \( \text{rel} = \text{irrelevant} \) and \( \text{sim} = \text{unrelated} \) then
17: \( \text{Dec.modality} \leftarrow \text{NI} \)
18: else
19: \( \text{Dec.modality} \leftarrow I \{\text{rel=relevant}\} \)
20: end if
21: end if
22: end if
23: else
24: return \text{no action} \{\text{Sit not worth handling}\}
25: end if
5.6. Assisting the user with IONWI

<table>
<thead>
<tr>
<th>User Task</th>
<th>Agent Action</th>
<th>Relevance / Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>View calendar (day, week, month)</td>
<td>Event reminder</td>
<td>irrelevant; relationship depends on situation-task similarity</td>
</tr>
<tr>
<td>View calendar (day, week, month)</td>
<td>New e-mail</td>
<td>irrelevant; relationship depends on situation-task similarity</td>
</tr>
<tr>
<td>New event</td>
<td>Suggestion of event attributes</td>
<td>relevant</td>
</tr>
<tr>
<td>New event</td>
<td>Event reminder</td>
<td>relationship depends on situation-task similarity</td>
</tr>
<tr>
<td>New event</td>
<td>New e-mail</td>
<td>relationship depends on situation-task similarity</td>
</tr>
<tr>
<td>New event</td>
<td>Overlapping events</td>
<td>relevant or irrelevant; related</td>
</tr>
<tr>
<td>Edit event</td>
<td>Suggestion of event attributes</td>
<td>relevant</td>
</tr>
<tr>
<td>Edit event</td>
<td>Event reminder</td>
<td>relationship depends on situation-task similarity</td>
</tr>
<tr>
<td>Edit event</td>
<td>New e-mail</td>
<td>relationship depends on situation-task similarity</td>
</tr>
<tr>
<td>New contact</td>
<td>Event reminder</td>
<td>irrelevant; relationship depends on situation-task similarity</td>
</tr>
<tr>
<td>New contact</td>
<td>New e-mail</td>
<td>irrelevant; relationship depends on situation-task similarity</td>
</tr>
</tbody>
</table>

Table 5.1: Relationships between user tasks and agent actions

calendar management system and an email system. Then, the agent has to analyze if the notification is related to the particular task the user is currently carrying out. A domain expert can establish the relationships between the set of tasks a user can perform with an application, and also between user tasks and an agent’s assistance tasks, such as those we mentioned in the examples in the previous paragraphs. Table 5.1 shows some examples of relationships between user tasks and agent actions in the calendar management domain.

In some situations, such as an incoming email and a new event task, some further analysis is necessary. We need to compare the characteristics of the situation to be notified against the characteristics of the primary user task. To achieve this goal, we can represent a user task with a set of attribute-value pairs, in the same way as we represent situations, and compare then these representations. We assume that the information about the features describing a user’s tasks in a given application can be extracted from the user interface (for example, label names, menu bars) or can be defined by a domain expert using, for example, a task model. In such a model, a task defines how the user can reach a goal in a specific user interface of an application. The goal is either a desired modification of the state of a system or a query to it. Figure 5.5 shows a task model describing the add meeting task in a calendar management system. The notation we use is the ConcurTaskTrees notation [Mori et al., 2002]. Each task in the model has a tree-like structure where leaves are simple tasks and internal nodes represent tasks composed by other (simple or composed)

---

3 A more complex analysis taking into account the whole context in which the user is working is necessary; but it is beyond the scope of this work. We have reduced the problem making some assumptions and we propose a simple solution to deal with it.
5.6. Assisting the user with IONWI

Figure 5.5: Task model for adding a meeting
tasks. Task models have been used in Plan Recognition to determine what the user is doing, where his focus is, and then discover when to interact with him, for example, to provide him assistance [Rich et al., 2001]. Recently, tasks models have been used to determine the best point in a task to interrupt the user [Adamczyk and Bailey, 2004].

IONWI uses a similarity metric to compare the features involved in the task and the situation to determine the similitude between them. This metric computes a similarity function that gives as result a score representing how similar the primary user task and the situation are. If this score is higher than the threshold value $\phi$ specified by the metric, then the situation is related to the task. If not, they are unrelated. Equation 5.2 shows the function IONWI uses to compute the similarity between a situation $Sit_i$ and a task $Task_j$. In that function, $w_k$ is the weight of feature $k$, $sim_k$ is the similarity function for feature $k$, and $f_{ik}$ and $f_{jk}$ are the values for features $k$ in the situation and the task, respectively. The value $p$ is the minimum of $p_i$ and $p_j$, which are the number of features of the situation and the number of features of the task, respectively. We assume that features $k$ in both representations are comparable, that is, they refer to the same attribute.

$$similarity(Sit_i, Task_j) = \frac{\sum_{k=1}^{p} sim_k(f_{ik}, f_{jk}) * w_k}{p} \tag{5.2}$$

Figure 5.6 shows the features describing a new event task the user is performing and the features describing an interruption originated by an overlapping between this event and an older one.

For the example given in Figure 5.6, the similarity function is computed as shown in Equation 5.3. In this example we assume that every attribute has the same weight $w_k = 1$, and that two features are similar if they have the same value ($sim$ returns 1), and they are not similar otherwise ($sim$ returns 0).
5.7. Example

In this section we will provide an example of the whole process the IONWI algorithm involves. For simplicity reasons, we will consider an small set of input data corresponding to the interaction between the user of our example in Chapter 4 and the interface agent assisting him with his calendar management. The following paragraphs describe the preferences of this user regarding interruptions.

If there is an overlapping between a meeting organized by the user and an event organized by the user’s boss or by the professor in charge of “Object Oriented Programming”, then the user wants the agent to interrupt it. In other cases, the user wants to be notified without an interruption.

Figure 5.7 shows the file containing the interaction experiences corresponding to the user we are analyzing. This file contains experiences in which the situation is an overlapping between two events. We are only considering 10 interaction experiences.

\[
similarity(Sit_i, Task_j) = \frac{\sum_{m=1}^{6} \operatorname{sim}_m(meeting, meeting) + \ldots + \operatorname{sim}_6(user, user)}{6} = 1
\]  

(5.3)

If the value the similarity metric returns is greater than \( \phi = 0.6 \), the situation and the task are similar, i.e. the interruption is related to the primary user task.

As a summary, Algorithm 12 shows how to determine the relationship between a potential interruption and the primary user task.
Algorithm 12 Determining the relationship between interruptions and primary user tasks

Input: A situation \textit{Sit} that can originate an interruption and the \textit{Task} the user is engaged in

Output: The relevance relationship between \textit{Sit} and \textit{Task} (relevant, irrelevant related, irrelevant unrelated)

1: \( RR \leftarrow \emptyset \)
2: if \textit{Sit.application} not relate to \textit{Task.application} then
3: return \( IU \) \{Do not belong to the same application or no relationship set by expert\}
4: else
5: \( RR.relevance \leftarrow \text{get-relevance} (\textit{Sit}, \textit{Task}) \) \{domain expert pre-established relationships\}
6: if \( RR.relevance = "irrelevant" \) then
7: \( RR.relationship \leftarrow \text{get-relationship}(\textit{Sit}, \textit{Task}) \)
8: if \( RR.relationship = "undefined" \) then
9: \( s \leftarrow \text{similarity}(\textit{Sit}, \textit{Task}) \) \{Further analysis is needed\}
10: if \( s \geq \phi \) then
11: \( RR.relationship \leftarrow \text{related} \)
12: else
13: \( RR.relationship \leftarrow \text{unrelated} \)
14: end if
15: end if
16: end if
17: end if
18: return \( RR \)
Figure 5.7: Input file for IONWI

The first step of the IONWI algorithm consists of generating association rules from the user-agent interaction experiences. Figure 5.8 shows a subset of these association rules (we generated 1000 rules).

From all the association rules generated, the IONWI algorithms filters out those that are not relevant to its purposes according to the templates described in Section 5.4.4. The rules that survived the first filtering process are the following:

1. event type (1) = meeting, place (1) = office, agent action = warning, modality = notification, primary task = new event, primary task relevance = relevant → user reaction = interrupt next time, evaluation = failure; sup(0.6), conf(1)

2. place (1) = office, agent action = warning, modality = notification, primary task = new event, primary task relevance = relevant, user reaction = interrupt next time → evaluation = failure; sup(0.6), conf(1)

3. event type (1) = meeting, agent action = warning, modality = notification, primary task = new event, primary task relevance = relevant → user reaction = interrupt next time, evaluation = failure; sup(0.6), conf(1)

4. event type (1) = meeting, place (1) = office, modality = notification, primary task = new event, primary task relevance = relevant → user reaction = interrupt next time, evaluation = failure; sup(0.6), conf(1)
5.7. Example

5. event type (1) = meeting, place (1) = office, agent action = warning, modality = notification, primary task relevance = relevant → user reaction = interrupt next time, evaluation = failure; sup(0.6), conf(1)

6. event type (1) = meeting, place (1) = office, agent action = warning, modality = notification, primary task = new event → user reaction = interrupt next time, evaluation = failure; sup(0.6), conf(1)

The next step is eliminating redundant rules according to Shah [Shah et al., 1999] pruning rules. In this example, all the rules survived this filtering step. The reasoning process is as follows. At a first glance, rule 1 is redundant with respect to rule 2, rule 3, rule 4, rule 5 and rule 6. However, when analyzing if the rules taken in pairs are either both negative or both positive, we find that none of them fulfills this requirement. Thus, no rule is considered redundant and no rule is deleted.

For example, consider the following set of rules:

1. event type (1) = meeting, place (1) = office, agent action = warning, modality = notification, primary task = new event, primary task relevance = relevant → user reaction = interrupt next time, evaluation = failure; sup(0.6), conf(1)

2. place (1) = office, agent action = warning, modality = notification, primary task = new event, primary task relevance = relevant, user reaction = interrupt next time → evaluation = failure; sup(0.6), conf(1)

According to the notation used in Section 5.4.4, rule 1 can be written as $A \rightarrow B$. The support of the consequent, $Pr(B)$, is 0.6. The confidence of the rule, $Pr(B/A)$,
is 1. Since $1 > 0.6 \times 2$ is not true, the rule is not positive. Similarly, $0.6 > 2 \times 1$ is not true. Thus, the rule is not negative.

The same analysis is done with rule 2. For this rule, $Pr(B)$ is 0.6 and $Pr(B/A)$ is 1. Then, the calculus are the same than for rule 1 and the rule is neither negative nor positive. In consequence, the apparently redundant rule (rule 1) cannot be deleted.

Next, contradictory rules are searched. In this example, no contradictory rules are found. Thus, all the rules are considered to build the hypotheses. The components of each of the hypotheses generated by our algorithm are shown below.

- **Main Rule:** event type (1) = meeting, place (1) = office, agent action = warning, modality = notification, primary task = new event → user reaction = interrupt next time, evaluation = failure; sup(0.6), conf(1). **Positive Evidence:** event type (1) = meeting, place (1) = office, agent action = warning, modality = notification, primary task = new event, primary task relevance = relevant → user reaction = interrupt next time, evaluation = failure; sup(0.6), conf(1). **Negative Evidence:** none. **Certainty:** 0.57

For example, the certainty degree of hypothesis 1 is computed as follows.

$$Cer(H) = 0.7 \times 0.6 + 0.15 \times \frac{0.6}{0.6} - 0.15 \times 0 = 0.42 + 0.15 = 0.57$$

In this example, the certainty of the hypothesis is greater than the main rule support because the certainty calculus also includes the support of the positive evidence. This kind of adjustment is the purpose of including the positive and negative evidence in the certainty calculus. After summarizing the information contained in the hypotheses, the user interruption preferences are the following.

- **situation:** event type = meeting, place = office, agent action = warning; primary task = new event; modality of assistance: interruption

This interruption preference reflects directly one of the user’s preferences stated at the beginning of this section. The other preferences will surely come up after some more user-agent interaction experiences. As regards decision making, if the situation the agent has to deal with is the one contained in the user profile, then it will interrupt the user to provide him assistance. In any other situation, it will analyze the relationship between the current task and the interruption.

## 5.8 Summary

In this chapter we have presented a solution to one of the problems interface agents have to face when they want to personalise the assistance they provide to their users.
5.8. Summary

The IONWI algorithm enables an interface agent to learn when the user requires an interruption and when he does not want to be interrupted. The following chapter describes two application that implement our profiling algorithm.
Chapter 6

Applications and Tools implemented

To materialize and evaluate our proposed personalization approach we developed an application that implements our proposed approach, Calendar Agent, and a tool to test our algorithms, User Profiling Toolkit. Calendar Agent is an interface agent that assists users with the organization of their calendars. User Profiling Toolkit is a tool that enables us to make experiments with different user profiling and decision making algorithms. In this Chapter we provide an overview of these systems, focusing our attention on how the interaction between the user and the agent is personalized. We also describe how these applications were designed and implemented.

6.1 Introduction

In the previous chapters we presented a personalization approach to enhance the interaction between users and interface agents. Our approach suggests learning a user’s interaction and interruption preferences and assistance requirements by observing a user’s behavior while he is interacting with an interface agent. Then, the interface agent builds a user interaction profile and uses this profile to personalize its interaction with the user.

To materialize our proposed approach we developed an interface agent named Calendar Agent that assists users with their calendar management. We chose this domain because it is rich in the variety of assistance actions an agent can execute and in the variety of user responses to these assistance actions. Moreover, an agent assisting users with this task can perform the three assistance actions we are considering in this work: warnings (reminders, alerts), suggestions and actions on the user’s behalf. In turn, this assistance can be provided by interrupting the user while he is working with the application or without interrupting him. Calendar Agent assists users with the organization of their calendars by suggesting meeting dates,
times and places, by warning users about overlapping and upcoming events, and by managing the scheduling of some events on the user’s behalf. Section 6.2 describes the Calendar Agent application.

The second system we will describe is the User Profiling Toolkit. User Profiling Toolkit is a tool that enables us to make experiments with different user profiling algorithms. One of the problems we experienced with previously developed interface agents was evaluating them. In general, making experiments with a substantial amount of real users during a considerable period is not always possible. Besides, it is difficult to test the different parameters of a profiling algorithm by making experiments with real users. It would take as long and the results would probably be not completely useful. A common method for evaluating algorithms in the Machine Learning and Data Mining communities is using a tool that enables users to select different algorithms, set the values for the different parameters, and analyze and visualize results. Examples of such tools are WEKA\(^1\), Xlopes\(^2\), Hugin\(^3\), BKD\(^4\) and ArTool\(^5\). Thus, we decided to apply these ideas to evaluate and experiment with user profiling algorithms. Particularly, we experimented with the \textsc{WATSON} algorithm, the \textsc{IONWI} algorithm, and the decision making algorithms associated with them. Section 6.3 presents an overview of the User Profiling Toolkit. Finally, in Section 6.4 we provide details about how we designed and implemented the two applications.

### 6.2 Calendar Agent

A calendar management system enables a user to organize and visualize information about his activities. By using such a system a user can manage his calendar: he can schedule events, delete events, edit the information of an event, add priorities to events; visualize his calendar in different formats such as daily, weekly or monthly; manage his contacts, that is add a new contact, edit the information of a contact, send an email to a contact; and generate reports (a list of activities for a particular day). An event is typically described by several attributes: the event type (business meeting, gym class, a course, social meeting, appointment with the doctor, party), a description, a date, a time, a duration, the place where the event takes place, a list of participants (one or more), a state (confirmed, pending, canceled), a priority and a host. An event may be recurrent (daily, weekly, or monthly) or not. A user may

\(^1\)http://www.cs.waikato.ac.nz/~ml/weka
\(^2\)http://www.prudsys.com/Produkte/Algorithmen/Xlopes/
\(^3\)http://www.hugin.com/
\(^4\)http://www.bayesware.com/
\(^5\)http://www.cs.umb.edu/~laur/ARTool
also want to schedule activities for a particular day or for a week without specifying a time interval, such as: “go to the supermarket, pay the phone bill and go to the library”.

Managing our calendar and organizing our schedules are tasks we carry out almost every day, which demand us not only time but also effort. For example, to schedule a meeting with his clients a sales manager has to determine first a free time slot in his calendar, he has to choose a meeting place and he has to determine the duration of the meeting. Then, he has to find out if the meeting date and meeting time are convenient for his clients, and if they are not, he has to find another date, time and probably place. This manager also has to schedule meetings with his employees, meetings with his boss, gym classes twice a week, social dinners at the club, among other events. In order to schedule these events the sales manager considers, as everybody does, his priorities for the different kinds of events, his preferences for meeting dates and times, his relationship with the participants of the events, his commitments and goals.

An interface agent can assist users with such a complex task in many different ways, such as suggesting meeting places, meeting times and meeting dates; notifying the user about upcoming events; scheduling an appointment on the user’s behalf; rejecting an invitation for a party because the user has a business dinner, and organizing a meeting with some customers. Some meeting scheduling agents have been developed, which can suggest meeting dates and times and negotiate with other agents when conflicts arise [Kozierok and Maes, 1993, Sen and Durfee, 1994, Mitchell et al., 1994, Garrido and Sycara, 1996]. However, these agents do not have the capability of personalizing the interaction with users.

6.2.1 Overview of the agent

The Calendar Agent interacts both with the calendar management system and with the user in order to assist him. The agent interacts with MS Outlook, as shown in Figure 6.1. Calendar Agent was implemented in Java and its interfaces were implemented using the functionalities provided by MS Agent6.

There are several ways in which an agent can interact with its user: the agent can notify or alert the user about situations relevant to him; the agent can suggest the user some event parameters or the execution of actions; the user can ask the agent to perform some task on his behalf or he can ask it for advice; the agent can request some information from the user or can ask him to make a decision; the agent

http://www.microsoft.com/msagent
can execute actions autonomously, notifying the user afterwards.

For example, the agent can assist users in the following ways. The agent can suggest the user the date, the time, the duration and the place for an event given the type of event and the organizer. This assistance can be provided upon the user’s request or when the agent detects the user is scheduling a new event. The agent can alert the user about two or more overlapping events, it can propose the user a different event date or time for one of the overlapping events, and it can change one of the events on the user’s behalf, notifying him afterwards.

To assist a user, the Calendar Agent has to learn the user’s scheduling preferences, such as the type of events he attends, when he schedules each type of event, which events are more important than others (priorities), the places where events take place, and information about his contacts. The agent learns by observing the user’s behavior while he uses the calendar management system, through information the user explicitly provides, and by the feedback the user gives after the agent assists him. Figure 6.2 shows an overview of the Calendar Agent functionality. We can observe that this functionality is a materialization of our proposed approach, which was sketched in Figure 3.2 in Chapter 3.

6.2.2 Learning user profiles in the calendar management domain

To learn when and where the user prefers to schedule the different types of events, the agent records each new event operation the user executes. Each record contains the characteristics of an event scheduled by the user. In turn, the agent records the
feedback the user provides when it suggests him when and where to schedule a new event. The file containing this information is processed by the learning algorithm and a set of hypotheses about the user’s preferences is created.

To acquire information about the user’s scheduling preferences our agent uses a user profiling algorithm based on association rules. To obtain these hypotheses, the algorithm first generates association rules from the input file. The post-processing steps are similar to those of WATSON and IONWI. The algorithm first selects those association rules that are relevant to the agent’s learning goals, and, then, it eliminates redundant and contradictory rules. The hypotheses are generated from the rules surviving the filtering process. The hypotheses constitute the standard user profile. The agent uses this profile to assist the user by, for example, suggesting a meeting date for an event the user is scheduling.

The association rules our agent is interested in are the ones included in the following template:

- event type, [event priority], [event host] → date, [place], [time], [duration]

Redundant and contradictory rules are filtered out in the same way as in the WATSON and IONWI algorithms, but without considering any domain information to filter redundant rules.

To learn how the user deals with overlapping events, the agent records the user’s actions when an overlapping between two or more events is informed by the agent. Besides, the agent records the user’s feedback when it suggests the user how to reschedule the overlapping events. Each record contains the information of the two overlapping events together with the action the user executed, i.e. which event he changed and the new event information. The information is processed using association rules in the same way as in the previous example. In this case, the association rules our agent is interested in are:


- event one data, event two data → action, [new event data]
- event one data → action, new event data
- event two data → action, new event data

where “event one data” and “event two data” can be any combination of attributes describing the events in conflict, the “action” may be a change in one of the events or enabling the overlapping. In the former case, “new event data” contains the features the user changed.

### 6.2.3 Personalizing user-agent interaction

The *Calendar Agent* has the capability of providing different types of assistance actions depending on the user’s preferences. In turn, this assistance can be provided with different modalities, that is interrupting the user or not. We have focused our attention on three situations in which the agent can assist the user:

- The user is scheduling a new event. The agent can suggest the user some meeting parameters or it can autonomously set these parameters on the user’s behalf.

- The user is scheduling an event that overlaps with a previously scheduled event. The agent can warn the user about the inconvenient, it can suggest the user which event to reschedule and for when, or it can reschedule one of the events on the user’s behalf.

- The user receives an invitation to take part in an event. The agent can just warn the user, it can suggest the user to accept or reject the invitation, and it can reply to the invitation on the user’s behalf.

Figure 6.3 shows how the agent helps the user to schedule a new event. The user has indicated that he wants to schedule a dinner with his friends. The agent will suggest the event date and place, once the user has indicated which friends will take part in the event.

Regarding how the assistance is provided, if the assistance action is a warning or a suggestion, the agent can interrupt the user or it can send him a notification without interrupting him. We achieved this by using different user-interfaces to assist the user. Figure 6.4 shows an example of an interruption. In this example, the agent suggests the user to accept an invitation to take part in an event. Figure 6.5 shows how the agent can be hidden in the bar, not to disturb the user. In
the former case, the user’s work is interrupted and he has to pay attention to the message shown in the user-interface. In the latter case, the user decides when to pay attention to the message the agent wants to give him and he can continue with his work until he decides to stop it to pay attention to the notification.

To learn how to assist a user in a given situation, the Calendar Agent combines three learning algorithms: the user profiling algorithm that learns the user’s preferences with respect to calendar management, the WATSON algorithm, and the IONWI algorithm. The user profile built with these algorithms is used with the corresponding decision making algorithms to decide which assistance action to execute, the contents of the assistance (that is, what to suggest in the case of a suggestion or what to do in the case of an action on the user’s behalf) action and its modality.
6.3 User Profiling Toolkit

In this section we will describe the *User Profiling Toolkit*, which enables us to make experiments and evaluate user profiling algorithms. The aim of this tool is aiding agent developers and mainly user profiling algorithm developers to the evaluation and experimentation with these software pieces. In this work, we used the *User Profiling Toolkit* to carry out different experiments with *IONWI*, *WATSON*, and the decision making algorithms associated with them. We will describe the functionality of the tool with respect to this issue.

6.3.1 Toolkit Overview

As we have said, *User Profiling Toolkit* is inspired in some tools commonly used in the Machine Learning and Data Mining areas to experiment with different algorithms. Particularly, we based our tool in WEKA [Witten and Frank, 2000] and we have used the WEKA API to implement parts of the tool. The functionality of our tool includes the following tasks:

- Set the input for the user profiling algorithm
- Pre-process the input for the user profiling algorithm
- Set the user profiling algorithm and its parameters
- Set the parameters for the machine learning algorithm the user profiling algorithm is based on (Apriori in our example)
- Run the user profiling algorithm
- Visualize the user profile
- Visualize intermediate results
- Save the user profile for future use
- Set the decision making algorithm and its parameters
- Set the situations to evaluate the performance of the profiling algorithm, that is, to run the decision making algorithm
- Run the decision making algorithm
- Visualize the decisions and the explanations for the decisions

The following subsections explain in detail each of the functionalities mentioned above.
6.3. User Profiling Toolkit

6.3.2 Selecting input files

The User Profiling Toolkit enables users to select an input file containing a set of observations. In the case of the IONWI and WATSON algorithm these observations are user-agent interaction experiences. In the case of regular profiling algorithms these observation will be actions performed by the user with the computer application, such a new event tasks. The input file must be in the .arff (attribute-relation file format) format. An arff file is an ASCII text file that describes a list of instances sharing a set of attributes. Arff files have two distinct sections. The first section is the header information, which is followed by the data information. The header of the arff file contains the name of the relation, a list of the attributes (the columns in the data), and their types. The arff data section of the file contains the data declaration line and the actual instance lines. Each instance is represented on a single line, with carriage returns denoting the end of the instance. Attribute values for each instance are delimited by commas. They must appear in the order that they were declared in the header section. An example of an arff input file for WATSON is shown in Figure 6.6.

Once the tool has loaded the file, it displays information about the file, such as the number of instances, the number of attributes each experience contains, the names of the attributes, the values each attribute can take and some statistics about these attributes. In turn, the user can filter out some attributes he does not want the algorithm to process. Figure 6.7 shows the user-interface through which a user can set the input for a profiling algorithm and pre-process it.

---

7See Appendix B.
6.3. User Profiling Toolkit

6.3.3 Selecting and running a profiling algorithm

The User Profiling Toolkit enables us to select a user profiling algorithm. There are two types of algorithms according to this tool: batch algorithms and incremental algorithms. A batch algorithm generates a user profile from a database or file containing user-agent interaction experiences or user-application interaction experiences. If we want to update the database, then the algorithm has to be rerun for the profile to be updated. An incremental algorithm enables us to gradually update the user profile as new information is obtained. To achieve this goal, we have to provide two additional input files: a file containing the information added to the original data file (called increment) and a file containing the information deleted from the original data file (called decrement). The user profile is built considering the three input files.

Each profiling algorithm (both batch and incremental) has a set of parameters the user has to specify. If the user does not specify them, default values are assigned to the parameters. For example, for the WATSON algorithm, the user has to specify the values for \( \alpha, \beta, \gamma, \delta, \) and \( \theta \). The user can also indicate whether he wants to visualize just the resulting user profile or all the intermediate results (the different filtering processes in WATSON and IONWI). Figure 6.8 shows the parameters a user must set to run the WATSON algorithm.

Most user profiling algorithms are based on a certain machine learning technique, such as association rules, decision trees, Bayesian classifiers, among others. Thus, the user of the User Profiling Toolkit can select the algorithm (we called them basic
6.3. User Profiling Toolkit

The profiling algorithm is based on, in order to set the parameters for this algorithm. For example, the basic algorithm for \textit{WATSON} is Apriori. The user can then set the values for the minimum support, the minimum confidence, the number of rules to be generated, among others. This task is optional, since each user profiling algorithm sets the parameters for the basic algorithm. The basic algorithms our tool provides are those implemented in WEKA [Witten and Frank, 2000]. Figure 6.9 shows the user interface through which the user can set the parameters for the machine learning algorithm.

Once the parameters for the user profiling and the basic algorithms have been set, the user profiling algorithm can be run by pressing the start button. The results of the profiling algorithm are shown in the right panel of the tool. These results can be merely the components of the user profile or the different steps the algorithm performs during the user profile building task. In the case of \textit{WATSON} and \textit{IONWI}, the different components of the hypotheses composing the user interaction profile are shown. In Figure 6.10 we can observe the components of two hypotheses, namely the main rule, the positive evidence, the negative evidence and the certainty value. The user profile can be saved for its use in future experiments.

The \textit{User Profiling Toolkit} also enables users to visualize the results of the basic machine learning algorithm. Figure 6.11 shows the association rules generated by the Apriori algorithm when running \textit{WATSON} in an experiment.
6.3. User Profiling Toolkit

Figure 6.9: Basic algorithm parameters

Figure 6.10: Running a profiling algorithm
6.3. User Profiling Toolkit

6.3.4 Selecting and running a decision making algorithm

Once the user profile has been generated we need to test how good our approach is at providing assistance to the user in different situations. To simulate this process, User Profiling Toolkit enables users to provide one or more input situations, select a decision making algorithm and evaluate whether the assistance provided by the algorithm considering the user profile is correct or not.

The input situation describes a situation in which the user might need assistance, such as a new event operation or an overlapping between two events. The situation is expressed in an arff file, in the same way as the user-agent interaction experiences. To make the experiments easy to handle, our tool enables the definition of a set of situations and the decision making algorithm provides the results for each of them. Figure 6.22 shows the assistance actions suggested by the WATSON decision making algorithm for two situations.

Like with the case of user profiling algorithms, the decision making algorithms has some parameters the user can set. For example, the WATSON decision making algorithm has three parameters, namely $\tau_1$, $\tau_2$ and $\tau_3$. The user can change these parameters in order to make different experiments. If no values are set by the user, default values are assigned to the parameters.

The tool enables users to test decision making algorithms with the user profile.
Figure 6.12: Decision making
recently built or with an old user profile that can be retrieved from a file. Some
decision making algorithms provide an explanation for their decisions.

In the case of the \textit{IONWI} algorithm (and the same can be applied to other
algorithms), the user can also specify the task the user is carrying out when the
assistance is provided. The user task is specified in an arff file, in the same way
as situations and interaction experiences. The \textit{IONWI} decision making algorithm
considers the use profile, the situation, and the primary user task to decide whether
to interrupt the user or not.

\subsection{6.3.5 Visualizing a user profile}

The tool enables users to visualize the user profile, depending on the visualization
provided by the different profiling algorithms. So far, the \textit{WATSON} and \textit{IONWI}
algorithms only show the user profiles as text, detailing the composition of each
hypothesis, that is, the main rule, the positive evidence and the negative evidence.
Figure 6.13 shows an example of a user profile generated by \textit{WATSON}.

The following section gives details about how we designed and implemented our
profiling algorithms.
6.4 Design and Implementation

In the previous sections we described two applications we developed using our proposed approach. In this chapter we will briefly describe how we designed and implemented these applications. Particularly, we will describe the main software components an interface agent built under our approach must have and the interactions between these components.

Section 6.4.1 presents an overview of the design of an interface agent built using our approach. Section 6.4.2 describes the software components in charge of user profile building. Finally, Section 6.4.3 briefly describes how we implemented these components.

6.4.1 Overview

In order to achieve the personalization goals proposed by our approach, an interface agent must have various capabilities: observing the user’s behavior while he is working with the computer application; learning the user’s preferences and habits with respect to the computer application from this observation; observing the user’s interactions with the agent; learning the user’s assistance requirements and interruption preferences; deciding when and how to assist the user; and executing actions (warnings, suggestions, actions on the user’s behalf) to provide assistance to the user. These capabilities distinguish an interface agent built under our approach from the current available interface agents that do not personalize their interaction with users. Figure 6.14 shows the main components and connectors of our proposed approach, which provide an interface agent with the functionality described before.

To learn about the user’s habits and preferences with respect to the computer application he is working with, the Observation component observes the user’s be-
behavior while he is working with this application. The Observation component, as well as the Action component, depends on the underlying computer application. Thus, these tasks cannot be prescribed by our user profiling architecture. We do not force the developer to use a specific design for those components. However, we define some capabilities these components must possess.

The information obtained by the Observation component is processed by the Standard Profile Builder component to build the Standard User Profile. These components can be found in most interface agents, since building a user profile is a required capability for these agents. One of the functionalities that distinguishes our approach from others is the ability of the Observation component to observe and record each user agent interaction. The data obtained from this observation is sent to the User Agent Interaction Profile Builder component, which builds the User Interaction Profile. Both the Standard User Profile and the User Interaction Profile are subcomponents of the User Profile. The User Profile is used to decide how to assist the user when a certain problem situation or situation of interest occurs. The Deliberation component is in charge of taking this decision, which in our approach comprises not only the type of assistance action and the contents of the assistance action, but also its modality. The decision is communicated to the Action component, which is in charge of executing it. This action may be a suggestion, a warning, or an action on the user’s behalf executed directly on the computer application. After the assistance is provided, the Observation component captures implicit and explicit user feedback which is recorded and sent to the Profile Builder components.

Figure 6.15 presents a more detailed view of the components of our interface agents. In this figure, we can observe that the Observation component collects two different types of user experiences. First, it records the user’s interaction with the computer application and, second, it records a user’s interaction with the interface agent. The user-agent interaction experiences are sent to the User-Agent Interaction Experiences Manager, and the user-application interaction experiences are sent to the User-Application Interaction Experiences Manager. These two components are in charge of representing users’ experiences in the format required by the user profile building components.

The component in charge of building the user interaction profiles comprises two subcomponents: the WATSON Profile Builder component and the IONWI Profile Builder component. Each of them builds one of the components of the user interaction profile. The WATSON Profile Builder component is in charge of obtaining the User Assistance Requirements and the IONWI Profile Builder component is in
6.4. Design and Implementation

Figure 6.15: Detailed view of the components

The Deliberation component comprises four subcomponents. The component in charge of deciding which assistance action the agent should execute is the WATSON Decision Making component. The IONWI Decision Making component decides the modality for the assistance action. To achieve its goal, this component uses the information provided by the User Task Manager about the relationship (the relevance) between the current context and the primary user task. Finally, the Assistance Decision Making component decides the contents of the assistance action, that is, what to suggest in the case of a suggestion, what to warn in the case of a warning and what to do in the case of an action. The resulting decision is a combination of the decisions made by these three components. The decision is communicated to the Action component, which, in turn, is composed by two subcomponents: the Assistance Action Manager and the Assistance Modality Manager. The first subcomponent is in charge of executing the corresponding assistance action, and the second subcomponent is in charge of providing the assistance in the selected modality.

6.4.2 The profile building components

Figure 6.16 shows the components of the WATSON Profile Builder and the IONWI Profile Builder components when using association rules as machine learning tech-
6.4. Design and Implementation

![Diagram of User Interaction Profile Building Components]

Figure 6.16: User interaction profile building components

Both profile building components have the same subcomponents, but they differ in how these subcomponents work to produce the different components of the user interaction profile. The **Association Rule Generator** component is in charge of obtaining a set of association rules from the set of user-agent interaction experiences obtained by the **Observation** component and pre-processed by the **User-Agent Interaction Experiences Manager** component. The association rules are first processed by the **Interest Filtering** component, which selects those rules relevant to the goals of the profile building component. This filtering is done considering templates, as it was described in Chapters 4 and 5. The interesting association rules are then processed by the **Redundant Filtering** component, which filters out the redundant association rules. The redundancy filtering is performed taking into account Shah rules and domain information as it was described in Chapters 4 and 5. Then, those non-redundant association rules are sent to the **Contradictory Filtering** component, which eliminates the contradictory rules according to the definitions provided in Chapters 4 and 5. The association rules that survive these filtering processes are considered by the **Hypotheses Manager** component, which builds the hypotheses about the user’s assistance requirements and interruption preferences. The highly certain hypotheses or facts compose the **User Interaction Profile**.

The components presented do not prescribe an incremental solution for the profiling problem, since association rule mining algorithms are inherently batch. Figure 6.17 shows the profiling components for the incremental versions of **IONWI** and **WATSON**. In the incremental case, we have two new components: the **FUP Itemset Generator** and the **Updating Need Manager**. The first component is in charge of updating the itemsets according to the new interaction experiences. Details about how
Figure 6.17: Incremental profiling components

this is achieved were provided in Chapter 4. The itemsets generated by this component are used by the Association Rule Generator to build the association rules. In the non incremental version, the itemsets are also generated by this component. The Updating Need Manager determines when it is necessary to update the user profile, and when not. When it decides there is a need of an updating, it informs this decision to the FUP Itemset Generator component, which must act accordingly.

6.4.3 Implementation details

Both the User Profiling Toolkit and the Calendar Agent were implemented in the Java programming language. The User Profiling Toolkit uses some Java classes provided by the WEKA API [Witten and Frank, 2000]. Underlying the User Profiling Toolkit, we provide a framework developers can use to add new profiling algorithms to the tool. The profiling algorithms have to be accompanied by the corresponding decision making algorithm. Profiling algorithms can be batch or incremental. Batch algorithms inherit from the BatchAlgorithm class, which in turns inherits from the ProfilingAlgorithm class. Similarly, incremental algorithms inherit from the IncrementalAlgorithm class. These inheritance relationships can be seen in figures 6.18 and 6.19. Users can experiment using our tool with both the incremental and batch versions of the IONWI and WATSON algorithms. The IWatsonNaive class that appears in the diagram is a naive incremental algorithm since it updates the user profile when the size of the new database is bigger than a given threshold value.

A user who wants to add a new algorithm to the tool has to create a class for this algorithm inheriting from the corresponding class. The method that provides (which is abstract in the superclass) the main functionality of the algorithm is called
Figure 6.18: Profiling algorithms hierarchy
Figure 6.19: Incremental algorithms hierarchy
run. The user has to implement it. The profiling algorithm can use any basic algorithm, that is, any machine learning technique. However, the user will have to define what a profile component is if the profile contents are not hypotheses as in our algorithms. This can be done by creating a subclass of ProfileComponent. Figure 6.20 shows the class hierarchy of the components of a user profile. The classes IONWIHypothesis and WATSONHypothesis represent the user assistance requirements and user interruption preferences respectively.

Figure 6.21 shows the class hierarchy for the different filtering methods. The user can add new filtering processes to our algorithms or to new profiling algorithms by inheriting from the RuleFilter class. We have implemented the interest, redundant and contradictory rule filters both for WATSON (classes FContradictorio, Funitecovering, FRedundant) and for IONWI (classes FContradictorioI, FunitecoveringI, FRedundantI). We have also implemented the filters used by the confidence-based algorithm (CFUnitecovering class).

Figure 6.22 shows the class hierarchy for the decision making algorithms. These algorithm are implemented by three classes: DMWatson, DMIONWI and DMConfidenceBased. These classes are subclasses of the abstract class DMAAlgorithm. The
decision making algorithms take the Profile into account to take a decision.

6.5 Summary

In this Chapter we described an application that materializes our proposed personalization approach and an application that enabled us to evaluate our profiling algorithms. The following Chapter describes the experiments we carried out with these applications to evaluate and test the performance of the WATSON and IONWI algorithms and the learning capability of our personalization approach.
Chapter 7

Determining WATSON and IONWI Parameters

In Chapter 4 and Chapter 5 we described the WATSON and IONWI algorithms respectively. In this Chapter we show how we set the values of the different parameters of the two algorithms, and we describe how these parameters affect each algorithm.

7.1 Introduction

Several parameters and threshold values play a significant role within the WATSON and the IONWI algorithms. Some of these parameters are related to the machine learning technique the algorithms use - association rules. These parameters are:

- the support threshold value \( \text{minsup} \)
- the confidence threshold value \( \text{minconf} \)
- the minimum number of rules to be generated by Apriori
- the coefficient \( P \) to determine positive rules when pruning association rules
- the coefficient \( \bar{N} \) to determine negative rules when pruning association rules
- the \( \varepsilon_1 \) and \( \varepsilon_3 \) thresholds to determine whether two rules have similar confidence (strength) when pruning redundant rules in WATSON and IONWI respectively
- the \( \varepsilon_2 \) and \( \varepsilon_4 \) thresholds to determine whether two rules have similar confidence when pruning contradictory rules in WATSON and IONWI respectively
7.2 Coefficients to determine negative and positive rules

- the threshold value used in the DELI algorithm for incremental learning

The rest of the parameters, which are not related to the machine learning technique, are the following:

- the factors \( \alpha, \beta \) and \( \gamma \) used to compute the certainty degree of a hypothesis
- the minimum certainty degree \( \delta \) required in a hypothesis
- the minimum confidence value for a problem situation \( \tau_1 \)
- the minimum confidence value for an action \( \tau_2 \)
- the minimum confidence value for a suggestion \( \tau_3 \)
- the similarity threshold for comparing situations \( \psi \)
- the similarity threshold for comparing tasks and situations \( \phi \).

The following sections describe how we set the values for these parameters.

7.2 Coefficients to determine negative and positive rules

The coefficient \( P \) to determine positive rules when pruning rules is set to 2 and the coefficient \( N \) to determine negative rules when pruning rules is also set to 2, since these values were successfully used in [Shah et al., 1999] where the definition of positive and negative rules was given. These values are applied to both WATSON and IONWI.

7.3 Threshold values used to discover rules with similar confidence

The \( \varepsilon_1 \) and the \( \varepsilon_2 \) thresholds used to tell if two rules have similar strength or confidence are set to 0.06, since this value was used successfully in [Shah et al., 1999] the work defining the pruning rules WATSON and IONWI use. The \( \varepsilon_2 \) and \( \varepsilon_4 \) thresholds used to determine whether two rules have different confidence when pruning contradictory rules are set to 0.80.
7.4 Values for $\text{minsup}$ and $\text{minconf}$

The value of $\text{minsup}$ determines which association rules are generated and which are discarded. A high value will probably make us lose some important association rules. Thus, we have to determine a value for $\text{minsup}$ that enables us to discover those relationships between problem situations and users’ assistance requirements that we need to assist users.

In our context, a transaction database contains assistance experiences of a particular problem situation, for example two overlapping events. The support value indicates the percentage of transactions that contain a given situation-action pair, and it is computed as the ratio of the number of transactions in the database containing the itemset to the number of transactions or instances in the dataset. Suppose that there are $M$ different types of overlapping pairs of events stored in the database and that they are equally probable with probability of occurrence $1/M$. We consider that this value can be used as an approximation for the $\text{minsup}$ value. If there are $N$ transactions in the database, the number of transactions containing each situation-action pair is $N/M$. Then, to obtain the support of the itemset we have to divide this number by $N$. Thus, we obtain $1/M$. Consequently, a $\text{minsup}$ value of at most $1/M$ will be appropriate since in the real world the situations we handle are not equally probable.

To check whether our assertion was true, we tuned the value of $\text{minsup}$ and we analyzed the variation in the number of hypotheses obtained by our algorithms. Figures 7.1 to 7.5 show the number of hypotheses obtained with the $\text{WATSON}$ algorithm for different values of $\text{minsup}$ with different datasets. The points marked in the graphs with a circle indicate the $\text{minsup}$-hypotheses pair for which $\text{minsup}$ is equal to 1 divided by the number of instances in the corresponding dataset. The experiments were carried out with a $\text{minconf}$ value of 0.8 and 1000 association rules generated. The experiments were done with a subset of the datasets used to evaluate the performance of our algorithms. These datasets contain user-agent interaction experiences in different contexts and for different users. The contents of the datasets will be described in detail in Chapter 8.

We can observe that for most of the cases, the number of hypotheses obtained decreases considerably when the value of $\text{minsup}$ is higher than $1/M$, where $M$ is the number of instances in the dataset. Thus, we suggest that in order to obtain good results with the $\text{WATSON}$ algorithm, the value of $\text{minsup}$ should be smaller than $1/M$. This value ranged in our experiments from 0.04 to 0.1.

Figures 7.6 to 7.8 show the number of hypotheses obtained with the $\text{IONWI}$ algorithm for different values of $\text{minsup}$ with different datasets. The experiments
7.4. Values for minsup and minconf

Figure 7.1: minsup values versus Number of hypotheses (WATSON - User 1)

Figure 7.2: minsup values versus Number of hypotheses (WATSON - User 2)

Figure 7.3: minsup values versus Number of hypotheses (WATSON - User 3)
Figure 7.4: \textit{minsup} values versus Number of hypotheses (\textit{WATSON} - User 4)

Figure 7.5: \textit{minsup} values versus Number of hypotheses (\textit{WATSON} - User 5)
7.4. Values for \textit{minsup} and \textit{minconf}

![User 3](image)

Figure 7.6: \textit{minsup} values versus Number of hypotheses (\textit{IONWI} - User 3)

![User 4](image)

Figure 7.7: \textit{minsup} values versus Number of hypotheses (\textit{IONWI} - User 4)

were made with a \textit{minconf} value of 0.8 and 1500 association rules generated.

Like the case of \textit{WATSON}, we can observe that for most of the cases the number of hypotheses obtained decreases considerably when the value of \textit{minsup} is higher than $1/M$, where $M$ is the number of instances in the dataset. This value ranged in our experiment from 0.05 to 0.1. Consequently, a value of \textit{minsup} within this range would be appropriate when running the \textit{IONWI} algorithm.

The value of \textit{minsup} can be dynamically updated as new interaction experiences are recorded. For example, when the \textit{DELI} algorithm signals the need of an update of the user profile, the algorithm can set the value of \textit{minsup} according to the amount of different situations stored in the database. To determine the amount of different situations, we can group situations by similarity using, for example, the equation presented in Chapter 3 to tell whether two situations are similar.

With respect to the value of \textit{minconf}, it should be high since it indicates the probability of failing or succeeding at assisting a user in a given problem situation. Thus, an appropriate initial value is 0.8.
7.5 Parameters of the certainty function

As explained in Chapter 4, the certainty degree of a hypothesis is computed as a function of the supports of the rule originating the hypothesis and the rules considered as positive and negative evidence. We chose this method to compute the certainty degree of a hypothesis because the support of a rule gives us an indication of the usefulness of the rule, that is the amount of interaction experiences that contain the items involved in the rule. Thus, we take into account not only the number of experiences that contain the items in the main rule, but also the experiences containing the positive and the negative evidence. The positive evidence contains those similar situations with the same associated assistance action. Therefore, we sum the support of these rules to the support of the main rule to have a better picture of the usefulness of the hypothesis. Then, we subtract the support of the negative evidence because we want to reflect that those rules indicate that in those situations our hypothesis is not useful.

Three factors or term weights are used to compute the certainty degree of a hypothesis, namely $\alpha$, $\beta$ and $\gamma$. We set these factors to 0.7, 0.15 and 0.15 respectively. We selected these values because we consider the positive evidence equally important as the negative evidence, but we consider that the support of the main rule is more important than the negative and the positive evidence in the certainty calculus.

7.6 Number of rules to be generated by Apriori

One of the parameters of the Apriori algorithm (implemented in WEKA) is the number of rules it must generate. This value has an impact on the number of interesting rules our algorithms select, and on the number of hypotheses these algorithms obtain. If the number of rules is small, then very few hypotheses will be obtained.
Table 7.1: Number of rules versus Number of hypotheses (WATSON)

However, generating huge amounts of rules may be not useful. To obtain the minimum value for this parameter we performed a series of experiments with different datasets. For most of the datasets, we observed the following phenomenon: there is a value for which the number of interesting rules, and hence the number of hypotheses, reaches a maximum. If we increase the number of rules generated, no more new hypotheses are obtained. The parameter value for which this phenomenon occurs varies from dataset to dataset. Table 7.1 shows the different values we obtained for different datasets with the WATSON algorithm. The first column indicates the name of the dataset. The second column indicates the user to whom the dataset belongs. The third column indicates the number of instances the dataset contains. The fourth column indicates the number of rules in which the number of hypotheses reaches its maximum. We varied the number of rules from 50 to 5000. In those cases where no equilibrium was reached, the number of rules is the maximum number of rules used in the experiment. The fifth column lists the number of rules that survived the first filtering step, that is the interesting rules. The sixth column lists the number of hypotheses generated by WATSON. Finally, the last column indicates the average certainty value for these hypotheses.
## 7.6. Number of rules to be generated by Apriori

<table>
<thead>
<tr>
<th>DATASET</th>
<th>USER</th>
<th>No. of Instances</th>
<th>No. of Rules</th>
<th>No. of Int. Rules</th>
<th>No. of Hypotheses</th>
<th>AVG. CERTAINTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>holiday</td>
<td>3</td>
<td>100</td>
<td>1500</td>
<td>37</td>
<td>4</td>
<td>0.3495</td>
</tr>
<tr>
<td>holiday</td>
<td>4</td>
<td>30</td>
<td>No equilibrium (500)</td>
<td>30</td>
<td>4</td>
<td>0.36</td>
</tr>
<tr>
<td>holiday</td>
<td>7</td>
<td>99</td>
<td>500</td>
<td>7</td>
<td>3</td>
<td>0.3196</td>
</tr>
<tr>
<td>new event</td>
<td>3</td>
<td>100</td>
<td>No equilibrium (5000)</td>
<td>62</td>
<td>6</td>
<td>0.3233</td>
</tr>
<tr>
<td>new event</td>
<td>4</td>
<td>99</td>
<td>800</td>
<td>4</td>
<td>1</td>
<td>0.2772</td>
</tr>
<tr>
<td>new event</td>
<td>7</td>
<td>100</td>
<td>700</td>
<td>12</td>
<td>6</td>
<td>0.3249</td>
</tr>
<tr>
<td>overlapping</td>
<td>3</td>
<td>92</td>
<td>No equilibrium (5000)</td>
<td>27</td>
<td>6</td>
<td>0.2628</td>
</tr>
<tr>
<td>overlapping</td>
<td>4</td>
<td>100</td>
<td>800</td>
<td>3</td>
<td>3</td>
<td>0.1446</td>
</tr>
<tr>
<td>overlapping</td>
<td>7</td>
<td>102</td>
<td>500</td>
<td>3</td>
<td>2</td>
<td>0.2529</td>
</tr>
<tr>
<td>time</td>
<td>3</td>
<td>106</td>
<td>800</td>
<td>19</td>
<td>12</td>
<td>0.2815</td>
</tr>
<tr>
<td>time</td>
<td>4</td>
<td>100</td>
<td>800</td>
<td>13</td>
<td>8</td>
<td>0.2811</td>
</tr>
<tr>
<td>time</td>
<td>5</td>
<td>125</td>
<td>No equilibrium (5000)</td>
<td>4</td>
<td>2</td>
<td>0.2767</td>
</tr>
<tr>
<td>time</td>
<td>7</td>
<td>94</td>
<td>300</td>
<td>7</td>
<td>6</td>
<td>0.1590</td>
</tr>
<tr>
<td>Average</td>
<td>98</td>
<td>744</td>
<td>150</td>
<td>110</td>
<td>5</td>
<td>0.2685</td>
</tr>
</tbody>
</table>

Table 7.2: Number of rules versus Number of Hypotheses (IONWI)

The experiment was carried out with fixed values for $\alpha = 0.7, \beta = 0.15, \gamma = 0.15$, $\text{minsup}=0.1, \text{minconf}=0.8$ and $\delta = 0.1$. The last row shows the average values for the different columns.

As a result of this experiment, we decided to set the minimum number of rules to be generated by the Apriori algorithm in 500\(^1\). We can observe that the average number of interesting rules is approximately 37 and the average number of hypotheses generated is approximately 13. The average certainty value for the hypotheses generated by WATSON is approximately 0.30.

Table 7.2 shows the results obtained when tuning the number of rules in the Apriori algorithm within the IONWI algorithm from 50 to 5000.

As a result of the experiment with the IONWI algorithm, we decided to set the minimum number of rules to be generated by the Apriori algorithm in 750. We can observe that the average number of interesting rules is approximately 18 and the average number of hypotheses generated is approximately 5. The average certainty value for the hypotheses generated by IONWI is approximately 0.27. As the reader can observe, there are significant differences between the results obtained with IONWI and the results obtained with WATSON.

---

\(^1\)This value is a minimum. One should select a higher value to guarantee that a considerable amount of interesting rules and hypotheses will be generated. We set this value in 1000 for WATSON and 1500 for IONWI.
7.7. Value of the certainty threshold $\delta$

One of the key aspects in the decision making process is the value that the threshold $\delta$ takes, since it determines which hypotheses are turned into facts and which not. According to the figures shown in Table 7.1, we decided to set the value of $\delta$ in 0.20 for the WATSON algorithm, since the average certainty value for most of the hypotheses obtained with WATSON is greater than this number. Similarly, the certainty threshold for the IONWI algorithm is set to 0.20 according to the average certainty values in Table 7.2.

7.8. Number of instances in the dataset

A major issue in our approach is the number of user-agent interaction experiences that are required so that our algorithm can produce a useful output. In other words, we need to know how many user-agent interaction experiences an agent needs to start learning about the user.

Our algorithms are based on association rules, which come from the data mining area. In this area databases are huge; thus, we wanted to test whether this was a prerequisite for our approach. Consequently, we run the WATSON and the IONWI algorithms with different sizes of datasets and we analyzed the amount of rules and hypotheses generated. Figure 7.9 and 7.10 show the results we obtained.

We run the WATSON algorithm with four types of datasets, each containing user-agent interactions from 5 different users. For each user-dataset type combination, we run the algorithm varying the number of instances in the file from 10
7.9. Parameters for incremental learning

Figure 7.10: Number of instances versus Number of hypotheses (IONWI)

to 100. The graph in Figure 7.9 shows the average number of hypotheses given the
different number of instances for the four dataset types. We can observe that for
most cases the number of hypotheses increases as the number of instances increases.
We can also observe that with few interaction experiences the algorithm produces
a useful output, that is the agent has learned something about the user. According
to our experiment, the number of hypotheses generated in the early learning stages
varied from 3 to 11. This means that our approach does not require huge amounts
of data to work. However, the bigger the datasets the better for the agent since it
will learn more about the user. From an statistical point of view, the number of
instances should be at least 30.

The same experiment was done with the IONWI algorithm. We can observe
in Figure 7.10 that, in average, the number of hypotheses increases as the number
of instances increases. We can also observe that with few interaction experiences
the algorithm produces a useful output, as it occurs with WATSON. The number of
hypotheses obtained with IONWI is smaller than the number of hypotheses obtained
with WATSON. The number of hypotheses obtained in the early learning stages
varied from 1 to 6. The figure shows that the learning capability of the algorithm
with few instances is good.

In this experiment we have not analyzed the quality of the hypotheses generated.
This issue is studied is the following Chapter.

7.9 Parameters for incremental learning

In Chapter 4 we introduced the FUP2 and the DELI algorithm, which are used to
incrementally update the association rule set and to determine when to update it,
respectively. The DELI algorithm determines when to update the association rule
set by computing the $\frac{|L \cap L'|}{|L|}$ ratio and comparing it against a threshold value. In the
previous formula $L$ is the number of frequent itemsets in the original user - agent
interaction experience database and $L'$ is the number of frequent itemsets in the new
database. In this section we present the results of an experiment we carried out to
determine the appropriate value for the threshold.

To determine the value of this parameter we run the incremental WATSON
algorithm with different sizes of the increment database, fixing the original database
size. We studied the value of the ratio $\frac{|L \cap L'|}{|L|}$ and the number of hypotheses generated
by the algorithm in relation to the number of hypotheses associated with the original
database. Table 7.3 shows the results we have obtained. The columns in this table
indicate: the user, the dataset, the original database size, the number of frequent
itemsets found for the original database, the number of hypotheses obtained for
this database, the increment size, the number of frequent itemsets obtained for the
new database, the number of itemsets in the symmetric different between the old
frequent itemsets and the new ones, the ratio $\frac{|L \cap L'|}{|L|}$, and the number of hypotheses
generated for the new database. The experiments was carried out with the following
parameters: minsup=0.1, minconf=0.8, number of rules=1000.

The figures in italic in Table 7.3 are the ratio values for which the number of new
hypotheses is considerably different (greater in the general case) from the original
number of hypotheses. In most cases, this ratio value is approximately 1. Thus, this
is the maximum value our threshold can take.

The appropriate value for the threshold depends on how many hypotheses the
agent designer is ready to loose; that is, it is a tradeoff between the hypotheses
obtained (some can be obsolete and others are lost) and the overhead produced
when running the incremental algorithm. For example, if we want to loose at most
two hypotheses then the threshold value should be approximately 0.5, according to
the results we obtained. This low threshold value will require more updates than a
threshold value of 1, for example. Since the computational times of the algorithms
are small, we suggest considering low threshold values.

The same experiment was performed with the IONWI algorithm. Tables 7.4
and 7.5 show the results we have obtained when varying the size of the increment
database. The experiments was carried out with the following parameters: min-
sup=0.1, minconf=0.8, number of rules=1500.

As in the previous table, the figures in italic in Tables 7.4 and 7.5 are the ratio
values for which the number of new hypotheses is considerably different from the
original number of hypotheses. The average value of this ratio is 0.7 for the IONWI
algorithm. Thus, this is the maximum value our threshold can take. Then, the agent
<table>
<thead>
<tr>
<th>USER</th>
<th>DATASET</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>holiday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>holiday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>holiday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>holiday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>overlapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>overlapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>overlapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>overlapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>overlapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>overlapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>overlapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>overlapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>holiday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>holiday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>holiday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>holiday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>holiday</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>new event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>overlapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>overlapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>overlapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>overlapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.3: Ratio values for different sizes of the increment database (WATSON)
| User | Dataset | |DB| | |L| | Hypotheses | |DB| | |L| | Hypotheses | |DB| | |L| | Hypotheses |
|------|---------|----------------|----------------|------|----------------|----------------|------|----------------|----------------|------|----------------|----------------|------|----------------|----------------|------|----------------|----------------|
| 3    | holiday | 50 126 | 5 | 10 | 129 | 23 | 0.1825 | 3 |
| 3    | holiday | 50 126 | 5 | 20 | 116 | 44 | 0.3493 | 3 |
| 3    | holiday | 50 126 | 5 | 30 | 167 | 79 | 0.6269 | 5 |
| 3    | holiday | 50 126 | 5 | 40 | 244 | 156 | 1.2380 | 5 |
| 3    | holiday | 50 126 | 5 | 50 | 310 | 226 | 1.7036 | 6 |
| 3    | new event | 50 107 | 5 | 10 | 103 | 38 | 0.3551 | 6 |
| 3    | new event | 50 107 | 5 | 20 | 245 | 200 | 1.8604 | 4 |
| 3    | new event | 50 107 | 5 | 30 | 225 | 200 | 1.8604 | 4 |
| 3    | new event | 50 107 | 5 | 40 | 237 | 216 | 2.0186 | 4 |
| 3    | new event | 50 107 | 5 | 50 | 300 | 278 | 2.5081 | 4 |
| 3    | time | 50 447 | 0 | 10 | 427 | 170 | 0.2603 | 0 |
| 3    | time | 50 447 | 0 | 20 | 205 | 300 | 0.0711 | 3 |
| 3    | time | 50 447 | 0 | 30 | 230 | 339 | 0.7583 | 0 |
| 3    | time | 50 447 | 0 | 40 | 243 | 402 | 0.8903 | 1 |
| 3    | time | 50 447 | 0 | 50 | 290 | 408 | 1.0490 | 7 |
| 3    | overlapping | 50 736 | 1 | 10 | 450 | 324 | 0.4402 | 1 |
| 3    | overlapping | 50 736 | 1 | 20 | 450 | 306 | 0.4157 | 0 |
| 3    | overlapping | 50 736 | 1 | 30 | 327 | 503 | 0.6834 | 4 |
| 3    | overlapping | 50 736 | 1 | 40 | 334 | 562 | 0.7035 | 4 |
| 4    | new event | 50 206 | 0 | 10 | 169 | 103 | 0.5000 | 0 |
| 4    | new event | 50 206 | 0 | 20 | 175 | 123 | 0.5970 | 4 |
| 4    | new event | 50 206 | 0 | 30 | 130 | 202 | 0.0805 | 3 |
| 4    | new event | 50 206 | 0 | 40 | 134 | 208 | 1.0007 | 4 |
| 4    | new event | 50 206 | 0 | 50 | 141 | 215 | 1.0456 | 4 |
| 4    | time | 50 165 | 6 | 10 | 165 | 74 | 0.4484 | 6 |
| 4    | time | 50 165 | 6 | 20 | 143 | 134 | 0.8121 | 2 |
| 4    | time | 50 165 | 6 | 30 | 133 | 174 | 1.0045 | 2 |
| 4    | time | 50 165 | 6 | 40 | 164 | 217 | 1.3151 | 2 |
| 4    | time | 50 165 | 6 | 50 | 191 | 248 | 1.5030 | 6 |
| 4    | overlapping | 50 268 | 3 | 10 | 213 | 96 | 0.3544 | 5 |
| 4    | overlapping | 50 268 | 3 | 20 | 158 | 135 | 0.5037 | 5 |
| 4    | overlapping | 50 268 | 3 | 30 | 177 | 187 | 0.6977 | 1 |
| 4    | overlapping | 50 268 | 3 | 40 | 159 | 189 | 0.7062 | 1 |
| 4    | overlapping | 50 268 | 3 | 50 | 152 | 210 | 0.7835 | 1 |
| 7    | holiday | 50 138 | 3 | 10 | 118 | 34 | 0.2403 | 3 |
| 7    | holiday | 50 138 | 3 | 20 | 106 | 64 | 0.9537 | 4 |
| 7    | holiday | 50 138 | 3 | 30 | 105 | 72 | 0.5217 | 4 |
| 7    | holiday | 50 138 | 3 | 40 | 133 | 117 | 0.8478 | 4 |
| 7    | holiday | 50 138 | 3 | 50 | 134 | 134 | 0.0710 | 5 |
| 7    | new event | 50 314 | 3 | 10 | 212 | 158 | 0.5031 | 3 |
| 7    | new event | 50 314 | 3 | 20 | 155 | 225 | 0.7169 | 5 |
| 7    | new event | 50 314 | 3 | 30 | 142 | 242 | 0.7707 | 5 |
| 7    | new event | 50 314 | 3 | 40 | 145 | 257 | 0.8184 | 5 |
| 7    | new event | 50 314 | 3 | 50 | 144 | 264 | 0.8407 | 6 |

Table 7.4: Ratio values for different sizes of the increment database (IONW1)


| User | Dataset | |DB| | |Hypotheses | |DB| | |Hypotheses | |DB| | |Hypotheses |
|------|---------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7    | overlapping | 50 | 293 | 1 | 10 | 183 | 121 | 0.4000 | 4 |
| 7    | overlapping | 50 | 293 | 1 | 20 | 173 | 143 | 0.4708 | 6 |
| 7    | overlapping | 50 | 293 | 1 | 30 | 150 | 172 | 0.5771 | 5 |
| 7    | overlapping | 50 | 293 | 1 | 40 | 148 | 178 | 0.5979 | 6 |
| 7    | overlapping | 50 | 293 | 50 | 50 | 150 | 182 | 0.6107 | 5 |
| 7    | time      | 50 | 164 | 6 | 10 | 128 | 64 | 0.2002 | 8 |
| 7    | time      | 50 | 164 | 6 | 20 | 108 | 102 | 0.2119 | 6 |
| 7    | time      | 50 | 164 | 6 | 30 | 102 | 128 | 0.7804 | 6 |
| 7    | time      | 50 | 164 | 6 | 40 | 107 | 137 | 0.8253 | 8 |

Table 7.5: Ratio values for different sizes of the increment database (IONWI)

designer should analyze how many new hypotheses (or old invalid hypotheses) he is ready to loose, and set the threshold value accordingly.

### 7.10 Minimum confidence values for different assistance actions

As explained in Chapter 3, when the agent does not have information in the user interaction profile to assist a user in a given situation, it uses the confidence-based algorithm to make a decision about the type of assistance to provide. This confidence-based algorithm has two threshold values: tell-me threshold and do-it threshold. We decided to set the tell-me threshold in 0.3 and the do-it threshold in 0.8, since these are the values used in [Maes, 1994] and [Kozierok and Maes, 1993] where the algorithm was proposed.

When the agent has information about the user’s assistance requirements and interruption preferences, it also considers the confidence on the action content to take a decision. The algorithm has three thresholds: \( \tau_1 \), \( \tau_2 \), and \( \tau_3 \). \( \tau_1 \) tells the agent if the situation is worth handling, \( \tau_2 \) is similar to the do-it threshold, and \( \tau_3 \) is similar to the tell-me threshold. We have set the \( \tau_1 \) threshold in 0.1, the \( \tau_2 \) threshold in 0.8, and the \( \tau_3 \) threshold in 0.3. We consider that at least 10% of the situations the agent has managed have to be similar to the situation at hand in order to handle it. For the other two thresholds, we have chosen the same values than for the thresholds of the confidence-based algorithm, since they have similar meanings.
7.11 Similarity thresholds for comparing two situations

As described in Chapter 3, when deciding how to assist the user, if the agent does not find a profile item containing the situation at hand it looks for a similar situation. To determine if two situations are similar we have defined a similarity function. However, the comparison of situations is domain dependent. Each attribute can have a different similarity function associated with it and a different weight in the function. We have only set the similarity threshold for the situations we have studied. We consider that two situations are similar if they share 60 per cent of their attributes (applying the percentage to the situation with the smallest number of attributes), and if they have exactly the same values in the most important attributes. For example, when comparing events the attribute “event type” is the most relevant. Thus, two situations will be similar if they have the same event type and if the 60% of their attributes are similar.

Similarly, we have set the similarity threshold used when comparing situations and user tasks within the IONWI algorithm. We consider that the task and the situation must have the 60 percent of their attributes in common.

The agent designer can set different threshold values when defining the similarity functions for the situations and tasks that appear in the application domain he is working with.

7.12 Summary

In this Chapter we have described the experiments we carried out to set the values of the parameters of the WATSON and IONWI algorithms. The values obtained for the different parameters should be considered as suggested values. However, the users of our algorithm can set the values they consider appropriate.

The following Chapter describes the experiments we carried out to evaluate the performance of our algorithms.
Chapter 8

Experimental Evaluation

In Chapter 4 we presented the WATSON algorithm, which learns the assistance action a user prefers in different situations or contexts. In Chapter 5 we described the IONWI algorithm, which learns when an agent can interrupt the user to provide him assistance and when not. In Chapter 7 we described how we set the different parameters for these algorithms. In this Chapter we describe the experiments we carried out to evaluate the performance of these algorithms and their ability to learn a user’s assistance requirements and interruption preferences.

8.1 Introduction

To evaluate the ability of the WATSON algorithm to learn a user’s assistance requirements and to provide users the type of assistance they expect, we carried out a series of experiments using the User Profiling Toolkit, which was described in Chapter 6. Similarly, we used this tool to evaluate the capability of the IONWI algorithm of learning a user’s interruption preferences.

We made a set of controlled experiments, simulating the interaction between users and agents with the tool under controlled situations. This kind of controlled experiments enabled us to test the performance of our algorithms in different scenarios, that is with different combinations of parameters, different characteristics of datasets, and different user behaviors. Evaluating learning algorithms using a tool like the User Profiling Toolkit is a common practice in the Machine Learning community. User profiles or user models can be tested for accuracy separately from their use in systems and agents, such as in [Corbett et al., 1993] and [Brusilovsky and Eklund, 1998].

This Chapter is organized as follows. Section 8.2 describes the data we used to carry out the experiments. Then, Sections 8.3 and 8.4 describe the experiments
we carried out to evaluate the performance of the *WATSON* algorithm. We made two kinds of experiments. First, we evaluated *WATSON* ability to learn a user’s assistance requirements. To achieve this goal, we compared the user assistance requirements obtained by the algorithm against those indicated by users. The results we obtained are shown and discussed in Section 8.3. Second, we compared the precision of the *WATSON* decision making algorithm against the confidence-based decision making algorithm. That is, we compared the results of assisting a user with *WATSON* and without *WATSON*. This experiment is described in Section 8.4.

Section 8.5 describes the experiments we carried out to evaluate the performance of the *IONWI* algorithm. We evaluated *IONWI* ability to learn a user’s interruption preferences. To achieve this goal, we compared the user interruption preferences obtained by the algorithm against those explicitly indicated by users. Finally, Section 8.6 presents the conclusions we draw out from the experiments.

### 8.2 User Profiles and Datasets

#### 8.2.1 User profiles and datasets for *WATSON*

To carry out the experiments with *WATSON*, we used 39 datasets containing user-agent interaction experiences in the calendar management domain. Each database record is composed of the attributes that describe the problem situation or situation of interest originating the interaction, the assistance action the agent executed, the user feedback and the evaluation of the interaction experience.

The datasets containing experiences in the calendar management domain were built according to real user profiles, that is, according to a certain set of interaction preferences established by real users. The profiles were obtained by interviewing a set of 12 users. These users are a subset of those who participated in the experiment reported in Chapter 2 that were able to state their preferences with respect to user-agent interaction. We asked each user to indicate which assistance action he/she prefers in different situations and in different instances of a particular situation. We then used this information to build artificial datasets containing the user-agent interaction experiences. The artificial datasets reflect the agent’s behavior. At early interaction stages, when the agent has few information about the user it only makes warnings and, eventually, suggestions. Then, when agent and user have interacted for some time the agent provides the type of assistance expected by the user, provided that the confidence on the action is high enough.

The four situations we analyzed in the calendar management domain are:
8.2. User Profiles and Datasets

- The user is scheduling a new event, and the agent has information about the potential time, place and/or duration.
- The user is scheduling an event that overlaps with a previously scheduled event.
- The time to go from the place where the event being scheduled takes place to the location where the next event takes place is not enough.
- The user is scheduling a business (or work-related) event for a holiday.

The names of the datasets are new event, overlapping, time, and holiday, respectively. Each dataset contains user-agent interaction experiences for the type of situation indicated by its name. Table 8.1 describes the main features of the different datasets containing user-agent interaction experiences. We have different amounts of datasets for the different users, depending on the information they gave us through the questionnaire. Appendix D contains the description of the different user profiles we used to build the datasets1.

The attributes describing the overlapping and time datasets are the following:

- event type (1): It describes the type of one of the overlapping events (the first event is the oldest one). It can take one of about 15 predefined values such as business meeting, class, exams, party, and gym class2.
- host (1): It describes the organizer of the first event. It can take one of a set of about 10 predefined values, which categorize the host in friend, family, work-mate, and so on.
- topic (1): It describes the topic of the first event. It can take one of a set of about 15 predefined values, such as projects, thesis, course, and birthday.
- participants (1): It describes the participants that take part in the first event. It can take one of about 15 predefined values that categorize the participants in friends, wok-mates, employees, family, and so on.
- place (1): It describes the place where the first event takes place. It takes a value from a set of about 10 predefined values, such as home, office, university, and cinema.

1The datasets we used are available at http://www.exa.unicen.edu.ar/~schiia.
2Concept hierarchies or ontologies can be used to defined the values for each attribute.
### 8.2. User Profiles and Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>User</th>
<th>N(^0) of Attributes</th>
<th>N(^0) of Instances</th>
<th>Missing Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>1</td>
<td>13</td>
<td>32</td>
<td>no</td>
</tr>
<tr>
<td>holiday</td>
<td>1</td>
<td>11</td>
<td>56</td>
<td>no</td>
</tr>
<tr>
<td>overlapping</td>
<td>1</td>
<td>15</td>
<td>39</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>2</td>
<td>11</td>
<td>58</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>2</td>
<td>9</td>
<td>35</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>2</td>
<td>13</td>
<td>37</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>3</td>
<td>13</td>
<td>75</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>3</td>
<td>16</td>
<td>106</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>3</td>
<td>11</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>new event</td>
<td>3</td>
<td>11</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>4</td>
<td>10</td>
<td>101</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>4</td>
<td>13</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>4</td>
<td>16</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>5</td>
<td>10</td>
<td>120</td>
<td>yes</td>
</tr>
<tr>
<td>new event</td>
<td>5</td>
<td>10</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>5</td>
<td>15</td>
<td>96</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>5</td>
<td>15</td>
<td>125</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>7</td>
<td>10</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>new event</td>
<td>7</td>
<td>10</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>7</td>
<td>13</td>
<td>102</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>7</td>
<td>16</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>8</td>
<td>15</td>
<td>80</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>8</td>
<td>15</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>8</td>
<td>10</td>
<td>89</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>10</td>
<td>9</td>
<td>30</td>
<td>no</td>
</tr>
<tr>
<td>overlapping</td>
<td>10</td>
<td>11</td>
<td>30</td>
<td>no</td>
</tr>
<tr>
<td>time</td>
<td>10</td>
<td>13</td>
<td>31</td>
<td>no</td>
</tr>
<tr>
<td>holiday</td>
<td>10</td>
<td>10</td>
<td>89</td>
<td>yes</td>
</tr>
<tr>
<td>new event</td>
<td>10</td>
<td>10</td>
<td>90</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>16</td>
<td>15</td>
<td>88</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>16</td>
<td>15</td>
<td>88</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>28</td>
<td>15</td>
<td>25</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>28</td>
<td>12</td>
<td>30</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>30</td>
<td>11</td>
<td>31</td>
<td>no</td>
</tr>
<tr>
<td>holiday</td>
<td>30</td>
<td>9</td>
<td>34</td>
<td>no</td>
</tr>
<tr>
<td>time</td>
<td>30</td>
<td>13</td>
<td>32</td>
<td>no</td>
</tr>
<tr>
<td>overlapping</td>
<td>31</td>
<td>11</td>
<td>47</td>
<td>no</td>
</tr>
<tr>
<td>holiday</td>
<td>31</td>
<td>9</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td>time</td>
<td>31</td>
<td>13</td>
<td>58</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 8.1: Characteristics of the datasets for *WATSON*
8.2. User Profiles and Datasets

- event type (2): It describes the type of the second event.
- host (2): It describes the organizer of the second event.
- topic (2): It describes the topic of the second event.
- participants (2): It describes the participants that take part in the second event.
- place (2): It depicts the place where the second event takes place.
- agent action: It describes the action the agent executed to assist the user.
- user reaction: It describes the feedback the user provided after the agent’s assistance.
- evaluation: It describes the evaluation of the assistance experience, according to the user feedback.

For user profiling, the attributes describing the date and time of the event were removed from the files because they generated useless association rules that related, for example, a day or a month with a type of assistance action.

The attributes involved in the new event and holiday datasets are those used to described an event, namely event type, topic, participants, host and place.

8.2.2 User profiles and datasets for IONWI

To make the experiments with the IONWI algorithm we used 26 datasets containing user-agent interaction experiences in the calendar management domain. Each database is composed of the attributes that describe the problem situation or situation of interest originating the interaction, the primary user task, the modality of the assistance, the relationship between the situation and the user task, the user feedback, and the evaluation of the interaction experience.

- modality: It indicates how the assistance was provided, interrupting the user or not
- user task: It describes the task the user was carrying out when the agent proactively assisted him.
- task relevance: It describes the relationship between the user task and the situation originating the interruption. The situation can be relevant or irrelevant to current task, and in this latter case, it can be related or unrelated to it.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>User</th>
<th>N° of Attributes</th>
<th>N° of Instances</th>
<th>Missing Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>holiday</td>
<td>1</td>
<td>11</td>
<td>52</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>1</td>
<td>15</td>
<td>39</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>1</td>
<td>18</td>
<td>32</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>3</td>
<td>13</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>new event</td>
<td>3</td>
<td>13</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>3</td>
<td>16</td>
<td>92</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>3</td>
<td>19</td>
<td>106</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>4</td>
<td>13</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>new event</td>
<td>4</td>
<td>13</td>
<td>99</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>4</td>
<td>16</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>4</td>
<td>19</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>5</td>
<td>13</td>
<td>120</td>
<td>yes</td>
</tr>
<tr>
<td>new event</td>
<td>5</td>
<td>13</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>5</td>
<td>18</td>
<td>96</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>5</td>
<td>18</td>
<td>125</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>7</td>
<td>13</td>
<td>99</td>
<td>yes</td>
</tr>
<tr>
<td>new event</td>
<td>7</td>
<td>13</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>7</td>
<td>16</td>
<td>102</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>7</td>
<td>19</td>
<td>94</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>16</td>
<td>15</td>
<td>88</td>
<td>yes</td>
</tr>
<tr>
<td>new event</td>
<td>16</td>
<td>12</td>
<td>52</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>16</td>
<td>12</td>
<td>46</td>
<td>yes</td>
</tr>
<tr>
<td>time</td>
<td>16</td>
<td>17</td>
<td>78</td>
<td>yes</td>
</tr>
<tr>
<td>overlapping</td>
<td>28</td>
<td>15</td>
<td>30</td>
<td>yes</td>
</tr>
<tr>
<td>new event</td>
<td>28</td>
<td>13</td>
<td>30</td>
<td>yes</td>
</tr>
<tr>
<td>holiday</td>
<td>28</td>
<td>12</td>
<td>30</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 8.2: Characteristics of the datasets for IONWI

- user reaction: It describes the feedback the user provided after the agent’s assistance. The feedback refers to the assistance modality.

Like the datasets used to test WATSON, these datasets were built according to a set of interaction preferences indicated by a subset of the users (seven users) that participated in the study reported in Chapter 2. These users expressed their interruption preferences for the four contexts described before in this section. The datasets were artificially built with this information. Table 8.2 describes the main characteristics of the datasets.
8.3 Evaluating WATSON ability to learn a user's assistance requirements

The first kind of test we performed was evaluating the ability of the WATSON algorithm to discover a user's assistance requirements. To achieve this goal, we run the WATSON algorithm with the different datasets and we obtained the assistance requirements for the different users. Then, we compared the results obtained with the preference descriptions these users gave us through the questionnaire. To make this experiment we run the algorithm with the different databases in a “batch” fashion, that is, we processed the databases only once and we studied the resulting user profile. We used the User Profiling Toolkit to carry out this test.

Figures 8.1 to 8.8 present the results we have obtained for the four types of the datasets. The graphs in Figures 8.1, 8.3, 8.5, and 8.7 plot the percentage of assistance requirements correctly identified by the algorithm (with respect to the total number of assistance requirements obtained); the number of incorrect assistance requirements; and the number of “hidden” assistance requirements, that is those assistance requirements that were not explicitly stated by the user but are correct. Each figure presents the percentage values obtained when averaging the results for the different users with a given dataset. The graphs in Figures 8.2, 8.4, 8.6 and 8.8 show the percentage of correct assistance requirements (with respect to the number of requirements specified by the user) and the percentage of missing assistance requirements, that is those assistance requirements the algorithm could not detect. Each figure shows the average percentage values obtained with the different users for a particular dataset.

To understand our results, we can make a parallel between the measures we have obtained and the precision and recall metrics used in Information Retrieval [Oard and Marchionini, 1996]. The precision metric is the ratio of the number of relevant documents to all the documents retrieved by the system. The recall metric is the ratio of the number of relevant documents retrieved to all the relevant documents available. Thus, as shown in Equation 8.1 we can define our precision metric as the ratio of the number of correct assistance requirements to the total number of assistance requirements generated by WATSON.

\[
WATSON_{\text{precision}} = \frac{\text{number of correct assistance requirements}}{\text{number of assistance requirements}} \tag{8.1}
\]

Similarly, as shown in Equation 8.2, we can define our recall metric as the ratio of the number of correct assistance requirements to the number of assistance
8.3. Evaluating WATSON ability to learn a user’s assistance requirements

![Graph showing precision and recall for WATSON](image)

**Figure 8.1:** WATSON precision (holiday dataset)

![Graph showing recall for WATSON](image)

**Figure 8.2:** WATSON recall (holiday dataset)

requirements specified by the user.

\[
WATSON_{\text{recall}} = \frac{\text{number of correct assistance requirements}}{\text{number of assistance requirements specified by user}} \quad (8.2)
\]

Figures 8.9 and 8.10 show the average values obtained by combining the results for the different datasets.

We can observe in the figures that the percentage of incorrect assistance requirements is small, and that the percentage of correct assistance requirements plus the percentage of hidden assistance requirements is considerably high. We call hidden assistance requirements to those requirements that although they are not explicitly stated by the user, they are correct. The percentage of correct assistance requirements plus the percentage of hidden assistance requirements can be considered as the precision of the algorithm. This value is approximately 86%. Thus, we can state that the learning capability of the WATSON algorithm is good.

On the other hand, there are about 35% of the assistance requirements specified by the user that were not discovered by our algorithm. The recall value for our
algorithm is then 65%. Several factors have an impact on this value, such as the number of rules generated, the number of times a certain situation occurs with respect to the total number of situations recorded, the value of minsup. As regards the number of rules, we can increase the recall value by increasing the number of rules generated by the Apriori algorithm, as studied in Chapter 7 (the experiments were carried out with 1000 rules). As regards situation frequency, in some datasets the number of times a given situation appears is small. Thus, it is likely that this situation does not appear as a hypothesis when running the algorithm. The value of minsup should be very small so that these situations could be captured.

8.4 Comparing WATSON against the confidence-based algorithm

In order to compare our approach with the standard interface agent approaches, we compared WATSON precision at assisting a user against the precision of a confidence-based decision making algorithm. This kind of test is common in the
8.4. Comparing *WATSON* against the confidence-based algorithm

**Figure 8.5: WATSON precision (new event dataset)**

**Figure 8.6: WATSON recall (new event dataset)**

**Figure 8.7: WATSON precision (time dataset)**
8.4. Comparing \textit{WATSON} against the confidence-based algorithm

![Pie chart showing Time](chart1.png)

Figure 8.8: \textit{WATSON} recall (time dataset)

![Pie chart showing Average](chart2.png)

Figure 8.9: Average \textit{WATSON} precision

![Pie chart showing Average](chart3.png)

Figure 8.10: Average \textit{WATSON} recall
User Modeling area, in which experiments are carried out to test the performance of a system with a user model and without it [Chin, 2001]. In the experiment we are describing, the user interaction profile plays the role of the user model in the User Modeling tests.

In order to compare the performances of the two algorithms we used one of the metrics defined in [Brown and Santos, 1998]. The precision metric measures an interface agent’s ability to accurately provide assistance to a user. The precision metric is defined as shown in Equation 8.3. This equation is an adaptation of the precision metric proposed by [Brown and Santos, 1998].

\[
\text{Decision Making}_{\text{precision}} = \frac{\text{number of correct assistance actions}}{\text{number of assistance actions}} \quad (8.3)
\]

The precision metric is used to evaluate the performance of an interface agent when it has to decide between a warning, a suggestion or an action on the user’s behalf. In this case, for each problem situation, we compare the number of correct assistance actions against the total number of assistance actions the agent has executed. An assistance action is correct if it is the one expected by the user in a given problem situation. When working with real users, the user’s feedback tells us whether an assistance action is correct or not. In our case, the user profile description containing a user’s preferences and requirements tells us whether the assistance action is correct or not.

Figures 8.11 to 8.14 show the precision of the two algorithms for different datasets and users. Each bar graph plots the percentage of problem situations in which the agent assisted the user correctly (percentage of correct assistance actions) with respect to the total number of times the agent assisted the user. The bar on the
Figure 8.12: WATSON versus Confidence-Based Algorithm (new event dataset)

Figure 8.13: WATSON versus Confidence-Based Algorithm (overlapping dataset)

Figure 8.14: WATSON versus Confidence-Based Algorithm (time dataset)
8.5 Evaluating IONWI ability to learn a user’s interruption preferences

Left shows the results obtained with WATSON and the bar on the right shows the results obtained with the confidence-based algorithm. The last pair of bars in each graph plots the results obtaining when averaging the precision for the different users with a given dataset.

We can observe that the percentage of correct assistance actions is higher for the WATSON algorithm than for the confidence-based algorithm. The average precision of the WATSON decision making algorithm is approximately 70%, while the precision of the confidence-based approach is 60%. We consider than a 10% increase in precision is an important achievement. Moreover, we consider that the enhancement of the agent’s capability of learning what the user wants is even more important. We can appreciate the difference between the two algorithms when, for example, the user requires just a warning about a frequent problem and the confidence-based algorithm proposes a suggestion or an action. The WATSON algorithm will only make a warning, although the confidence on the assistance contents is high. When the problem is not frequent and warnings are required, the two algorithms behave similarly.

8.5 Evaluating IONWI ability to learn a user’s interruption preferences

Like in the case of WATSON, we performed an experiment to evaluate the ability of the IONWI algorithm to discover a user’s interruption preferences. To achieve this goal, we run the IONWI algorithm with the different datasets and we obtained the interruption preferences for the different users. Then, we compared the results obtained with the preferences these users gave us through the questionnaire.

The metrics we used are similar to those defined for the WATSON algorithm. As shown in Equation 8.4 we can define our precision metric as the ratio of the number of correct interruption preferences to the total number of interruption preferences generated by IONWI.

\[ IONWI_{\text{precision}} = \frac{\text{number of correct interruption preferences}}{\text{number of interruption preferences}} \quad (8.4) \]

Similarly, as shown in Equation 8.5, we can define our recall metric as the ratio of the number of correct interruption preferences to the number of preferences specified by the user.
8.5. Evaluating IONWI ability to learn a user’s interruption preferences

![New Event](image1)

**Figure 8.15: IONWI precision (new event dataset)**

![New Event](image2)

**Figure 8.16: IONWI recall (new event dataset)**

\[
IONWI_{recall} = \frac{\text{number of correct interruption preferences}}{\text{number of interruption preferences specified by user}}
\] (8.5)

Figures 8.15 to 8.22 present the results we have obtained for the four datasets, namely holiday, new event, overlapping and time. For each user profile, the graphs in Figures 8.15, 8.17, 8.19 and 8.21 plot the percentage of interruption preferences correctly identified by the algorithm (with respect to the total number of preferences obtained); the number of incorrect interruption preferences; and the number of “hidden” preferences. Each figure shows the percentage values obtained when averaging the results we got with the different users for a given dataset. The graphs in Figures 8.16, 8.18, 8.20 and 8.22 show the percentage of correct interruption preferences (with respect to the number of preferences specified by the user) and the percentage of missing interruption preferences, that is those that the algorithm could not detect. Each graphic shows the average percentage values of the results obtained with the different users for a particular dataset. Figures 8.23 and 8.24 show the average values obtained by combining the results of the different datasets.
8.5. Evaluating IONWI ability to learn a user’s interruption preferences

Figure 8.17: IONWI precision (holiday dataset)

Figure 8.18: IONWI recall (holiday dataset)

Figure 8.19: IONWI precision (overlapping dataset)
8.5. Evaluating *IONWI* ability to learn a user’s interruption preferences

![Overlapping](image)

Figure 8.20: *IONWI* recall (overlapping dataset)

![Time](image)

Figure 8.21: *IONWI* precision (time dataset)

![Time](image)

Figure 8.22: *IONWI* recall (time dataset)
8.5. Evaluating *IONWI* ability to learn a user’s interruption preference

![Average](image1)

**Figure 8.23:** Average *IONWI* precision

![Average](image2)

**Figure 8.24:** Average *IONWI* recall
8.6. Conclusions

We can observe in the figures that the percentage of incorrect interruption preferences is small (9% in average), and that the percentage of correct preferences plus the percentage of hidden preferences is considerably high. The percentage of correct interruption preferences plus the percentage of hidden preferences can be considered as the precision of the algorithm. This value is approximately 91%. Thus, we can state that the learning capability of the IONWI algorithm is very good.

Regarding the algorithm recall, 25% of the interruption preferences specified by the user were not discovered by our algorithm. Thus, the recall value is 75%. The experiments were carried out by generating 1500 association rules. As discussed in Chapter 7, generating a greater number of rules can improve the recall value.

8.6 Conclusions

In this Chapter we presented the results we obtained when evaluating the WATSON and IONWI algorithms in the calendar management domain. The results show that the precision of the WATSON algorithm at learning users’ assistance requirement is high (86%). On the other hand, the recall value for this algorithm is about 65%. Although it is not small, the recall can be improved. We have discussed in this Chapter how we can improve this value by tuning different parameters of the algorithm.

We compared the precision of the decision making algorithm used in our approach against the precision of a confidence based algorithm. The precision of our approach is 10% higher than the precision of the confidence-based approach. We consider that this improvement in the precision at assisting users is important. Moreover, we are improving the interaction between users and interface agents by learning the type of assistance users’ expect in different contexts.

As regards interruptions, the results we presented show that the precision of the IONWI algorithm at learning users’ interruption preferences is high (90%). The recall value for this algorithm is 75%. Thus, we can conclude that the performance of the algorithm is good and that we have achieved our goal.
Chapter 9

Conclusions

This final chapter summarizes the work reported in this thesis. We present an overview of the thesis, we describe the main contributions of our work, the limitations we found, and the prospective future work and research.

9.1 Summary

As interface agents become more complex, adaptive, autonomous and intelligent, the importance of their showing appropriate behavior increases, and conversely, the sensitivity of users to inappropriate behavior will increase. As stated in [Reeves and Nass, 1996], users apply the schemas learned for interpreting and interacting with humans to other agents that behave, in some minimal ways, like humans. Thus, they expect their agents to act in a polite, efficient and unobtrusive way, adjusting their behaviors to what it is called human-computer etiquette [Miller, 2004].

To achieve this goal, interface agents have to personalize their interaction with users and assist them as they expect. In this work, we showed through an experimental study that there are different user-agent interaction issues that have to be personalized in order to improve agent behavior. By personalizing user-agent interaction we can avoid agent rejection and fulfill users’ expectations with respect to agents’ behavior.

Our work addresses two of the personalization issues we have experimentally discovered. Our approach enables an agent to assist the user with the assistance action (warning, suggestion, action on the user’s behalf) he expects and prefers in a given context. In turn, the agent can also learn the expected modality for the assistance action, that is, interrupting the user’s work or not.

Our approach proposes the definition of a user interaction profile that contains information about how the user wants to be assisted in different contexts. The user
interaction profile enhances the standard user profiles interface agents have built thus far. In addition, we developed two profiling algorithms that build the two components of the user interaction profile. The algorithms, WATSON and IONWI, obtain a user’s assistance requirements and interruption preferences by observing a user’s interaction with the agent and with the computer application the user is working with. The user interaction profile built with these algorithms is then used by the agent to decide how to interact and assist the user in a different contexts.

We evaluated our proposed approach in the calendar management domain. The results we obtained show that our approach improves interface agent performance, since agents consider not only users’ preferences with respect to a certain application but also user’s requirements and preferences with respect to their interaction with agents.

9.2 Main Contributions

There are two main contributions of our work in the Interface Agents and the Human-Computer Interaction areas. The work described in this thesis report contributes to the understanding of personalization of user-agent interaction, and to the enhancement of interface agent capabilities.

- **Understanding of personalization of user-agent interaction**: The study presented in Chapter 2 contributes to a better understanding of the relationship between users and interface agents. We studied different aspects of user-agent interaction that have to be personalized not to disappoint the user. Our study has implications both in the design and development of interface agents. The information we found can be used by agent developers to build interface agents capable of adapting to users’ expectations and preferences regarding user-agent interaction.

- **Enhancing interface agents with new personalization capabilities**: Our approach enables interface agents to personalize not only the assistance given to users with a given computer application, but also the way in which they interact with users. The WATSON algorithm enables an agent to learn a user’s assistance requirements. Thus, the agent can act as the user expects and prefers in different contexts. The IONWI algorithm enables an interface agent to learn a user’s interruption preferences. In consequence, the agent can interact with the user without interrupting him unexpectedly or hindering his work.
9.3. Limitations

There are also two secondary (but important) contributions. Our work contributes to the testing and evaluation of interface agents and it proposes the use of data mining technique for user profiling.

- **A tool for testing user profiling algorithms:** It is widely known that one of the problems with interface agents is testing them and evaluating their performance in different scenarios. The User Profiling Toolkit enables agent developers to evaluate the user profile building and the decision making capabilities of their agents. The agent developer has to separate the user profiling algorithm and embed it in our tool in order to test it with different configurations of parameters. Then, the agent developer can simulate the agent behavior when assisting the user by incorporating the decision making or action selection algorithm into the tool. The tool enables agent developers to define different mock problem situations or situations of interest in order to evaluate the decision making algorithm.

- **Use of data mining techniques for learning with few data:** In this work we have used association rules to build user interaction profiles. We can find a few works using data mining techniques to build user profiles, and in most cases, the user profiling task is actually a data mining task [Adomavicius and Tuzhilin, 2001, Mobasher et al., 2001]. Our work is innovative in the way association rules are processed to build a user interaction profile. In Data Mining, it is generally needed a user to check and validate the association rules generated and the knowledge obtained. In our work, the WATSON and IONWI algorithms automatically filter the association rules according to different criteria: subjective interestingness (templates), redundancy and contradictoriness. We do not need a human to check the results obtained. The work reported in [Adomavicius, 2002] proposes a set of operators to validate rules, but it still requires the presence of an expert.

9.3 Limitations

In this section we describe the limitations we have found with respect to our work. Our approach assumes that the agent has a standard user profiling algorithm that learns a user’s preferences with respect to the computer application the agent is assisting the user with. As long as this user profiling algorithm works well, that is its precision at learning a user’s preferences is high, our algorithms will work well. Our algorithm depends on the standard profiling algorithm because our decision
9.4 Future Work and Research

In this work we proposed a solution to two of the problems interface agents have to tackle to personalize their interaction with users. However, there are several issues discussed in Chapter 2 that still remain unsolved. For example, the user interaction profile can be enlarged with new components containing information about the type of assistant the user wants (submissive, authoritative), about the tolerance of the user to different agent errors in different contexts, about the frequency of remainders in different contexts, among others. Consequently, new interaction profiling algorithms will have to be developed and combined with IONWI and WATSON to build the new user interaction profile.

Our proposed approach uses association rules as learning technique. The results we obtained with this first proposal are good and promising. However, we have to compare the results obtained with association rules against the results obtained with other learning techniques. We are now working with techniques such as Bayesian Networks and Case-Based Reasoning. In order to use these techniques to learn users’
assistance requirements and interruption preferences, changes are required. Mainly, we must change the way in which inputs are processed and in the way hypotheses are generated both in \textit{WATSON} and \textit{IONWI}.

As shown in Chapter 8, we evaluated our proposed approach in a particular application domain, calendar management. As a future work, our proposal will be evaluated in other domains in which the agent has different assistance actions and assistance modalities to assist the user. We are currently working on a \textit{HomeStock Agent}. \textit{HomeStock Agent} is an interface agent that assists users with food stock management and menu preparation. It can suggest users the ingredients they can use to prepare a meal depending on the food stock available and on the guests. It can also tell users about some groceries that are not in stock and about others that have gone off, and then it can prepare a shopping list.
Appendix A

Interface Agents

Intelligent agents constitute a technology widely used in those areas where systems or applications are required to show an autonomous behavior and to have the ability to adapt to changes in their environment. Due to this wide variety of areas using intelligent agents, ranging from e-commerce to air traffic control systems, many definitions have been built for these agents. Although there is no agreement within the Artificial Intelligence (AI) community about the definition of this concept, perhaps the most general way in which the term agent is used is to denote a hardware or, more usually, a software-based computer system that has the following properties [Wooldridge and Jennings, 1995]:

- Autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;

- Social ability: agents interact with other agents, and possibly humans, via some kind of agent-communication language;

- Reactivity: agents perceive their environment, which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined, and respond in a timely fashion to changes that occur in it;

- Pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.

Interface agents are a particular kind of intelligent agents that are suitable to help users with computer tasks. We can use a metaphor to describe interface agents’ behavior, comparing them to a human secretary or personal assistant who is collaborating with the user in the same work environment [Maes, 1994]. According
to the personal assistant metaphor, the assistant is initially not very familiar with the habits and preferences of her employer and may not even be very helpful. The assistant needs some time to become familiar with the particular working methods of the employer and organization at hand. However, with every experience the assistant learns, either by watching how the employer performs tasks, by receiving instructions from the employer, by asking the employer for information or by learning from other more experienced assistants within the organization, she becomes more effective. Gradually, more tasks that were initially performed directly by the employer can be taken care by the assistant.

Consider, for example, the task of managing our daily activities, such as business meetings, appointments with the doctor or the dentist, courses, gym classes, parties and dinners with friends. In order to schedule an event we first have to find a free time slot in our agenda, checking that the date and time are appropriate for the event type. Then, we have to find an appropriate place, determine the duration of the event, send invitations to the participants of the event (if we are scheduling a meeting, for example), and then, if some conflicts arise, we might have to reorganize everything. It would be great if we could delegate or share the management of our schedules with a personal assistant. A personal assistant could perform these tasks if she knows our preferences, our commitments, our priorities and the way we usually arrange our activities and solve conflicts. In this way, we could share some scheduling tasks with our secretary and we could spend our time in more productive tasks.

An interface agent can act in the same way as a human secretary does. It can learn the user’s scheduling preferences and priorities, acquire knowledge about a user’s contacts and the relationships the user has with them, and learn the way in which the user manages his calendar. Thus, the agent could alleviate the user’s work by suggesting him meeting places, meeting dates and meeting times, by scheduling some events automatically, by warning about conflicts between events and by suggesting how to reorganize them. The agent will acquire more information as it interacts with the user and it will then become more efficient and competent.

A learning interface agent acquires its competence from four different sources, as shown in Figure A.1. First, the agent learns by continuously “looking over the shoulder” of the user as he is working with a computer application. The interface agent can monitor the activities of the user, keep track of all his actions over a period of time, find regularities and recurrent patterns and offer assistance according to them. A second source of learning is direct and indirect user feedback. Indirect feedback happens when the user neglects a suggestion of the agent and takes a
different action instead. The user can also give explicit negative or positive feedback for the assistance provided by the agent. Third, the agent can learn from examples or instructions given explicitly by the user. The user can train the agent by giving it hypothetical examples of situations and telling it what to do in those cases. Finally, a fourth method used by the interface agent to acquire competence is to ask for advice to agents that assist other users with the same task.

There are three main benefits of using interface agent technology. First, interface agents save a user’s time, work and effort, increasing in this way his productivity. Second, the agent has the capability of adapting its behavior as time goes by according to the habits and preferences of the user, becoming gradually more competent. Finally, the agents can share their knowledge and know-how, while respecting each user’s privacy.

There are several kinds of interface agents, covering a wide spectrum of interactions depending on the relative expertise and initiative of the agents. For example, tutoring agents have great expertise and initiative, and the primary shared goal is to increase the students’ expertise. At the other end of the spectrum, we can find intelligent assistants that have somewhat less expertise and initiative, and the primary goal is to successfully accomplish some task delegated by the user or specified by the agent developer. In the middle of the spectrum are peer collaborators, in which the expertise and initiative of the participants are relatively evenly matched.

Our work is placed within a collaboration approach, in which a user and an agent coordinate their actions towards achieving shared goals [Rich and Sidner, 1998]. In this approach, the user is engaged in a cooperative process in which humans and agents both initiate communication (mixed-initiative), monitor events and perform tasks. For example, the user may request the agent’s help and the agent can autonomously give the user a piece of advice. In this paradigm an interface agent plays the same role a human plays when two humans collaborate on a task involving
a shared artifact, such as two computer users working on a spreadsheet together. In addition to communicating with each other, both the user and the interface agent can interact with the shared artifact (typically a software application) and observe each other’s interactions with the shared artifact.

Interface agents provide personalization in many different ways, depending on their purposes. For example, a browsing assistant guides the user to those Web pages that are relevant to him; a meeting scheduling assistant can arrange meetings on the user’s behalf according to his preferences and priorities; a news filtering agent only presents to the user those news that are relevant to him. These agents use the information contained in the user profile to achieve their personalization goals. Table A.1 shows a set of interface agents assisting users in various tasks, with a brief description of what a user profile consists of for each of them.

Recently, other forms of personalization have been taken into account in agent development. In [André et al., 1999] the personality of an interface agent is considered. The personality of an agent is defined by several features known as the OCEAN model [Wiggins, 1996]: openness, conscientiousness, extroversion, agreeableness and neuroticism. An agent behaves according to a particular combination of these features. For example, as regards extroversion, an agent can be extrovert, neutral or introvert. Regarding agreeableness, an agent can be agreeable, neutral or disagreeable. Each user can select the personality he wants for his interface agent.

The audiovisual appearance of interface agents has also been taken into account in [André and Rist, 2002], where the goal is representing interface agents with lifelike characters since this seems to be a more natural way of interacting with users. Each user can choose a different character as a representation for his agent, such as a parrot, a genie or a wizard.

The work described in [Scerri et al., 2002] presents the concept of adjustable autonomy, which refers to an agent’s dynamically varying its own autonomy, transferring decision making control to other entities in key situations. The key question is whether and when agents should make autonomous decisions and when they should transfer decision-making control to other entities (typically human users). The authors introduce the notion of a transfer-of-control strategy, which is a planned sequence of transfer-of-control actions, including both those that actually transfer control and those that simply buy more time to get input. The goal is finding an strategy that maximizes the expected utility of the decision, considering several factors such as: the relative decision-making quality of the entities, the probability that a given entity will respond in a timely manner, the costs incurred by waiting for a response, and any intermediate actions that the agent can take to limit those
<table>
<thead>
<tr>
<th>Agent</th>
<th>Agent Task</th>
<th>User Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calendar Agent</td>
<td>Meeting scheduling assistant</td>
<td>User scheduling preferences and commitments</td>
</tr>
<tr>
<td>[Kozierok and Maes, 1993]</td>
<td></td>
<td>Set of priority weightings for each meeting feature. Relative importance of user contacts.</td>
</tr>
<tr>
<td>CAP</td>
<td>Meeting scheduling assistant</td>
<td>Decision trees used to suggest meeting features: duration and place</td>
</tr>
<tr>
<td>[Mitchell et al., 1994]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CiteSeer</td>
<td>Searches for relevant publications on the Web</td>
<td>TF-IDF [Salton and McGill, 1983] vectors representing keywords supplied by the user and publications previously read</td>
</tr>
<tr>
<td>[Bollacker et al., 1998]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Letizia</td>
<td>Browsing assistant</td>
<td>Keywords and preference ordering of a set of links. User habits represented by reading time</td>
</tr>
<tr>
<td>[Elberman, 1995]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Magi</td>
<td>Email filtering</td>
<td>Rules generated from user actions are used to sort and filter emails. User feedback</td>
</tr>
<tr>
<td>[Payne, 1994]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Margin Notes</td>
<td>Memory aid</td>
<td>TF-IDF vectors and set of interesting documents provided by the user</td>
</tr>
<tr>
<td>[Rhodes, 2000]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MORSE</td>
<td>Recommendation of movies</td>
<td>User-supplied ratings for already seen movies</td>
</tr>
<tr>
<td>[Fisk, 1997]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NewsAgent</td>
<td>Generates personalized digital newspapers</td>
<td>Hierarchy of interesting news topics. Topics defined by sets of weighted words. User habits represented by average reading time</td>
</tr>
<tr>
<td>[Cordero et al., 1999]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NewsDude</td>
<td>Reads relevant news aloud</td>
<td>Interesting news given by the user. Listening time as feedback. TF-IDF vectors (short term profile) and a Bayesian classifier (long term profile)</td>
</tr>
<tr>
<td>[Billsus and Pazzani, 1999]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NewT</td>
<td>News filtering</td>
<td>Examples of + and - news. Newsgroup related fields (newsgroups, keywords, author, source, etc.) described by vectors of weighted terms</td>
</tr>
<tr>
<td>[Sheh and Maes, 1993]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Personal WebWatcher</td>
<td>Browsing assistant</td>
<td>Documents read by the user represented as TF-IDF vectors. Naive Bayesian Classifier</td>
</tr>
<tr>
<td>[Miadenic, 1999]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSUN</td>
<td>News filtering</td>
<td>S-entities that describe relationships (edges) between words (vertices) within documents (graphs)</td>
</tr>
<tr>
<td>[Elligott and Sorensen, 1995]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QueryGuesser</td>
<td>Assists users of DBMS by personalizing queries</td>
<td>Bayesian network representing relationships among query features</td>
</tr>
<tr>
<td>[Schiaffino and Amandi, 2000]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Syskill&amp;Webert</td>
<td>Browsing assistant</td>
<td>Set of keywords and associated probability ratings for interesting and uninteresting pages. User + and - feedback. Examples of interests supplied by the user.</td>
</tr>
<tr>
<td>[Pazzani and Billsus, 1997]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WebMate</td>
<td>Browsing and searching assistant</td>
<td>Multiple TF-IDF vectors representing news read by the user. Relevance feedback</td>
</tr>
<tr>
<td>[Chen and Sycara, 1998]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.1: Examples of interface agents
costs. Decision-theoretic techniques are used to determine the expected utility of a particular strategy within a specific domain. The work reported is part of the Electric Elves project [Chalupsky et al., 2001]. In [Falcone and Castelfranchi, 2001] the authors analyze trust, autonomy and delegation. They mainly study trust as the basis of delegation. They also propose a framework for the theory of adjustable social autonomy in complex scenarios.
Appendix B

Questionnaire

1. Have you ever used/developed an interface agent?
   
   (a) Yes (Development - Use)
   
   (b) No

2. If you have to provide explicit feedback for the suggestions provided by your agent, which will be your reaction?

   (a) I would feel annoyed
   
   (b) I would not object to providing feedback if it helps the agent learn
   
   (c) I would not object at the beginning, but I would do as time passed by
   
   (d) I would not object if the mechanism is simple (yes or no, relevant or irrelevant), but I would do if it demands me time and effort

3. Which kind of assistance would you like to receive from your agent?

   (a) All of them, but depending on the context
   
   (b) All of them except from autonomous actions

   (c) Only warnings; the rest of them only upon request

4. Which would your reaction be if the type of assistance provided by the agent is different from the one you expect? (e.g.: suggestion instead of warning, action instead of suggestion)

   (a) I would feel completely annoyed
   
   (b) I would feel annoyed and I would teach the agent via feedback
(c) It depends on the magnitude of the error
(d) I would not feel annoyed and I would fix the situation

5. Would you tolerate this situation at the beginning of your interaction with the agent?

(a) Yes
(b) No

6. Would your reaction be the same for every type of assistance action? Explain

(a) Yes
(b) No

7. Would you allow your agent to perform actions on your behalf?

(a) Never
(b) Always
(c) Only if I am sure about what it is going to do
(d) Only if I ask it to do so
(e) Only in certain situations (not compromising me)

8. Regarding how the agent provides assistance, would your tolerate interruptions?

(a) Yes
(b) No
(c) Only if the underlying situation is urgent and relevant

9. How would you interact with your agent?

(a) Actively collaborating and interacting with it
(b) I would prefer the agent to work in background, acting only upon my request
(c) I would prefer the agent to intervene in certain situations, and I would interact with him actively in these cases
10. Could you identify contexts or situations in which you clearly prefer each type of assistance action? Give examples for the domains studied

(a) Yes

(b) No
Appendix C

The Attribute-Relation File Format

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software. The contents of this Appendix are a summary of the tutorial given at Weka site\(^1\).

C.1 Overview

ARFF files have two distinct sections. The first section is the Header information, which is followed by the Data information.

The Header of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. An example header on the standard IRIS dataset looks like this:

\[
\%
\。
Title: Iris Plants Database  
\%
\end{verbatim}
% 2. Sources:  
% (a) Creator: R.A. Fisher  
% (b) Donor: Michael Marshall (MARSHALL-PLU@io.arc.nasa.gov)  
% (c) Date: July, 1988  
@relation iris  
@attribute sepallength NUMERIC  
@attribute sepalwidth NUMERIC  
@attribute petallength NUMERIC  
@attribute petalwidth NUMERIC

\(^1\)http://www.cs.waikato.ac.nz/~ml/weka/arff.html
C.2. The ARFF Header Section

@attribute class {Iris-setosa, Iris-versicolor, Iris-virginica}

The Data of the ARFF file looks like the following:
@data
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
4.9,3.4,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
4.6,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa

Lines that begin with a % are comments. The @relation, @attribute and @data declarations are case insensitive.

C.2 The ARFF Header Section

The ARFF Header section of the file contains the relation declaration and attribute declarations.

C.2.1 The @relation Declaration

The relation name is defined as the first line in the ARFF file. The format is:

@relation <relation-name>

where <relation-name> is a string. The string must be quoted if the name includes spaces.

C.2.2 The @attribute Declarations

Attribute declarations take the form of an ordered sequence of @attribute statements. Each attribute in the data set has its own @attribute statement which uniquely defines the name of that attribute and its data type. The order the attributes are declared indicates the column position in the data section of the file. For example, if an attribute is the third one declared then Weka expects that all the attributes values will be found in the third comma delimited column.
The format for the @attribute statement is:

@attribute <attribute-name> <datatype>

where the <attribute-name> must start with an alphabetic character. If spaces are to be included in the name then the entire name must be quoted.

The <datatype> can be any of the four types currently (version 3.2.1) supported by Weka:

- numeric
- <nominal-specification>
- string
- date [<date-format>]

where <nominal-specification> and <date-format> are defined below. The keywords numeric, string and date are case insensitive.

NUMERIC ATTRIBUTES

Numeric attributes can be real or integer numbers.

NOMINAL ATTRIBUTES

Nominal values are defined by providing an <nominal-specification> listing the possible values: \{<nominal-name1>, <nominal-name2>, <nominal-name3>, ...\}

For example, the class value of the Iris dataset can be defined as follows:

@attribute class \{Iris-setosa,Iris-versicolor,Iris-virginica\}

Values that contain spaces must be quoted.

STRING ATTRIBUTES

String attributes allow us to create attributes containing arbitrary textual values. This is very useful in text-mining applications, as we can create datasets with string attributes, then write Weka Filters to manipulate strings (like StringToWord-VectorFilter). String attributes are declared as follows:

@attribute LCC string

DATE ATTRIBUTES

Date attribute declarations take the form:

@attribute <name> date [<date-format>]

where <name> is the name for the attribute and <date-format> is an optional string specifying how date values should be parsed and printed (this is the same
format used by SimpleDateFormat). The default format string accepts the ISO-8601 combined date and time format: "yyyy-MM-dd'T'HH:mm:ss".

Dates must be specified in the data section as the corresponding string representations of the date/time.

C.3 ARFF Data Section

The ARFF Data section of the file contains the data declaration line and the actual instance lines.

C.3.1 The @data Declaration

The @data declaration is a single line denoting the start of the data segment in the file. The format is:

@data

C.3.2 The instance data

Each instance is represented on a single line, with carriage returns denoting the end of the instance. Attribute values for each instance are delimited by commas. They must appear in the order that they were declared in the header section (i.e. the data corresponding to the nth @attribute declaration is always the nth field of the attribute).

Missing values are represented by a single question mark, as in:

@data 4.4,?,1.5,?,Iris-setosa

Values of string and nominal attributes are case sensitive, and any that contain space must be quoted, as follows:

@relation LCCvsLCSH
@attribute LCC string
@attribute LCSH string
@data
AG5, 'Encyclopedias and dictionaries;Twentieth century.'
AS262, 'Science – Soviet Union – History.'
AE5, 'Encyclopedias and dictionaries.'
AS281, 'Astronomy, Assyro-Babylonian;Moon – Phases.'
AS281, 'Astronomy, Assyro-Babylonian;Moon – Tables.'
Dates must be specified in the data section using the string representation specified in the attribute declaration. For example:

@relation Timestamps
@attribute timestamp DATE "yyyy-MM-dd HH:mm:ss"
@data
"2001-04-03 12:12:12"
"2001-05-03 12:59:55"

C.4 Full example on the ZOO dataset

% Changes to WEKA Format: SRG - November 1994

% 1. Boolean attributes changed from 1 and 0 to Enumerated attribute with values {true and false}

% 2. Class Number (attribute 18) changed to an Enumerated type with values {1,2,3,4,5,6,7}

% 1. Title: Zoo database
% 2. Source Information
% – Creator: Richard Forsyth
% – Donor: Richard S. Forsyth - Grosvenor Avenue - Mapperley Park - Nottingham NG3 5DX - 0602-621676
%– Date: 5/15/1990
% 3. Relevant Information:
% – A simple database containing 17 Boolean-valued attributes. The "type" attribute appears to be the class attribute. Here is a breakdown of which animals are in which type: (I find it unusual that there are 2 instances of "frog" and one of "girl"!)
% Class# Set of animals:
% 1 (41) aardvark, antelope, bear, boar, buffalo, calf, cavy, cheetah, deer, dolphin, elephant, fruitbat, giraffe, girl, goat, gorilla, hamster, hare, leopard, lion, lynx, mink, mole, mongoose, opossum, oryx, platypus, polecats, pony, porpoise, puma, pussycat, raccoon, reindeer, seal, sealion, squirrel, vampire, vole, wallaby, wolf
% 2 (20) chicken, crow, dove, duck, flamingo, gull, haw, kiwi, lark, ostrich, parakeet, penguin, pheasant, rhea, skimmer, skua, sparrow, swan, vulture, wren
% 3 (5) pit viper, seasnake, slowworm, tortoise, tuatara
% 4 (13) bass, carp, catfish, chub, dogfish, haddock, herring, pike, piranha, seahorse, sole, stingray, tuna
% 5 (4) frog, frog, newt, toad
% 6 (8) flea, gnat, honeybee, housefly, ladybird, moth, termite, wasp
% 7 (10) clam, crab, crayfish, lobster, octopus, scorpion, seawasp, slug, starfish, worm
%
% 5. Number of Instances: 101
% 6. Number of attributes: 18 (animal name, 15 Boolean attributes, 2 numerics)
% 7. attribute Information: (name of attribute and type of value domain)
%
% 1. animal name: Unique for each instance
% 2. hair Boolean
% 3. feathers Boolean
% 4. eggs Boolean
% 5. milk Boolean
% 6. airborne Boolean
% 7. aquatic Boolean
% 8. predator Boolean
% 9. toothed Boolean
% 10. backbone Boolean
% 11. breathes Boolean
% 12. venomous Boolean
% 13. fins Boolean
% 14. legs Numeric (set of values: {0,2,4,5,6,8})
% 15. tail Boolean
% 16. domestic Boolean
% 17. catsize Boolean
% 18. type Numeric (integer values in range [1,7])
%
% 8. Missing attribute Values: None
% 9. Class Distribution: Given above
@relation zoo
@attribute animal {aardvark, antelope, bass, bear, boar, buffalo, calf, carp, catfish, cavy, cheetah, chicken, chub, clam, crab, crayfish, crow, deer, dogfish, dolphin, dove, duck, elephant, flamingo, flea, frog, fruitbat, giraffe, girl, gnat, goat, gorilla, gull, haddock, hamster, hare, hawk, herring, honeybee, housefly, kiwi, ladybird, lark, leopard, lion, lobster, lynx, mink, mole, mongoose, moth, newt, octopus opossum, oryx, ostrich, parakeet, penguin, pheasant, pike, piranha, pitviper, platypus,
polecat, pony, porpoise, puma, pussycat, raccoon, reindeer, rhea, scorpion, seahorse, seal, sealion, seasnake, seawasp, skimmer, skua, slowworm, slug, sole, sparrow, squirrel, starfish, stingray, swan, termite, toad, tortoise, tuatara, tuna, vampire, vole, vulture, wallaby, wasp, wolf, worm, wren

@attribute hair {false, true}
@attribute feathers {false, true}
@attribute eggs {false, true}
@attribute milk {false, true}
@attribute airborne {false, true}
@attribute aquatic {false, true}
@attribute predator {false, true}
@attribute toothed {false, true}
@attribute backbone {false, true}
@attribute breathes {false, true}
@attribute venomous {false, true}
@attribute fins {false, true}
@attribute legs INTEGER [0, 9]
@attribute tail {false, true}
@attribute domestic {false, true}
@attribute catsize {false, true}
@attribute type { 1, 2, 3, 4, 5, 6, 7 }

@data
aardvark,true,false,false,true,false,true,false,false,false,false,4,false,false,true,1 antelope,true,false,false,true,false,false,false,false,false,false,4(true,false,false,4(true,bass,false,false,true,false,false,false,false,4(true,false,true,1 bear,true,false,false,true,false,false,false,false,false,false,4(true,false,1 boar,true,false,false,true,false,false,false,false,false,false,4(true,false,1 buffalo,true,false,false,true,false,false,false,false,false,false,4(true,false,1 calf,true,false,false,true,false,false,false,false,false,false,4(true,true,1 carp,false,true,false,false,false,true,false,true,0(true,false,1 catfish,false,false,true,false,false,true,false,false,false,false,4(true,false,1 cavy,true,false,false,false,false,false,false,false,false,false,4(true,false,1 cheetah,true,false,false,true,true,false,false,false,false,false,4(true,false,1 chicken,false,true,false,false,false,false,false,false,false,false,2(true,false,2 chub,false,false,false,true,false,false,false,false,false,false,4(false,false,4 clam,false,false,true,false,false,false,false,false,false,false,7

..........
Appendix D

Association Rules

A formal statement of the association rule mining problem is [Agrawal et al., 1993]:
let $I = I_1, I_2, ..., I_m$ be a set of $m$ distinct attributes, also called items or literals. Let
$D$ be a database, where each record (tuple) $T$ has a unique identifier and contains
a set of items such that $T \subseteq I$. An association rule is an implication of the form
$X \rightarrow Y$, where $X, Y \subseteq I$, are sets of items called itemsets and $X \cap Y = \emptyset$. $X$ is
called the antecedent of the rule, and $Y$ is the consequent.

Two important measures for association rules, support and confidence, can be
defined as follows. The support of an association rule is the ratio (in percent) of
the records that contain $X \cup Y$ to the total number of records in the database.
Therefore, if we say that the support of a rule is 5% then it means that 5% of the
total records contain $X \cup Y$. Support is the statistical significance of an association
rule. Grocery store managers probably would be concerned about how peanut butter
and bread are related if less than 5% of store transactions have this combination of
purchases. While a high support is often desirable for association rules, this is not
always the case. For example, if we were using association rules to predict the failure
of telecommunications switching nodes based on that set of events occur prior to a
failure, even if these events do not occur very frequently association rules showing
this relationship would still be important.

For a given number of records, confidence is the ratio (in percent) of the number
of records that contain $X \cup Y$ to the number of records that contain $X$. Thus, if we
say that a rule has a confidence of 85%, it means that 85% of the records containing
$X$ also contain $Y$. The confidence of a rule indicates the degree of correlation in
the dataset between $X$ and $Y$. Confidence is a measure of the strength of a rule.
Often a large confidence is required for association rules. If a set of events occur
a small percentage of the time before a switch failure or if a product is purchased
only very rarely with peanut butter, these relationships may be not of much use for
management.

D.1 The Apriori Algorithm

Mining of association rules from a database consists of finding all rules that meet the user-specified threshold for support and confidence. The problem of mining association rules can be decomposed into two subproblems as stated in Algorithm 13.

**Algorithm 13 Basic Algorithm**

**Input:** \( I, D, \text{minsup}, \text{minconf} \)

**Output:** Association rules satisfying minsup and minconf

1. Find all sets of items which occur with a frequency that is greater than or equals to the user-specified threshold support minsup
2. Generate the desired rules using the large itemsets, which have user-specified threshold confidence minconf

The first step in Algorithm 13 finds large or frequent itemsets. Itemsets other than those are referred as small itemsets. Here an itemset is a subset of the total set of items of interest from the database. An interesting (and useful) observation about large itemsets is that if an itemset \( X \) is small, any superset of \( X \) is also small.

In the remainder of this chapter we use \( L \) to designate the set of large itemsets. The second step of the algorithm finds association rules using large itemsets obtained in the first step. The following example illustrates this basic process for finding association rules from large itemsets. Consider a small database with four items \( I=\{\text{Bread, Butter, Eggs, Milk}\} \) and four transactions as shown in Table D.1. Table D.2 shows all itemsets for I. Suppose that the minimum support and minimum confidence of an association rule are 40% and 60%, respectively. There are several potential association rules. For discussion purposes we only look at those in Table D.3. At first, we have to find out whether all sets of items in those rules are large. Secondly, we have to verify whether a rule has a confidence of at least 60%. If the above conditions are satisfied for a rule, we can say that there is enough evidence to conclude that the rule holds with a confidence of 60%. Itemsets associated with the aforementioned rules are: \{Bread, Butter\}, and \{Butter, Eggs\}. The support of each individual itemset is at least 40% (see Table D.2). Therefore, all of these itemsets are large. The confidence of each rule is presented in Table D.3. It is evident that the first rule (Bread \( \rightarrow \) Butter) holds. However, the second rule (Butter \( \rightarrow \) Eggs) does not hold because its confidence is less than 60%.

The identification of the large itemsets is computationally expensive |Agrawal
D.1. The Apriori Algorithm

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Bread, Butter, Eggs</td>
</tr>
<tr>
<td>T2</td>
<td>Butter, Eggs, Milk</td>
</tr>
<tr>
<td>T3</td>
<td>Butter</td>
</tr>
<tr>
<td>T4</td>
<td>Bread, Butter</td>
</tr>
</tbody>
</table>

Table D.1: Transaction Database

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
<th>Large/Small</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bread</td>
<td>50%</td>
<td>large</td>
</tr>
<tr>
<td>Butter</td>
<td>100%</td>
<td>large</td>
</tr>
<tr>
<td>Eggs</td>
<td>50%</td>
<td>large</td>
</tr>
<tr>
<td>Milk</td>
<td>25%</td>
<td>small</td>
</tr>
<tr>
<td>Bread, Butter</td>
<td>50%</td>
<td>large</td>
</tr>
<tr>
<td>Bread, Eggs</td>
<td>25%</td>
<td>small</td>
</tr>
<tr>
<td>Bread, Milk</td>
<td>0%</td>
<td>small</td>
</tr>
<tr>
<td>Butter, Eggs</td>
<td>50%</td>
<td>large</td>
</tr>
<tr>
<td>Butter, Milk</td>
<td>25%</td>
<td>small</td>
</tr>
<tr>
<td>Eggs, Milk</td>
<td>25%</td>
<td>small</td>
</tr>
<tr>
<td>Bread, Butter, Eggs</td>
<td>25%</td>
<td>small</td>
</tr>
<tr>
<td>Bread, Butter, Milk</td>
<td>0%</td>
<td>small</td>
</tr>
<tr>
<td>Bread, Eggs, Milk</td>
<td>0%</td>
<td>small</td>
</tr>
<tr>
<td>Butter, Eggs, Milk</td>
<td>25%</td>
<td>small</td>
</tr>
<tr>
<td>Bread, Butter, Eggs, Milk</td>
<td>0%</td>
<td>small</td>
</tr>
</tbody>
</table>

Table D.2: Support for Itemsets in Table C.1 and Large Itemsets with support of 40%

<table>
<thead>
<tr>
<th>Rule</th>
<th>Confidence</th>
<th>Rule Hold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bread—Butter</td>
<td>100%</td>
<td>yes</td>
</tr>
<tr>
<td>Butter—Bread</td>
<td>50%</td>
<td>no</td>
</tr>
<tr>
<td>Butter—Eggs</td>
<td>50%</td>
<td>no</td>
</tr>
<tr>
<td>Eggs—Butter</td>
<td>100%</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table D.3: Confidence of some association rules where minconf is 60%
and Srikant, 1994]. However, once all sets of large itemsets \((l \in L)\) are obtained, there is a straightforward algorithm for finding association rules given in [Agrawal and Srikant, 1994] which is restated in Algorithm 14.

**Algorithm 14** Find association rules given large itemsets  

**Input:** \(T, D, \text{minsup}, \text{minconf}, L\)  

**Output:** Association rules satisfying minsup and minconf  

1. Find all nonempty subsets, \(x\), of each large itemset \(l\) in \(L\)  
2. **for** every subset \(x\) **do**  
3. Obtain a rule of the form \(x \rightarrow (l - x)\) if the ratio of the frequency of occurrence of \(l\) to that of \(x\) is greater than or equal to the \text{minconf}  
4. **end for**

For example, suppose we want to see whether the first rule \{Bread\}→\{Butter\} holds for our example. Here \(l = \{\text{Bread, Butter}\}\), and \(x=\{\text{Bread}\}\). Therefore, \((l - x) = \{\text{Butter}\}\). Now, the ratio of \text{support}(\text{Bread, Butter})\) to \text{support}(\text{Bread})\) is 100\% which is greater than the minimum confidence. Therefore, the rule holds. For a better understanding, let us consider the third rule, Butter→Eggs, where \(x = \{\text{Butter}\}\), and \((l - x) = \{\text{Eggs}\}\). The ratio of \text{support}(\text{Butter, Eggs})\) to \text{support}(\text{Butter})\) is 50\% which is less than 60\%. Therefore, we can say that there is not enough evidence to conclude \{\text{Butter}\}→\{\text{Eggs}\} with 60\% confidence. Since finding large itemsets in a huge database is very expensive and dominates the overall cost of mining association rules, most research has been focused on developing efficient algorithms to solve step 1 in Algorithm 13 [Agrawal and Srikant, 1994, Cheung et al., 1996, Klementinen et al., 1994].

The Apriori algorithm developed by [Agrawal and Srikant, 1994] is a great achievement in the history of mining association rules. It is by far the most well-known association rule algorithm. This technique uses the property that any subset of a large itemset must be a large itemset. It is assumed that items within an itemset are kept in lexicographic order.

The Apriori algorithm generates the candidate itemsets by joining the large itemsets of the previous pass and deleting those subsets which are small in the previous pass without considering the transactions in the database. By only considering large itemsets of the previous pass, the number of candidate large itemsets is significantly reduced.

In the first pass, the itemsets with only one item are counted. The discovered large itemsets of the first pass are used to generate the candidate sets of the second pass using the \text{apriori_gen}() function. Once the candidate itemsets are found, their supports are counted to discover the large itemsets of size two by scanning the database. In the third pass, the large itemsets of the second pass are considered
D.1. The Apriori Algorithm

<table>
<thead>
<tr>
<th>Large itemsets in the third pass (L₃)</th>
<th>Join (L₃, L₃)</th>
<th>Candidate sets of the fourth pass (C₄ after pruning)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Apple, Bagel, Chicken},</td>
<td>{Apple, Bagel, Chicken, Diet-Coke},</td>
<td>{Apple, Bagel, Chicken, Diet-Coke}</td>
</tr>
<tr>
<td>{Apple, Bagel, Diet-Coke},</td>
<td>{Apple, Chicken, Diet-Coke}</td>
<td></td>
</tr>
<tr>
<td>{Apple, Chicken, Diet-Coke}</td>
<td>{Apple, Chicken, Eggs}</td>
<td></td>
</tr>
<tr>
<td>{Apple, Chicken, Eggs}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Bagel, Chicken, Diet-Coke}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table D.4: Finding candidates sets using \textit{apriori-gen}

as the candidate sets to discover large itemsets of this pass. This iterative process terminates when no new large itemsets are found. Each pass \(i\) of the algorithm scans the database once and determines large itemsets of size \(i\). \(L_i\) denotes large itemsets of size \(i\), while \(C_i\) is candidates of size \(i\). The \textit{apriori\_gen()} function as described in [Agrawal and Srikant, 1994] has two steps. During the first step, \(L_{k-1}\) is joined with itself to obtain \(C_k\). In the second step, \textit{apriori\_gen()} deletes all itemsets from the join result, which have some \((k-1)\)-subset that is not in \(L_{k-1}\). Then, it returns the remaining large \(k\)-itemsets.

\begin{algorithm}
\caption{\textit{apriori-gen} function}
\begin{algorithmic}[1]
  \Require set of all large \((k-1)\)itemsets \(L_{k-1}\)
  \Ensure a superset of the set of all large \(k\)-itemsets
  \State insert into \(C_k\) \{joint step\}
  \State select \(p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}\)
  \State from \(L_{k-1} p, L_{k-1} q\)
  \State where \(p.item_1 = q.item_1, \ldots, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}\)
  \{prune step\}
  \For{all itemsets \(c\) in \(C_k\)}
    \For{all \((k-1)\)-subsets \(s\) of \(c\)}
      \If{\(s\) not in \(L_{k-1}\)}
        \Delete{\(c\) from \(C_k\)}
      \EndIf
    \EndFor
  \EndFor
\end{algorithmic}
\end{algorithm}

Consider the example given in Table D.4 to illustrate the \textit{apriori\_gen()}. Large itemsets after the third pass are shown in the first column. Suppose a transaction contains \{Apple, Bagel, Chicken, Eggs, Diet-Coke\}. After joining \(L_3\) with itself, \(C_4\) will be \{\{Apple, Bagel, Chicken, Diet-Coke\}, \{Apple, Chicken, Diet-Coke, Eggs\}\}. The prune step deletes the itemset \{Apple, Chicken, Diet-Coke, Eggs\} because its subset with 3 items \{Apple, Diet-Coke, Eggs\} is not in \(L_3\).

The \textit{subset()} function returns subsets of candidate sets that appear in a transac-
tion. Counting support of candidates is a time-consuming step in the algorithm [Cen-
giz, 1997]. To reduce the number of candidates that need to be checked for a given
transaction, candidate itemsets $C_k$ are stored in a hash tree. A node of the hash
tree either contains a leaf node or a hash table (an internal node). The leaf nodes
contain the candidate itemsets in sorted order. The internal nodes of the tree have
hash tables that link to child nodes. Itemsets are inserted into the hash tree using
a hash function. When an itemset is inserted, it is required to start from the root
and go down the tree until a leaf is reached. Furthermore, $L_k$ are stored in a hash
table to make the pruning step faster [Srikant, 1996] Algorithm 17 shows the Apriori
technique. As mentioned earlier, the algorithm proceeds iteratively.

Algorithm 16 Function count

\textbf{Input:} $C$: set of items, $D$: database
\textbf{Output:} count
1. for each transaction $T$ in $D$ do
2. for all subsets $x$ in $T$ do
3. if $x$ pertenece $C$ then
4. $x.\text{count} \leftarrow x.\text{count} + 1$
5. end if
6. end for
7. end for

Algorithm 17 Apriori algorithm

\textbf{Input:} $I$, $\text{minsup}$, $\text{minconf}$
\textbf{Output:} $L$
1. $C_1 \leftarrow I$
2. Generate $L_1$ by traversing database and counting each occurrence of an attribute
   in a transaction
3. for $k = 2$ to $L_{k-1} \neq \emptyset$ do
4. $C_k \leftarrow \text{apriori - gen}(L_{k-1})$
   \{Candidate itemset generation\}
   \{New k-candidate itemsets are generated from (k-1)-large itemsets\}
5. count($C_k$, $D$)
6. $L_k \leftarrow$ all $c$ in $C_k$ such that $c.\text{count} \geq \text{minsup}$
7. $k++$
8. end for
9. $L \leftarrow \cup L_k$
10. return $L$

Figure D.1 illustrates how the Apriori algorithm works on our example. Initially,
each item of the itemset is considered as a 1-item candidate itemset. Therefore, $C_1$
has four 1-item candidate sets which are \{Bread\}, \{Butter\}, \{Eggs\}, and \{Milk\}.
$L_1$ consists of those 1-itemsets from $C_1$ with support greater than or equal to 0.4. $C_2$
Figure D.1: Discovering large itemsets using the Apriori algorithm

is formed by joining $L_1$ with itself, and deleting any itemsets which have subsets not in $L_1$. This way, we obtain $C_2$ as $\{\{\text{Bread Butter}\}, \{\text{Bread Eggs}\}, \{\text{Butter Eggs}\}\}$. Counting support of $C_2$, $L_2$ is found to be $\{\{\text{Bread Butter}\}, \{\text{Butter Eggs}\}\}$. Using $\text{apriori\_gen}()$, we do not get any candidate itemsets for the third round. This is because the conditions for joining $L_2$ with itself are not satisfied.

### D.2 Incremental updating of association rules

In this section we will concentrate on the problem of maintaining the discovered association rules. Considering market basket analysis for example, the sales database is not a static database, because updates are constantly being applied to it. New records are frequently added and old records are deleted from the database because they are out of interest or to save storage space. In addition, existing records may be edited or changed to correct manual operational errors or for other reasons. Because of these update activities, the database keeps on changing. Consequently, new association rules may appear and some existing ones would become invalid. Thus, maintenance of the discovered association rules is an important problem.

Assuming that the two thresholds, minimum support and confidence do not change, there are several important characteristics in the update problem [Cheung et al., 1996]:

- The update problem can be reduced to finding the new set of large itemsets.

After that, the new association rules can be computed from the new large itemsets.
D.2. Incremental updating of association rules

- An old large itemset has the potential to become small in the updated database.
- An old small itemset could become large in the new database.
- To find the new large itemsets, all the records in the updated database, including those from the original database, have to be checked against every candidate set.

The problem of maintaining association rules was first studied in [Cheung et al., 1996]. That paper proposes the FUP algorithm, which can update the association rules in a database when new transactions are added to the database. A more general algorithm FUP2 was proposed in [Cheung et al., 1997]. This latter algorithm can update the discovered association rules when new transactions are added to and deleted from the database. In this section we summarize the main ideas of these two articles.

D.2.1 The FUP Algorithm

The FUP algorithm is based on the framework of Apriori and it also finds new large itemsets iteratively. The idea is to store the counts of all the large itemsets found in a previous mining operation. Using these stored counts and examining the newly added transactions, the algorithm can generate a very small number of candidate new large itemsets. The overall count of these candidate itemsets are then obtained by scanning the original database. Consequently, all new large itemsets are found.

Let $L$ be the set of large itemsets in the database $DB$, $s$ the minimum support, and $D$ be the number of transactions in $DB$. Assume that for each $X \in L$, its support count $X.support$, which is the number of transactions in $DB$ containing $X$, is available.

After some update activities, an increment $db$ of new transactions is added to the original database $DB$, and $d$ is the number of transactions in $db$. With respect to the same minimum support $s$, an itemset $X$ is large in the updated database $DB \cup db$ if the support of $X$ in $DB \cup db$ is no less than $s$, i.e. $X.support \geq s * (D + d)$.

Thus, the essence of the problem of updating association rules is to find the set $L'$ of large itemsets in $DB \cup db$. Note that a large itemset in $L$ may not be a large itemset in $L'$, on the other hand, an itemset $X$ not in $L$, may become a large itemset in $L'$.

The framework of FUP is similar to that of Apriori. It contains a number of iterations. The first iteration considers size-one itemsets, and at each iteration, all large itemsets of the same size are found. The candidate sets at each iteration are
generated based on the large itemsets found at the previous iteration. The main features of FUP are the following:

- At each iteration, the supports of the size-k large itemsets in $L$ are updated against the increment $db$ to filter out those losers, i.e. those that are no longer large in the updated database. Only the increment $db$ has to be scanned to do the filtering.

- While scanning the increment, a set of candidate sets, $C_k$, is extracted from the transaction in $db$, together with their supports in $db$ counted. The supports of these sets in $C_k$ are then updated against the $DB$ to find new large itemsets.

- Many sets in $C_k$ can be pruned away by a simple check on their supports in $db$ before the update against $DB$ starts.

The following notation defined in [Cheung et al., 1996] are used in the remaining of the Appendix. $L_k$ is the set of all size-k large itemsets in $Db$ and $L'_k$ is the set of all large k-itemsets in $DB \cup db$. $C_k$ is the set of size-k candidate sets in the k-th iteration of FUP. $X.support_D$, $X.support_d$ and $X.support_{UD}$ represent the support counts of an itemset $X$ in $DB$, $db$ and $DB \cup db$, respectively.

Algorithm 18 shows the first iteration of the FUP algorithm, i.e. the finding of large 1-itemset $L'_1$ in the updated database $DB \cup db$. These steps are outlined below.

1. Scan the increment $db$, for all itemsets $X \in L_1$, update its support count $X.support_{UD}$. Once the scan is completed, all the losers in $L_1$ are found by checking the condition $X.support_{UD} < s \times (D + d)$ on all $X \in L_1$. By removing the losers, the itemsets in $L_1$ which remain large after the update are identified.

2. In the same scan, a set $C_1$ is created to store, for each $T \in db$, all size-one itemset $X \subseteq T$ which is not in $L_1$. This becomes the set of candidate sets and their support in $db$ can also be found in the scan. If $X \in C_1$ and $X.support_d < s \times d$, $X$ can never be large in $DB \cup db$. Because of this, all the sets in $C_1$, whose support counts are less than $s \times d$, are pruned off. This gives us a very small candidate set for finding the new size-one large itemsets.

3. A scan is then conducted on $DB$ to update the support count $X.support_{UD}$ for each $X \in C_1$. By checking their support count, new large itemsets from $C_1$ are found. By combining with those identified in $L_1$, the set of all size-one large itemsets $L'_1$ is generated.
Algorithm 18 FUP - First Iteration

**Input:** $DB$: the original database (with its size equal to $D$); $L_k$: the set of all large k-itemsets in $DB$; $db$: an increment database with size equal to $d$; $s$: the minimum support threshold

**Output:** $L_1'$: the set of all large itemsets in $DB$ union $db$

1. $W \leftarrow L_1$ \{winners\}
2. $C \leftarrow \odot$ \{candidates\}
3. $L_1' \leftarrow \odot$
4. $P \leftarrow \odot$ \{for optimization\}
5. \textbf{for} all $T \in db$ \textbf{do}
6. \hspace{1em} \textbf{for} all 1-itemset $X \subseteq T$ \textbf{do}
7. \hspace{2em} \textbf{if} $X \in W$ \textbf{then}
8. \hspace{3em} $X.support_d$ ++
9. \hspace{2em} \textbf{else}
10. \hspace{3em} \textbf{if} $X \notin C$ \textbf{then}
11. \hspace{4em} $C \leftarrow C \cup \{X\}$
12. \hspace{4em} $X.support_d = 0$
13. \hspace{3em} \textbf{end if}
14. \hspace{3em} $X.support_d$ ++ \{init the support count and add X into C\}
15. \hspace{2em} \textbf{end if}
16. \hspace{1em} \textbf{end for}
17. \textbf{end for}
18. \textbf{for} all $X \in W$ \textbf{do}
19. \hspace{1em} \textbf{if} $X.support_{UD} \geq s \ast (D + d)$ \textbf{then}
20. \hspace{2em} $L_1' \leftarrow L_1' \cup \{X\}$ \{put winners into $L_1'$\}
21. \hspace{1em} \textbf{end if}
22. \textbf{end for}
23. \textbf{for} all $X \in C$ \textbf{do}
24. \hspace{1em} \textbf{if} $X.support_d < s \ast d$ \textbf{then}
25. \hspace{2em} $C \leftarrow C - \{X\}$ \{prune candidate sets in C\}
26. \hspace{2em} $P \leftarrow P \cup \{X\}$
27. \hspace{1em} \textbf{end if}
28. \textbf{end for}
29. \textbf{for} all $T \in DB$ \textbf{do}
30. \hspace{1em} \textbf{for} all 1-itemset $X \subseteq T$ \textbf{do}
31. \hspace{2em} \textbf{if} $X \in C$ \textbf{then}
32. \hspace{3em} $X.support_D$ ++
33. \hspace{2em} \textbf{end if}
34. \hspace{2em} \textbf{if} $X \in P$ \textbf{then}
35. \hspace{3em} remove $X$ from $T$
36. \hspace{2em} \textbf{end if}
37. \hspace{1em} \textbf{end for}
38. \textbf{end for}
39. \textbf{for} all $X \in C$ \textbf{do}
40. \hspace{1em} \textbf{if} $X.support_{UD} \geq s \ast (D + d)$ \textbf{then}
41. \hspace{2em} $L_1' \leftarrow L_1' \cup \{X\}$ \{put winners into $L_1'$\}
42. \hspace{1em} \textbf{end if}
43. \textbf{end for}
44. \textbf{return} $L_1'$
Algorithm 19 the k-th iteration of the FUP algorithm. For example, consider
the finding of large 2-itemset $L_2'$ in the updated database $DB \cup db$. This process is
outlined as follows:

1. Similar to the first iteration, losers in $L_2$ will be filtered out in a scan on $db$.
The filtering is done in two steps. First, some losers in $L_2$ can be filtered
out without checking them against $db$. The set of losers $L_1 - L_1'$ have been
identified in the first iteration. Therefore, any set $X \in L_2$, which has a subset
$Y$ such that $Y \subseteq L_1 - L_1'$, cannot be large and are filtered out from $L_2$ without
checking against $db$ and the support count of the remaining sets in $L_2$ are
updated and the large itemsets from $L_2$ are identified.

2. Similar to the first iteration, the second part at this iteration is to find the new
size-two large itemsets. The key is to generate a small set of candidate sets.
The set of candidate sets, $C_2$, is generated, before the above scan on $db$ starts,
by applying the $apriori\_gen$ function on $L_1'$. The sets in $L_2$ are excluded when
creating $C_2$ because they have already been handled. The support count of
the itemsets in $C_2$ can now be pruned by checking their support count. For
all $X \in C_2$, if $X.support_d < s \ast d$, $X$ is removed from $C_2$. All the removed sets
cannot be large in $DB \cup db$.

3. The last step is to scan $DB$ to update the support count for all itemsets
in $C_2$. At the end of the scan, all the sets $X \in C_2$, whose support count
$X.support_{UD} < s \ast (D + d)$, are identified as the new large itemsets. The set
$L_2'$, which contains all the large itemsets identified from $L_2$ and $C_2$ above, are
the set of all the size-two large itemsets.

The same algorithm is applied to the later iterations until no large itemsets is found.

**D.2.2 The FUP2 Algorithm**

The FUP is very efficient. However, the algorithm does not handle the case of
deleting transactions from the database. The FUP2 [Cheung et al., 1997] algorithm
can update the existing association rules when new transactions are added and
deleted form the database. It is a generalization of the FUP algorithm. Like FUP,
FUP2 makes use of the previous mining result to cut down the amount of work that
has to be done to discover the new set of rules.

After some activities, old transactions are deleted from the database $D$ and new
transactions are added. Let $\Delta -$ be the set of deleted transactions and $\Delta +$ be the
Algorithm 19 FUP - The k-th Iteration

**Input:** DB: the original database (with its size equal to D); $L_k$: the set of all large k-itemsets in DB; $db$: an increment database with size equal to d; s: the minimum support threshold

**Output:** $L'_k$: the set of all large itemsets in DB union db

1. $W \leftarrow L_k$ \{$winners$\}
2. $C \leftarrow \text{apriori} - \text{gen}(L'_k) - L_k$
3. $L'_k \leftarrow \emptyset$
4. for all k-itemset $X \in W$ do
   5. for all (k-1)itemset $Y \in L_{k-1} - L'_{k-1}$ do
      6. if $Y \subseteq X$ then
         7. $W \leftarrow W - \{X\}$
         8. break
      end if
   end for
   9. end for
10. end for
11. for all $T \in db$ do
12. for all $X \in \text{Subset}(W, T)$ do
13. $X.\text{support}_d$ ++ \{$\text{Subset}(W, T)$ returns all the sets in W contained in T$\}$
14. end for
15. end for
16. for all $T \in \text{Subset}(C, T)$ do
17. $X.\text{support}_d$ ++ \{$\text{find support of all X in C}$\}
18. end for
19. end for
20. for all $X \in W$ do
21. if $X.\text{support}_{UD} \geq s * (d + D)$ then
22. $L'_k \leftarrow L'_k \cup \{X\}$ \{$\text{put winners from W in Lk'}$\}
23. end if
24. end for
25. for all $X \in C$ do
26. if $X.\text{support}_d < s * d$ then
27. $C \leftarrow C - \{X\}$ \{$\text{prune candidate sets in C}$\}
28. end if
29. end for
30. for all $T \in DB$ do
31. for all $X \in \text{Subset}(C, T)$ do
32. $X.\text{support}_d$ ++
33. end for
34. end for
35. for all $X \in C$ do
36. if $X.\text{support}_{UD} \geq s * (D + d)$ then
37. $L_k \leftarrow L'_k \cup \{X\}$ \{$\text{put winners from C in Lk'}$\}
38. end if
39. end for
40. end for
41. return $L'_k$
set of newly added transactions. We assume that $\Delta- \subseteq D$. The updated database is denoted by $D'$, i.e. $D' = (D - \Delta-) \cup \Delta+$. The set of unchanged transactions is denoted by $D- = D - \Delta- = D' - \Delta+$. The support count of an itemset $X$ in the original database $D$ is denoted by $\sigma_X$. The set of large itemsets in $D$ is $L$ and $L_k$ is the set of $k$-itemsets in $L$. The new support count of an itemset $X$ in the updated database $D'$ is denoted by $\sigma'_X$, and the set of large itemsets in $D'$ by $L'$. $L'_k$ is the set of $k$-itemsets in $L'$. $\delta_X^+$ is the support count of itemset $X$ in the database $\Delta+$ and $\delta_X^-$ that of $\Delta-$. $\delta_X = \delta_X^+ - \delta_X^-$ is the change of support count of itemset $X$ as a result of the update activities.

As a result of a previous mining on the old database $D$, we have already found $L$ and $\sigma_X \forall X \in L$. Thus, the update problem is to find $L'$ and $\sigma'_X \forall X \in L'$ given the knowledge of $D$, $D'$, $\Delta-$, $\Delta-$, $\Delta+$, $L$ and $\sigma_X \forall X \in L$.

First, we will address the special case for transaction deletion only, which can be considered as a complement of FUP. For the delete-only case, we have $\Delta+ = \emptyset$ and hence $D' = (D - \Delta-) \cup \emptyset = D - \Delta-$. $\delta_X^--0 \forall X \subseteq I$.

To discover the large itemsets in the updates database $D'$, the FUP2 algorithm executes iteratively. In the $k$-th iteration, all the large itemsets in $D'$ are found as follows. As in Apriori, we form a set of candidates $C_k$ which is a superset of $L'_k$. In the first iteration $C_1$ is exactly the set $I$. In subsequent iterations, $C_k$ is calculated from $L'_{k-1}$, the large itemsets found in the previous iteration, using the apriori-gen function. All the itemsets in $L'_k$ are guaranteed to be contained in $C_k$.

Next, we use the old large $k$-itemsets $L_k$ from the previous mining result to divide the candidate set $C_k$ into 2 parts: $P_k = C_k \cap L_k$ and $Q_k = C_k - P_k$. $P_k$ ($Q_k$) is the set of candidate itemsets that are previously large (small) with respect to $D$. The goal is to select those itemsets that are currently large with respect to $D'$.

With the partitioning, for all candidates $X \in P_k$, we already know its support count $\sigma_X$ from the previous mining results. We find out $\delta_X^-$ by scanning $\Delta-$. Then, we can obtain the updated support count $\sigma'_X$, $\sigma'_X = \sigma_X + \delta_X^+ - \delta_X^- = \sigma_X + \delta_X$. Thus, a candidate $X$ from $P_k$ goes to $L'_k$ if and only if $\sigma'_X \geq |D| * s\%$.

For the candidates in $Q_k$, we only know that they were not large in the original database $D$. We do not know their support counts. However, since they were not large, we know that $\sigma_X < |D| * s\% \forall X \in Q_k$. We can make use of this information to tell which candidates from $Q_k$ may be large and which will not. For each candidate in $Q_k$, if it is large in $\Delta-$, then it cannot be large in $L'_k$. We first scan $\Delta-$ and obtain $\delta_X$ for each $X \in Q_k$. Then, we delete those candidates for which $\delta_X \geq |\Delta-| * s\%$, thus leaving in $Q_k$ those that are small in $\Delta-$. For the candidates $X$ that remain in $Q_k$, we scan $D-$ to obtain their new support counts $\sigma'_X$. Finally, we add to $L'_k$
those candidates $X$ from $Q_k$ for which $\sigma'_X \geq |D| * s\%$.

Thus, we have discovered which candidates from $P_k$ and $Q_k$ are large and put them into $L'_k$. We have also found out $\sigma'_X$ for each $X \in L'_k$. We have completed one iteration. In the subsequent iterations, large itemsets of larger sizes are discovered. The iterations go until either $C_k = \emptyset$ or $|L'_k| < k + 1$ for some $k$. The steps of the $k$th-iteration are summarized as follows:

1. Obtain a candidate set $C_k$ of itemsets. Halt if $C_k = \emptyset$. 
2. Partition $C_k$ into $P_k$ and $Q_k$, where $P_k = C_k \cap L_k$ and $Q_k = C_k - P_k$. 
3. Scan $\Delta-$ to find out $\delta_X^-$ for each $X \in P_k \cup Q_k$. 
4. For each $X \in P_k$, calculate $\sigma'_X$. 
5. Delete from $Q_k$ those candidates $X$ where $\delta_X^- \geq |\Delta - | * s\%$. 
6. Scan $\Delta-$ to find out $\sigma'_X$ of the remaining candidates $X \in Q_k$. 
7. Add to $L'_k$ those candidates $X$ from $P_k \cup Q_k$ for which $\sigma'_X \geq |D| * s\%$. 
8. Halt if $|L'_k| < k + 1$

In the general case for transaction deletion and insertion, we no longer assume that $\Delta^+ = \emptyset$. So, $\delta_X^+$ may be positive for any $X \subseteq I$. As before, we find out $L'_k$ and $\sigma'_X \forall X \in L'_k$ in the $k$th iteration. In each iteration we first form a candidate set $C_k$, and then partition it into two parts $P_k$ and $Q_k$ as before. Again for each candidate $X \in P_k$, we know $\sigma_X$ from the previous mining result. So, we only have to scan $\Delta-$ and $\Delta+$ to update the support count for the candidates in $P_k$. In the FUP2, $\Delta-$ is scanned first to find $\delta_X^-$ for each candidate $X$. As $\Delta-$ is scanned, we can deduct the support count at the same time, and remove a candidate from $P_k$ as soon as its support count drops below $|D'| * s\% - |\Delta-|$. This is because such a candidate has no hope to have $\sigma'_X \geq |D'| * s\%$, as $\delta_X^- \leq |\Delta + |$. Next, we scan $\Delta+$ to find each $\delta_X^+$ for each candidate in $P_k$. Finally, we calculate $\sigma'_X$ for each candidate in $P_k$, and add those with $\sigma'_X \geq |D'| * s\%$ to $L'_k$.

For the candidates $X \in Q_k$, again we do not know $\sigma_X$, but we know that $\sigma_X < |D| * s\%$. We are able to prune some candidates from $Q_k$ without knowing their counts in $D-$. According to Lemma 4 in [Cheung et al., 1997], for each candidate $X$ in $Q_k$, we obtain the values of $\delta_X^+$ and $\delta_X^-$ during the scans of $\Delta+$ and $\Delta-$. Then, we calculate $\delta_X$ and remove those with $\delta_X \leq (|\Delta+| - |\Delta-|) * s\%$ because they will not fall into $L'_k$. For the remaining candidates in $Q_k$, we scan $D-$ and obtain their
D.2. Incremental updating of association rules

support counts in $D_\leftarrow$. Adding this count to $\delta_X^+$ gives $\sigma'_X$. We add those candidates with $\sigma'_X \geq |D'| \times s\%$ from $Q_k$ to $L'_k$. This finishes the iteration.

Here is the final version of FUP2 (without the optimizations proposed in [Cheung et al., 1997]), for iteration $k$, where $k \geq 2$. For the first iteration, set $C_1 = I$.

1. Obtain a candidate set $C_k$ of itemsets. Halt if $C_k = \emptyset$.
2. Partition $C_k$ into $P_k$ and $Q_k$.
3. Scan $\Delta^-\rightarrow$ to find out $\delta_X^-$ for each $X \in P_k \cup Q_k$.
4. Scan $\Delta^+\rightarrow$ to find out $\delta_X^+$ for each $X \in P_k \cup Q_k$.
5. For each candidate $X \in P_k$, calculate $\sigma'_X$.
6. For each candidate $X \in Q_k$, delete $X$ if $\delta_X^+ - \delta_X^- \leq (|\Delta^+| - |\Delta^-|) \times s\%$.
7. Scan $D_\leftarrow$ and get the count of each $X \in Q_k$. Then, add this count to $\delta_X^+$ to obtain $\sigma'_X$.
8. Add to $L'_k$ those candidates $X \in P_k \cup Q_k$ where $\sigma'_X \geq |D'| \times s\%$.
9. Halt if $|L'_k| < k + 1$.

D.2.3 The DELI Algorithm

Although the FUP and FUP2 algorithms are very efficient, in the long run they still have certain performance overhead. We can apply FUP2 each time a new transaction is added to or deleted from the database. However, the overhead of such a run of FUP2 is very high. The total long-term overhead will be large because the algorithm is applied too frequently. On the other hand, if we always wait until a large amount of updates has accumulated before applying FUP2, we will not be able to discover the newest association rules quickly. This is not desirable, especially in time-critical applications where decisions have to be made as soon as a new phenomenon is observed. To have best results, the FUP and FUP2 algorithms should be run at suitable times.

A first intuition if to use the amount of updates as an indicator. When the amount of accumulated updates exceeds a certain threshold, FUP2 is applied. However, if the updates follow the same association rule patterns as the old database, then both the old and the new databases would exhibit the same association rules. Despite the large amount of updates, few useful new association rules can be found. Another intuitive approach is to find out the new association rules in the updated
D.2. Incremental updating of association rules

Database, and then compare them with the old rules to find out their difference. The size of the difference is a good indicator for the necessity to discover the new association rules. However, calculating this size requires us to find out the new association rules first, and this operation involves great overhead.

To solve this problem, in [Lee and Cheung, 1997] the DELI (Difference Estimation for Large Itemsets) algorithm is proposed. In this section, we summarize this paper. DELI uses a sampling technique to estimate the difference between the old and new association rules. This estimate is used as an indicator for whether the FUP2 algorithm should be applied to the database to accurately find out the new association rules. If the estimated difference is large enough with respect to some thresholds, the algorithm signals the need of an update operation. If the estimated difference is small, then we do not run FUP2 immediately. In this case, we can take the old rules as an approximation of the new rules.

D.2.4 Estimation of the difference between old and new association rules

For the update problem, we have to find $L'$ and $\sigma'_X \forall X \in L'$. But, first, we would like to estimate the difference between $L$ and $L'$. If the difference is small, then we do not update the association rules, but wait for more updates to come. If, however, the difference is larger than a given threshold, then enough updates have been accumulated and an update operation is necessary.

The difference between the old large itemsets $L$ and the new large itemsets $L'$ can be measured by the set symmetric difference between them. We use the notation $L \oplus L'$ to denote the symmetric difference between $L$ and $L'$. Note that

$$L \oplus L' = L' \oplus L = (L' - L) \cup (L - L')$$

Depending on the similarity between $L$ and $L'$, the size of $L \oplus L'$ can vary between 0 and $|L| + |L'|$. It is 0 when $L = L'$ and it is $|L| + |L'|$ when $L$ and $L'$ are disjoint. The smaller the size of $L \oplus L'$, the greater the similarity between $L$ and $L'$.

The ratio $\frac{|L \oplus L'|}{|L| + |L'|}$ can be used as a relative measurement of the difference between $L$ and $L'$. Since in this estimation problem we do not know $L'$, we cannot calculate the above ratio. Thus, Lee and Cheung chose to use the ratio $\frac{|L \oplus L'|}{|L|}$ instead. We will use this ratio as a difference measure for the old and new large itemsets. Since $L$ is known to us from the results of the last mining, it remains to estimate the value of $L \oplus L'$. The estimation problem is to efficiently estimate the size $|L \oplus L'|$ without finding out $L'$, given the knowledge of $D$, $D'$, $\triangle +$, $\triangle -$, $D-$, $L$ and $\sigma_x \forall X \in L$. 

D.2.5 Drawing a random sample

Consider the original database \( D \) and an arbitrary itemset \( X \subseteq I \). Since \( \sigma_x \) of the \( |D| \) transactions in the database contain \( X \), the probability that a transaction randomly selected from \( D \) contains \( X \) is \( p_x = \frac{\sigma_x}{|D|} \).

Suppose we draw \( m \) transactions from \( D \) with replacement to form a sample \( S \). Each transaction is drawn independently. As a result, each transaction in \( S \) has a probability of \( p_x \) of containing the itemset \( X \). Let the total number of transactions in \( S \) containing \( X \) be \( T_X \). Then, \( T_X \) is a binomially distributed random variable with parameters \( m \) and \( p_x \). With \( m \) sufficiently large (\( \geq 30 \)), \( T_X \) can be approximated by a normal distribution with mean \( m \cdot p_x \) and variance \( m \cdot p_x \cdot (1 - p_x) \). We can estimate the value of \( \sigma_x \) by the point estimator \( \hat{\sigma}_x = \frac{T_X}{m} \cdot |D| \). This is an unbiased estimator because \( \hat{\sigma}_x \) is normally distributed with mean \( m \cdot p_x \cdot \frac{|D|}{m} = \sigma_x \).

Its variance is \( m \cdot p_x \cdot (1 - p_x) \cdot \left( \frac{|D|}{m} \right)^2 = \sigma_x(\|D| - \sigma_x)/m \). Thus, we can obtain a 100(1-\( \alpha \))% confidence interval \([a_x, b_x]\) for \( \sigma_x \) where:

\[
a_x = \hat{\sigma}_x - z_{\alpha/2} \sqrt{\frac{\sigma_x(\|D| - \sigma_x)}{m}} \\
b_x = \hat{\sigma}_x + z_{\alpha/2} \sqrt{\frac{\sigma_x(\|D| - \sigma_x)}{m}}
\]

and \( z_{\alpha/2} \) is the critical value such that the area under the standard normal curve beyond \( z_{\alpha/2} \) is exactly \( \alpha/2 \). The value of \( \alpha \) is chosen by the user. The 100(1-\( \alpha \))% confidence interval for \( \sigma_x \) has the property that \( \Pr(\sigma_x \in [a_x, b_x]) = 100(1-\alpha)\% \). So, there is 100(1-\( \alpha \))% chance that the actual value of \( \sigma_x \) lies on the interval.

D.2.6 Estimating the symmetric difference

The DELI algorithm is iterative. In the k-th iteration, the algorithm first generates a set \( C_k \) of candidates itemsets. For the first iteration, \( C_1 \) is the set of 1-itemsets. For subsequent iterations, \( C_k \) is calculated by applying \text{apriori-gen} on the set \( \hat{L}_{k-1} \) calculated in the previous iteration as described below. \( \hat{L}_{k-1} \) is an approximation of \( L'_k \). The candidate set \( C_k \) so generated may not cover all the itemsets in \( L_k' \). However, itemsets missed by the approximation \( \hat{L}_{k-1} \) most likely have very small support counts, and hence candidates generated by them are unlikely to be large. So, \( C_k \) should cover most of the itemsets in \( L_k' \), and the missed ones are unimportant ones.

After its generation, the set \( C_k \) is partitioned into two parts as the FUP2 algorithm: \( P_k = C_k \cap L_k \) and \( Q_k = C_k - P_k \). \( P_k \) contains all the candidates that were
D.2. Incremental updating of association rules

large in the old database, while $Q_k$ contains those that were not large. Candidates in these partitions are treated differently. For each itemset $X \in P_k$, its old support count $\sigma_x$ in the old database can be retrieved from the previous mining results. Then, we find out $\delta_X^+$ and $\delta_X^-$ by scanning the updates $\Delta^+$ and $\Delta^-$. Thus, the new support count $\sigma_x'$ can be calculated. Those itemsets $X$ with $\sigma_x' \geq |D'| * s\%$ are large in the updated database. We add them to a set $L_k^{(\geq)}$. Note that the itemsets $Y \in (L_k - C_k) \cup (P_k - L_k^{(\geq)}) = L_k - L_k'$ are those that were large in the old database but not large in the new database. We count the number of such itemsets and denote this number by $\eta_k$.

For the candidates $X$ in $Q_k$, we do not know $\sigma_X$, but we know that $\sigma_X < |D| * s\%$. We can first find out $\delta_X$ by scanning the updates $\Delta^+$ and $\Delta^-$. Then, we are able to prune away some of the candidates: if $\delta_X \leq (|\Delta^+| - |\Delta^-|) * s\%$ then $\sigma_X = \sigma_X + \delta_X < (|D| + |\Delta^+| - |\Delta^-|) * s\% = |D'| * s\%$ and hence $X$ cannot be large. So, none of the itemsets $X$ in $Q_k$, satisfying $\delta_X \leq (|\Delta^+| - |\Delta^-|) * s\%$ can be large. They can be pruned away from $Q_k$. For each of the remaining candidates $X \in Q_k$, we use the method described before to find out a $100(1-\alpha)\%$ confidence interval $[a_x, b_x]$ for $\sigma_x$. This corresponds to the interval $[a_x + \delta_X, b_x + \delta_X]$ for $\sigma_x'$. Since the criterion for being large is $\sigma_x' \geq |D'| * s\%$, we compare this threshold value with the interval for $\sigma_x'$. There are three possibilities:

1. If $|D'| * s\% < a_x + \delta_X$, then $\sigma_x'$ has a large chance to exceed the threshold. Thus, it is likely to be large. We add this itemset to the set $L_k^{(\geq)}$

2. If $a_x + \delta_X \leq |D'| * s\% \leq b_x + \delta_X$, then $\sigma_x'$ may or may not exceed the threshold. So, it may be large or not. Such a candidate is added to the set $L_k^{(\approx)}$

3. If $b_x + \delta_X < |D'| * s\%$, then $\sigma_x'$ has little chance to exceed the threshold. So, the itemset is very likely to be not large. The candidate is dropped.

After all the candidates in $Q_k$ has been handled as described above, we calculate the value $\xi_k = |L_k^{(\geq)}| + |L_k^{(\approx)}|$. This value is considered as an upper bound for $|L_k' - L_k|$.

This completes one iteration. We have not found the exact value of the set of new large itemsets $L_k'$. We can make a conservative guess $\hat{L}_k = L_k^{(\ll)} \cup L_k^{(\geq)} \cup L_k^{(\approx)}$. The set $\hat{L}_k$ should contain most of the itemsets in $L_k'$.

Whether the next iteration is performed depends on several criteria. The first criterion is the degree of uncertainty that is introduced by our estimation of $\sigma_X$ with the confidence interval $[a_x, b_x]$ for the candidates in $Q_k$. The uncertainties so introduced are gathered in the set $L_k^{(\approx)}$. So, we can compare the uncertainty factor $u_k = \frac{|L_k^{(\approx)}}{|L_k|}$ against a predefined threshold $\bar{u}$. If this factor is too large, then the
amount of uncertainty introduced in this iteration is too large to reliably generate
the candidates for the next iteration. So, we declare that a good estimation cannot
be made and FUP2 should be run to do an accurate update.

The second criterion is the amount of changes in the set of large itemsets caused
by $\Delta+$ and $\Delta-$. The value $\eta_k + \xi_k$ is an approximate upper bound to $|L_k \ominus L'_k| = |L'_k - L_k| + |L_k - L_k|$. Thus, we test the ratio $d_k = \sum_{j=1}^{k} (\eta_j + \xi_j)/|L|$ against a
user-specified threshold $\bar{d}$. If this threshold is exceeded, then the set of old large
itemsets are so different from the set of new large itemsets that an accurate update
is deemed necessary. So, FUP2 should be called to do the update.

The third criterion is to see if the number of large itemsets generated in this
iteration is sufficient to generate any candidates for the next iteration. If so, the
algorithm continues with the next iteration. If not, the algorithm terminates. If the
algorithm comes to this point of termination, the amount of uncertainty and the
size of symmetric difference $L \ominus L'$ has not exceeded their thresholds, since $u_k \geq \bar{u} \wedge d_k \geq \bar{d} \forall k$. So, we can conclude that we have enough certainty to say that the large
itemsets in the new database are not too different from the old database. Hence, it
is acceptable to take the large itemsets in the old database as an approximation of
the updated database. Below we present a summary of the algorithm.

1. Obtain a random sample $S$ of size $m$ from the original database. Set $k = 1$.
2. Generate a candidate set $C_k$. For $k = 1$, $C_1 = I$. For $k > 2$, $C_k = \text{apriori\_gen}(L_{k-1}^\hat{\ominus})$
3. Divide $C_k$ into 2 parts: $P_k = C_k \cap L_k$ and $Q_k = C_k - P_k$
4. Scan $\Delta+$ and $\Delta-$ to obtain $\delta_X$ for all $X$ in $Q_k$
5. For each $X \in P_k$, retrieve $\sigma_X$ from the results of the previous mining operation.
   Then, calculate $\sigma_X' = \sigma_X + \delta_X$. If $\sigma_X' \geq |D'| * s\%$, add $X$ to $L_k^{(\leq)}$
6. Calculate $\eta_k = |L_k - C_k| + |P_k - L_k^{(\leq)}|$
7. For each $X \in Q_k$, if $\delta_X \leq (|\Delta+| - |\Delta-|) * s\%$, delete it from $Q_k$
8. For the remaining $X \in Q_k$, obtain a 100(1-$\alpha$)% confidence interval $[a_x, b_x]$ for
   $\sigma_x$ by examining the sample $S$. Then, do one of the following for $X$: (a) if
   $|D'| * s\% < a_x + \delta_X$, add $X$ to $L_k^{(>)}$; (b) if $a_x + \delta_X \leq |D| * s\% \leq b_x + \delta_X$, add
   $X$ to $L_k^{(=)}$; (c) if $b_x + \delta_X < |D'| * s\%$, drop $X$.
9. Calculate $\xi_k = |L_k^{(>)}| + |L_k^{(=)}|$
10. Let $\hat{L}_k = L_k^{(\leq)} \cup L_k^{(>)} \cup L_k^{(=)}$
11. If $u_k \geq \bar{u}$, signal the need for an update operation.

12. If $d_k \geq \bar{d}$, signal the need for an update operation.

13. If $\hat{L}_k$ is non-empty, increment $k$ and go to step 2. Otherwise, conclude that $L \approx L'$ and hence use $L$ as an approximation for $L'$.
Appendix E

User Profiles Descriptions

In this Appendix we briefly describe the preferences stated by the users who participated in the experiments described in Chapter 8. For each we describe his/her preferences with respect to calendar management (what kind of events he/she schedules, preferred times, dates and places), the type of assistance actions he/she prefers in different contexts, and the modality of the assistance expected in different contexts.

E.1 User 1

E.1.1 Type of User

Recalling the classification of users given in Chapter 2, this user can be classified as intermediate.

E.1.2 Preferences with respect to calendar management

The user usually schedules meetings with his boss, work-mates, and subordinates at the office. The user schedules classes and meetings with different professors and students at the university. The user goes frequently to the doctor. The user goes out with friends. They go to the cinema, theater and restaurants.

E.1.3 User Assistance Requirements

- When two events overlap, and they are business meetings, meetings at the university, courses or exams, the user expects the agent to make a suggestion and eventually make an action on the user’s behalf.
E.2. User 2

- When the overlapping events are personal events such as an appointment with the doctor, an appointment with the dentist, a gym class, a party, a dinner, the user wants the agent to warn him.

- If the user is scheduling a business event for a holiday and the organizer is the user’s boss or by another person different from the user, the agent must warn the user.

- If the user is scheduling a business event for a holiday and the organizer is the user, the user wants the agent to make a suggestion and eventually perform an action on the user’s behalf.

- If the user is going to be late to an event, he wants the user to warn him about this inconvenient.

E.1.4 User Interruption Preferences

- If the user is scheduling a personal event for a holiday such as a party, a gym class, a dinner, the agent should just notify him.

- If the user is scheduling a meeting or a course for a holiday, the agent should interrupt him.

- If an event organized by the user or by a friend of the user overlaps with another event, the user wants to be notified.

- If an event taking place at the university or organized by a professor overlaps with another event, the user expects to be interrupted.

- If the user might get late to a business meeting, he expects the agent to interrupt him.

E.2 User 2

E.2.1 Type of User

As regards interface agents, the user is an expert.

E.2.2 Preferences with respect to calendar management

The user studies Computer Science Engineering and he works in a small company, in which he is in charge of a reduced group of employees. The user schedules business
meetings, classes and exams. The user enjoys going to the cinema and to the theater. The user goes to a gym in the evenings. The user is often invited to parties and dinners.

E.2.3 User Assistance Requirements

- If the user is organizing an event for a holiday and the employees take part, the user expects a suggestion
- If the user is scheduling a meeting for a holiday, the user expects a warning.
- If two events overlap and one of them is an appointment with the doctor, the user expects a suggestion
- If an exam overlaps with another event, the user expects a warning
- If a business meeting overlaps with another event, the user wants a suggestion
- If the user is going to arrive late to an event and the event is a business meeting or a meeting at the university the user wants a warning.
- If the user may arrive late to an event organized by him, the user does not want any assistance.

E.3 User 3

E.3.1 Type of User
The user is an expert regarding interface agents.

E.3.2 Preferences with respect to calendar management
The user works at a software company. Thus, he has business meeting with his boss and/or subordinates to discuss about the projects he works in. These meetings are generally on Monday mornings. The user goes to the gym with a group of friends after 7 pm. The user generally goes to the cinema or goes out to dinner with his friends. The user works as a teacher assistant at the university. He usually schedules meetings with a professor on Friday afternoons.
E.3.3 User Assistance Requirements

- If a business meeting overlaps with a meeting at the university, the user expects a warning.

- If a business meeting overlaps with another type of meeting, the user expects an action on the user’s behalf. The agent should reschedule the other meeting.

- If a meeting organized at the user’s home overlaps with another meeting, the user expects a suggestion.

- If a meeting at the university overlaps with another meeting, the user expects an action on the user’s behalf. The agent should reschedule the other meeting.

- If the user is scheduling a new business meeting or a meeting at the university, the user expects an action.

- If the user might arrive late to an event, the agent should change the event time provided that the organizer is the user or a friend of the user.

E.3.4 User Interruption Preferences

- If a business meeting overlaps with a meeting at the university, the user expects an interruption.

- If a business meeting overlaps with another type of meeting, the user expects a notification.

- If a meeting organized at the user’s home overlaps with another meeting, the user expects an interruption.

- If a meeting at the university overlaps with another meeting, the user expects a notification.

- If the user is scheduling a business event for a holiday, he wants an interruption.

- If the user is scheduling a meeting at the university for a holiday, he wants an interruption.

- If the user receives an invitation for a new event, he wants an interruption only if it is a business event.

- If the user might arrive late to an event, he wants a notification.
E.4 User 4

E.4.1 Type of User
The user is inexpert as regards interface agent technology.

E.4.2 Preferences with respect to calendar management
The user works in a training center from 9 a.m. to 1 p.m and from 3 p.m. to 8 p.m. She teaches at a primary school on Thursday mid-days. She is taking a postgraduate course on Saturdays. She generally trains or studies with a friend. She usually goes to the cinema or dinners.

E.4.3 User Assistance Requirements
- If an event overlaps with a work-related event, the user expects a suggestion about how to reschedule the overlapping event.
- If a postgraduate course overlaps with another event, the user expects a suggestion about how to reschedule the overlapping event.
- If two personal events overlap (training, dinners, cinema), the user wants just a warning.
- If a work-related event is being scheduled for a holiday, the user expects a warning.
- If a university-related event is being scheduled for a holiday, the user does not want any assistance.
- If the user is scheduling a new event in which friends take part, the user wants an action.
- If the user might get late to an event organized by herself, she expects an action.
- If the user might get late to an event organized by her friends or classmates, she expects an action.

E.4.4 User Interruption Preferences
- If an event overlaps with a work-related event, the user expects an interruption.
• If a postgraduate course overlaps with another event, the user expects an interruption.

• If two personal events overlap (training, dinners, cinema), the user wants just a notification.

• If a work-related event is being scheduled for a holiday, the user expects an interruption.

• If an invitation to a new work-related event arrives, the user wants an interruption.

E.5 User 5

E.5.1 Type of User

According to Chapter 2, the user is inexpert.

E.5.2 Preferences with respect to calendar management

The user studies at the university, everyday from 8 a.m. to 5 p. m. The user also works as a teaching assistant. Thus, she schedules meetings with a professor and classes. The user works during the weekends from 3 p.m. to 10 p.m. The user takes gym classes three times a week after 8 p.m. The user has health problems, so she generally schedules appointments with the doctor. These appointments are generally during the evenings.

E.5.3 User Assistance Requirements

• If a work-related event overlaps with another type of event, the user expects the agent to reschedule the event on his behalf.

• If a meeting at the university, a class or a course overlap with an appointment with the doctor, the user expects a suggestion about how to change the appointment.

• If a meeting at the university, a class or a course overlap with another event, the user expects a suggestion.

• If a gym class overlaps with another personal event, the user wants a warning.
• If the user schedules a university-related event for a holiday, she wants a warning.

• If the user is scheduling an appointment with the doctor, she expects a suggestions from the agent.

• If the user is scheduling an appointment with the dentist, she expects a suggestions from the agent.

• If the user is scheduling a work-related or university-related event, she expects no assistance.

• If the user might get late to a university-related event, she wants a suggestion.

E.5.4 User Interruption Preferences

• If a work-related event overlaps with another type of event, the user expects a notification.

• If a meeting at the university, a class or a course overlap with an appointment with the doctor, the user expects an interruption.

• If a meeting at the university, a class or a course overlap with another event, the user expects a notification.

• If a gym class overlaps with another personal event, the user wants a notification.

• If the user schedules a university-related event for a holiday, she wants to be notified.

• If the user schedules an appointment with the doctor for a holiday, she wants to be interrupted.

• If the user schedules an appointment with the dentist for a holiday, she wants to be interrupted.

• If the user schedules a university-related or work-related event for a holiday, she expects an interruption.
E.6  User 7

E.6.1  Type of User

The user is in an intermediate level.

E.6.2  Preferences with respect to calendar management

The user works at an electric company. He attends business meetings of high priority that take place on Wednesday mornings. The user is sitting for exams at the university, and taking some courses. He usually goes to the gym after 7 p.m.

E.6.3  User Assistance Requirements

- If a business meeting overlaps with another event, the user wants the agent to make a suggestion on how to reschedule this event.

- If two high priority events overlap, the user expects a suggestion.

- If a gym class overlaps with other personal event the user wants an action on his behalf, rescheduling the gym class.

- If the user is scheduling a new business event, he expects a warning.

- If the user is scheduling a new event of low priority such as a gym class, he expects an action.

- If the user might arrive late to an event, he wants a warning.

E.6.4  User Interruption Preferences

- If a business meeting overlaps with another event, the user wants the agent to interrupt him.

- If two high priority events overlap (classes, exams, those that cannot be changed), the user expects an interruption.

- If a gym class overlaps with other personal event the user wants a notification.

- If a new event arrives, the user wants to be interrupted only if it is a business event.

- If the user is scheduling a business event for a holiday, he wants to be interrupted.
E.7 User 8

E.7.1 Type of User
The user is considered as intermediate.

E.7.2 Preferences with respect to calendar management
The user works at a software development company as a project leader.

E.7.3 User Assistance Requirements
- If the organizer of an overlapping event is the user’s boss, the agent should reschedule the other event.
- If the organizer of the overlapping events is the user, a work-mate or a subordinate, the agent should make a suggestion.
- If the organizer is the user’s boyfriend, a family member, or a friend, the agent should warn the user.
- If the user’s boss schedules a business event for a holiday, the user expects a suggestion.
- If the user, a work-mate or a subordinate organizes an event for a holiday, the user expects a suggestion.
- If the user might get late to a university-related event, she expects a suggestion.
- If the user might get late to an event organized by herself, she does not want assistance

E.8 User 10

E.8.1 Type of User
The user is in an intermediate level.

E.8.2 User Assistance Requirements
- If the user schedules a business event for a holiday organized by himself, he expects a suggestion.
• If the user schedules a business event for a holiday organized by his boss, he expects a warning.

• If the user schedules a business event for a holiday organized by a subordinate, he expects a warning.

• If the user might arrive late to a business dinner after a business meeting, he expects a suggestion.

E.9 User 16

E.9.1 Type of User

According to the user classification given in Chapter 2, the user is inexpert.

E.9.2 Preferences with respect to calendar management

The user works at a chemist’s from 9 a.m. to 1 p.m. The user studies at the university. He usually studies with some classmates at the afternoons. During the weekends, he goes to the cinema with some friends. He goes to the doctor often because he has a health problem that requires periodic analysis. He goes to the gym in the evening.

E.9.3 User Assistance Requirements

• If two events overlap, and they are exams, classes or work, the user wants a warning.

• If a class or an exam overlap with another type of event, the user wants a suggestion (to reschedule the other event).

• If an appointment with the doctor overlaps with a work-related event, the user expects a warning.

• If an appointment with the doctor overlaps with another event, the user expects a suggestion.

• If a gym class overlaps with a personal event, the user wants a warning.

• The user does not want autonomous actions.

• If the user schedules an exam or a class for a holiday, he expects a warning.
• If an invitation for a new event arrives, the user expects a warning.

• If the user might arrive late to a business event, a university-related event, a gym class or an event organized by a friend, he wants a warning.

E.9.4 User Interruption Preferences

• If the user is scheduling a work-related event for a holiday, he wants a notification.

• If the user is scheduling an appointment with the doctor for a holiday, he expects an interruption.

• If the user is scheduling a university-related event for a holiday, he wants to be interrupted.

• If two events overlap, and they are related to the user’s work or the university, the user wants an interruption.

• If the overlapping events are personal (gym classes, dinners, parties), the user wants a notification.

• If the overlapping events involve an appointment with the doctor, he wants an interruption.

• If an invitation to take part in a work-related or university-related event arrives, the user wants to be interrupted.

• If the user is arriving late to a business event, he wants to be interrupted.

• If the user is arriving late to a personal event, he wants a notification.

• If the user is arriving late to an appointment with the doctor, he wants a notification.

E.10 User 28

E.10.1 Type of User

The user is an expert regarding interface agent technology.
E.10.2 Preferences with respect to calendar management

The user takes gym classes twice a week, from 7.30 to 8.30 p.m. The user works at the university from 9 a.m. to 6 p.m. The user takes English classes once a week from 7 p.m. to 8 p.m. English classes are on Mondays or Fridays. If the user has to go to the doctor or dentist, she prefers to go after 6 p.m. She teaches on Tuesday mornings.

E.10.3 User Assistance Requirements

- If a gym class overlaps with another event, the user wants a warning.
- If an appointment with the doctor overlaps with another event, the user expects a suggestion.
- If two personal events (dinner, cinema, party) overlap, the user wants a warning.
- If a meeting overlaps with a class or with another meeting, the user expects a warning.
- If two classes overlap, the user expects a suggestion.
- If the user is scheduling a business event for a holiday, she expects a warning.

E.10.4 User Interruption Preferences

- If a business meeting organized by the user overlaps with a high priority event organized by somebody else, the user wants to be interrupted.
- In other cases, the user wants to be notified.
- If the user receives an invitation for an event organized by her superiors or that takes place that day, she wants to be interrupted.
- In other cases, she expects notifications.

E.11 User 30

E.11.1 Type of User

The user has an intermediate level with respect to the user and development of interface agents.
E.11.2 User Assistance Requirements

- If two events overlap and one of them is organized by the user, the user expects the agent to reschedule this event on his behalf.

- If two events overlap and one of them involves the projects the user is working with, the user expects the agent to reschedule this event on his behalf.

- If an event organized by the user’s boss overlaps with another event, the user expects a warning.

- If the agent might arrive late to an event organized by himself, he expects the agent to reschedule this event on his behalf.

- If the user is scheduling an event for a holiday and the organizer of this event is the user or the user’s boss, the user expects a warning.

E.12 User 31

E.12.1 Type of User

The user is intermediate with respect to interface agent technology.

E.12.2 User Assistance Requirements

- If the user schedules a course for a holiday, he expects a warning.

- If the user schedules a meeting for a holiday, he expects a suggestion.

- If the user schedules a business meeting for a holiday, he wants an action on his behalf.

- If a gym class overlaps with a course, the user expects an action.

- If two meetings overlap, the user wants the agent to suggest him how to reschedule them.

- If a business meeting overlaps with a personal event (party, cinema, dinner) the user wants just a warning.

- If the user might arrive late to an appointment with the doctor or dentist, he wants an action on his behalf.
• If the user might arrive to a meeting after another meeting, he expects a warning.

• If the user might arrive late to a party, he expects a warning.
Bibliography


Bibliography


