<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>An Infrastructure for Privacy-Enabled Sharing of Context Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Hynes, Gearoid</td>
</tr>
<tr>
<td><strong>Publication Date</strong></td>
<td>2011-08-29</td>
</tr>
<tr>
<td><strong>Item record</strong></td>
<td><a href="http://hdl.handle.net/10379/2166">http://hdl.handle.net/10379/2166</a></td>
</tr>
</tbody>
</table>

Some rights reserved. For more information, please see the item record link above.
An Infrastructure for Privacy-Enabled Sharing of
Context Data

Gearoid Hynes

Submitted in fulfillment of the requirements for the degree of
Doctor of Philosophy

SUPERVISOR:
Prof. Manfred Hauswirth

DIRECTOR:
Prof. Stefan Decker

Digital Enterprise Research Institute, National University of Ireland, Galway
August 2011
Declaration & Disclaimer

I declare that this thesis is composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

The work reported in this thesis was supported by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2) and by the European FP7 project PECES (ICT-224342).

Gearoid Hynes
Abstract

The concept of context originated in the ubiquitous computing domain where information regarding the user’s situation was gathered using sensing devices, allowing the application to automatically adapt according to the user’s current situation. Context awareness enabled the development of smart spaces in a multitude of different environments such as offices, homes, cars, hospitals and universities.

Traditionally, the smart space that produced the context data was the sole consumer of that data and accessing the context data from third-party applications was cumbersome. Consequently, smart spaces became stovepipes; having little integration beyond their isolated application scenarios. The development of context-aware applications was limited to environments with access to rich sources of context data and which had large resources available for the development of context-aware applications. The development of these context-aware applications required expertise in the areas of context modelling, context gathering, context management and data privacy. As a result, the inclusion of context-aware behaviours in applications outside of ubiquitous computing research prototypes has been limited.

This thesis defines and presents a distributed web-based model for the controlled sharing of context data between heterogeneous third-parties. A web-based approach is adopted in order to maximize the variety of potential applications, such as mobile, desktop, smart space and web-based applications. This ‘Context as a Service’ (CxaaS) approach to context management enables applications to access context-data from domains beyond their usual operational boundaries.

The support for ubiquitous computing scenario oriented sharing of context data has been addressed in three key areas: (i) discovery of context data from remote sources concerning new and returning users, (ii) enabling the privacy sensitive sharing of context data between heterogeneous third-parties, and (iii) provision of a formal context model appropriate for use with the discovery and privacy mechanisms.

The model defined in this thesis is evaluated with an implementation of a comprehensive business use-case. A RESTful approach is adopted in this particular instantiation of the CxaaS model which facilitates the rapid development of context-aware applications through the use of popular libraries. The core CxaaS context ontology is supported and mechanisms are provided to extend this model if necessary.
Acknowledgments

During my research I have received support and encouragement from a wide range of people. Firstly I would like to thank Vinny Reynolds and Professor Manfred Hauswirth for their guidance and knowledge, and I would like to thank Professor Birgitta Koenig-Ries for her time and for agreeing to be my external examiner.

I would like to thank my colleagues, especially Aidan Hogan, Fergal Monaghan, Vinhtuan Thai, Alex Schutz and all the members of the USM group, for their comments and insights. I also appreciate the work done by the administrative staff in DERI in keeping the institute running like clockwork; especially Hilda, Sylvia, Claire, Michelle, Andrew, Gerard, Brian, Mike and Brian.

I am grateful to the members of the MAI group at the Tyndall National Institute, for originally introducing me to the area of ubiquitous computing, and Dr. David O’Sullivan for providing me with the opportunity to join DERI.

I would like to thank my family and friends, mother and father, for providing encouragement, stress release and distractions. A particular thanks goes to my fiancée, GG, for her support, love and patience.
Publications

Parts of this work have appeared in the following publications:

**Peer-reviewed publications**


**National & European Union project deliverables**

# Contents

Abstract ................................................................. v
Acknowledgments ......................................................... vii
Publications ............................................................... ix
Contents ................................................................. xi
List of Figures .......................................................... xvii
List of Tables .......................................................... xix
List of Listings .......................................................... xxi

1 Introduction and Motivation ........................................ 1
  1.1 Introduction ....................................................... 1
  1.2 Problem Statement ............................................... 2
  1.3 Approach .......................................................... 4
  1.4 Hypothesis ......................................................... 5
  1.5 Contributions ...................................................... 6
  1.6 Evaluation ........................................................ 7
  1.7 Thesis Outline ..................................................... 8

2 Context Sharing Scenario ............................................ 11
  2.1 Fundamental Scenarios .......................................... 11
    2.1.1 Ubicomp Scenario Summary .................................. 11
    2.1.2 AmI Scenario Summary ....................................... 12
  2.2 Scenario Analysis ................................................ 13
    2.2.1 Actors and Environments .................................... 13
    2.2.2 Context Types ................................................. 13
    2.2.3 User Mobility and Context Sharing ......................... 14
    2.2.4 Context Privacy and Control ................................ 15
    2.2.5 Miscellaneous ................................................ 16
2.3 The Challenges ......................................................... 17
  2.3.1 Mobility and Discovery ......................................... 17
  2.3.2 Exchange of Context Between Third-Parties .................. 17
  2.3.3 Context Privacy Policies ....................................... 18
  2.3.4 Node Consistency ............................................... 18
  2.3.5 Deployment Restrictions ....................................... 18
  2.3.6 Additional Challenges ......................................... 18

2.4 A Motivational New Scenario ........................................ 19
  2.4.1 Scenario Details ................................................ 20
  2.4.2 Scenario Analysis .............................................. 20

2.5 Comparison Criteria Definition ..................................... 21

2.6 Summary .................................................................. 23

3 Background and Related Work ............................................ 25
  3.1 Ubiquitous Computing and Context-Awareness ..................... 25
    3.1.1 Definition of Context .......................................... 26
  3.2 The Historical Evolution of Context-Aware and Ubiquitous Computing ................................................................................................. 27
    3.2.1 The Formative Period ............................................ 28
    3.2.2 The Classical Period .......................................... 30
    3.2.3 Post-Classical Trends .......................................... 32
    3.2.4 Summary .......................................................... 34
  3.3 Controlled Sharing and Management of Context Data ............... 35
    3.3.1 Architecture Types .............................................. 35
    3.3.2 Core Related Work ............................................. 36
    3.3.3 Summary of Existing Approaches ........................... 47
  3.4 Analysis of Related Work ............................................ 50
    3.4.1 Individual Criteria ............................................. 50
    3.4.2 Combined Criteria ............................................. 52
  3.5 Summary .................................................................. 55

4 Context as a Service and the Network Model .......................... 57
  4.1 Requirements ................................................................ 57
    4.1.1 Adopted Approach .............................................. 57
    4.1.2 Core Requirements ............................................. 59
    4.1.3 Functional Requirements ...................................... 60
  4.2 Introduction to Context as a Service ................................. 61
    4.2.1 CxaaS Primary Components ................................... 61
    4.2.2 Node Level ....................................................... 63
    4.2.3 Network Level ................................................... 64
    4.2.4 Data Model ....................................................... 66
## Contents

6.3.9 Linking to Remote Context and Context Sources ................................................. 117
6.3.10 Groups .............................................................................................................. 117
6.4 URI Structure ........................................................................................................... 118
6.5 Granularity ................................................................................................................ 119
   6.5.1 Types of Granularity ......................................................................................... 119
   6.5.2 Modelling Granularity ...................................................................................... 121
   6.5.3 Location Granularity ......................................................................................... 122
   6.5.4 Event Granularity ............................................................................................ 123
   6.5.5 Activity Granularity ......................................................................................... 123
6.6 Service Descriptions .................................................................................................. 123
   6.6.1 Overview ........................................................................................................... 124
   6.6.2 Example ............................................................................................................ 124
6.7 Requirements Mapping ............................................................................................... 126
6.8 Summary .................................................................................................................... 126

7 Privacy Protection ......................................................................................................... 129
   7.1 Facets of Privacy ..................................................................................................... 129
      7.1.1 Requester ......................................................................................................... 130
      7.1.2 Context ............................................................................................................ 131
      7.1.3 Granularity ..................................................................................................... 131
   7.2 Policy Specification Approaches ............................................................................ 131
      7.2.1 Representation Options .................................................................................. 131
      7.2.2 Rules Languages ........................................................................................... 132
   7.3 Access Requests ...................................................................................................... 134
      7.3.1 Policy Structure ............................................................................................. 134
      7.3.2 Application of the Privacy Policy .................................................................. 135
      7.3.3 Reasoning Approach ...................................................................................... 137
   7.4 Privacy Policy Rules ............................................................................................... 137
      7.4.1 Constructing Policy Rules .............................................................................. 140
      7.4.2 Policy Execution ............................................................................................ 142
   7.5 Access Request and Privacy Policy Examples ....................................................... 142
      7.5.1 The Privacy Policy ......................................................................................... 143
      7.5.2 Access Requests ............................................................................................ 145
   7.6 Requirements Mapping ........................................................................................... 147
   7.7 Summary ................................................................................................................ 148

8 Evaluation ..................................................................................................................... 151
   8.1 Introduction ............................................................................................................. 151
   8.2 Requirements Analysis ........................................................................................... 153
      8.2.1 Scenario Specific Requirements .................................................................... 154
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>CxaaS Node Overview</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>Scenario Elements</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>A timeline of context-aware research and development</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>A timeline of commercial hardware and software related to context-awareness</td>
<td>28</td>
</tr>
<tr>
<td>3.3</td>
<td>The Xerox ParcTab</td>
<td>29</td>
</tr>
<tr>
<td>4.1</td>
<td>Overview of CxaaS Architecture</td>
<td>63</td>
</tr>
<tr>
<td>4.2</td>
<td>CxaaS-Node and its Local Ubicomp Environment</td>
<td>65</td>
</tr>
<tr>
<td>4.3</td>
<td>Sequence Diagram for Retrieving Remote Context Data</td>
<td>71</td>
</tr>
<tr>
<td>4.4</td>
<td>Sequence Diagram for Registering a User with a Lookup Service</td>
<td>72</td>
</tr>
<tr>
<td>4.5</td>
<td>User Entering Smart Space State Transition Diagram</td>
<td>76</td>
</tr>
<tr>
<td>4.6</td>
<td>CxaaS-Network Use Case Example</td>
<td>77</td>
</tr>
<tr>
<td>5.1</td>
<td>ILM and ECM Lifecycles</td>
<td>82</td>
</tr>
<tr>
<td>5.2</td>
<td>Hayden’s Data Lifecycle</td>
<td>84</td>
</tr>
<tr>
<td>5.3</td>
<td>Chantzara and Ferscha’s Lifecycles</td>
<td>86</td>
</tr>
<tr>
<td>5.4</td>
<td>The CxaaS Lifecycle</td>
<td>89</td>
</tr>
<tr>
<td>5.5</td>
<td>CxaaS Node Architecture</td>
<td>92</td>
</tr>
<tr>
<td>5.6</td>
<td>Sequence Diagram for a Context Creation Event on a CxaaS Node</td>
<td>93</td>
</tr>
<tr>
<td>5.7</td>
<td>Sequence Diagram for a Context Read Event on a CxaaS Node</td>
<td>95</td>
</tr>
<tr>
<td>5.8</td>
<td>Sequence Diagram for a Context Update Event on a CxaaS Node</td>
<td>97</td>
</tr>
<tr>
<td>5.9</td>
<td>Sequence Diagram for a Context Deletion Event on a CxaaS Node</td>
<td>98</td>
</tr>
<tr>
<td>5.10</td>
<td>Mapping Between CxaaS-Node Architecture and CxaaS Lifecycle</td>
<td>100</td>
</tr>
<tr>
<td>6.1</td>
<td>Key Classes of the Cxaas Ontology</td>
<td>112</td>
</tr>
<tr>
<td>6.2</td>
<td>Activities Classification</td>
<td>115</td>
</tr>
<tr>
<td>6.3</td>
<td>CxaaS Node Classification</td>
<td>116</td>
</tr>
<tr>
<td>6.4</td>
<td>LinkingPropertyGranularityScheme Example</td>
<td>120</td>
</tr>
<tr>
<td>6.5</td>
<td>CommonPropertyGranularityScheme Example</td>
<td>121</td>
</tr>
<tr>
<td>7.1</td>
<td>CxaaS Policy Ontology</td>
<td>134</td>
</tr>
</tbody>
</table>
### List of Figures

7.2 Privacy Policy Activity Diagram ........................................... 136
7.3 ‘Barack’ Requests Bill’s Location ........................................ 145
7.4 White House CxaaS Node Requests Bill’s Calendar Event .......... 146
7.5 White House CxaaS Node Requests Bill’s Location .................. 147

8.1 Evaluation Scenario Deployment .......................................... 159
8.2 CLS Resource Class Diagram ............................................. 161
8.3 Lookup Service Architecture .............................................. 163
8.4 ConServ Architecture ..................................................... 167
8.5 ConServ’s Resources Class Diagram .................................... 169
8.6 ConServ Request Processing State Diagram ......................... 173
8.7 Barista Helper Architecture .............................................. 174
8.8 UbiCafé Sequence Diagram ................................................ 175
8.9 EcoHome Assistant Architecture ....................................... 177
8.10 Exhibition Sequence Diagram ........................................... 179
8.11 SmartMeetings Architecture ............................................. 181
8.12 Meeting Sequence Diagram ............................................. 183
8.13 Sequence Diagram Showing Network Communications and Requirements ........................................... 185
8.14 FR.1.1: Interfaces Sends Redirect to the Requester ................ 186
8.15 FR.1.1: Interfaces Sends Detection Event to Requester ............ 186
8.16 FR.2.1: Identifier Information Returned to Requester .......... 188
8.17 FR.2.2: User Resource Returned by the Lookup Service ........... 189
8.18 FR.2.2: Service Resource Returned by the Lookup Service ....... 189
8.19 FR.2.3: Email-based Identifier Returned by CLS ................... 191

1 CxaaS Identifier Profile Screenshot ...................................... 231
2 CxaaS Service Profile Screenshot .......................................... 231
3 Details of user ‘gearoid’ on ConServ node http://conserv.deri.ie ...... 232
4 Listing of a privacy policy’s rules, listed according to precedence .... 232
5 UbiCafé’s Barista Helper Homepage ....................................... 233
6 UbiCafé’s Barista Helper Nearby Device List ............................ 233
7 EcoHome Assistant Exhibitor Information ............................... 234
8 PoFriend Screenshot ............................................................ 234
9 Walter searches for meeting attendee Donna using “Smart Meetings” 235
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Context data shared between environments</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Fire Eagle Location Granularity Levels</td>
<td>46</td>
</tr>
<tr>
<td>3.2</td>
<td>Comparison of Related Work</td>
<td>49</td>
</tr>
<tr>
<td>3.3</td>
<td>The Key Combined Criteria (CC)</td>
<td>52</td>
</tr>
<tr>
<td>3.4</td>
<td>Criteria C1.1 and C2.1</td>
<td>53</td>
</tr>
<tr>
<td>3.5</td>
<td>Criteria C3 and C5.1</td>
<td>54</td>
</tr>
<tr>
<td>3.6</td>
<td>Criteria C2.1 and C5.1</td>
<td>55</td>
</tr>
<tr>
<td>4.1</td>
<td>Core Requirements</td>
<td>59</td>
</tr>
<tr>
<td>4.2</td>
<td>Functional Requirements</td>
<td>62</td>
</tr>
<tr>
<td>4.3</td>
<td>Policy Types</td>
<td>66</td>
</tr>
<tr>
<td>4.4</td>
<td>Lookup Service Operations</td>
<td>73</td>
</tr>
<tr>
<td>4.5</td>
<td>CxaaS Node Core Operations</td>
<td>75</td>
</tr>
<tr>
<td>4.6</td>
<td>CxaaS Requirements Support</td>
<td>78</td>
</tr>
<tr>
<td>5.1</td>
<td>ILM Lifecycle Stage Descriptions</td>
<td>83</td>
</tr>
<tr>
<td>5.2</td>
<td>ECM Lifecycle Stage Descriptions</td>
<td>83</td>
</tr>
<tr>
<td>5.3</td>
<td>Hayden’s Lifecycle Stage Descriptions</td>
<td>85</td>
</tr>
<tr>
<td>5.4</td>
<td>Context Lifecycle Stage Descriptions</td>
<td>87</td>
</tr>
<tr>
<td>5.5</td>
<td>CxaaS-Node Requirements Support</td>
<td>103</td>
</tr>
<tr>
<td>6.1</td>
<td>Chen and Kotz Context Types</td>
<td>106</td>
</tr>
<tr>
<td>6.2</td>
<td>Bolchini’s Context Types</td>
<td>106</td>
</tr>
<tr>
<td>6.3</td>
<td>Bisgaard’s Context Types</td>
<td>107</td>
</tr>
<tr>
<td>6.4</td>
<td>Ontology namespaces and prefixes</td>
<td>111</td>
</tr>
<tr>
<td>6.5</td>
<td>Class URI Structure and Example</td>
<td>118</td>
</tr>
<tr>
<td>6.6</td>
<td>Property URI Structure and Example</td>
<td>118</td>
</tr>
<tr>
<td>6.7</td>
<td>Nested Property URI Structure Example</td>
<td>119</td>
</tr>
<tr>
<td>6.8</td>
<td>Location Granularity Levels</td>
<td>122</td>
</tr>
<tr>
<td>6.9</td>
<td>Event Granularity Levels</td>
<td>123</td>
</tr>
<tr>
<td>6.10</td>
<td>Activity Granularity Levels</td>
<td>124</td>
</tr>
</tbody>
</table>
List of Tables

6.11 CxaaS-Data Requirements Support ........................................... 127
7.1 Selected RIF Builtins ............................................................... 141
7.2 Completed CxaaS Requirements Support Table ............................... 148
8.1 Requirements Validation ............................................................ 156
8.2 CxaaS Lookup Service Interfaces ............................................... 165
8.3 CxaaS Lookup Service URL Structure ......................................... 166
8.4 ConServ Interfaces ................................................................. 171
8.5 Requirements Validated by Entire Scenario ................................... 196
8.6 CxaaS’s fulfilment of the comparison criteria .................................. 199
List of Listings

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Piece of Service Description Example</td>
<td>125</td>
</tr>
<tr>
<td>7.1</td>
<td>Basic structure of a rule</td>
<td>138</td>
</tr>
<tr>
<td>7.2</td>
<td>Basic rule structure including required first terms</td>
<td>138</td>
</tr>
<tr>
<td>7.3</td>
<td>Basic rule structure for a granted access request</td>
<td>139</td>
</tr>
<tr>
<td>7.4</td>
<td>Basic rule structure for a denied access request</td>
<td>139</td>
</tr>
<tr>
<td>7.5</td>
<td>Grant Service #1 access to detections at room level</td>
<td>140</td>
</tr>
<tr>
<td>7.6</td>
<td>Policy rule to deny all access requests</td>
<td>141</td>
</tr>
<tr>
<td>7.7</td>
<td>Example User Privacy Policy</td>
<td>143</td>
</tr>
<tr>
<td>8.1</td>
<td>FR.1.1 Context Requests</td>
<td>184</td>
</tr>
<tr>
<td>8.2</td>
<td>A Section of WSDL for ConServ</td>
<td>186</td>
</tr>
<tr>
<td>8.3</td>
<td>FR.2.1 Lookup Service Request</td>
<td>187</td>
</tr>
<tr>
<td>8.4</td>
<td>FR.2.2 Requests for the Discovery of Context Data</td>
<td>188</td>
</tr>
<tr>
<td>8.5</td>
<td>FR.2.3 Lookup Service Email-based Identifier Request</td>
<td>190</td>
</tr>
<tr>
<td>8.6</td>
<td>FR.4.3 Request Location Information From Exhibition</td>
<td>194</td>
</tr>
<tr>
<td>8.7</td>
<td>Donna’s Exhibition Calendar Access Rule</td>
<td>194</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction and Motivation

1.1 Introduction

The goal of ubiquitous computing and similar paradigms such as ambient intelligence (Shadbolt, 2003), pervasive computing (Satyanarayan, 2001) and sentient computing (Hopper, 2004), is to move beyond the desktop into the ‘third wave of computing’ where ‘the best computer is a quite invisible servant’ (Weiser and Brown, 1997). Mark Weiser coined the term and proposed the concept of ‘Ubiquitous Computing’ in 1988 while working at Xerox PARC.

Weiser envisaged computers receding into the background of our lives while maintaining their ability to interact with and aid people. Awareness of the user’s current situation, or what is now known as ‘context’, is central to the goal of computers extending the user’s unconscious (Weiser, 1993). By embedding computing resources of varying sizes and capabilities into the background of our regular physical environments, and combining them with knowledge of the user’s current situation, the environment can automatically interact with and help the user when required.

Due to its position of straddling the boundary between the physical and digital worlds the ubiquitous computing domain can leverage research from a wide range of domains such as wireless sensor networks, localization, smart phones, Augmented Reality and the Semantic Web to name a few. At the outset ubiquitous computing applications required the development of custom hardware (Want et al., 1996, 1992). However, as the capabilities of consumer electronics increased, and the smart phone in particular has become an increasingly powerful computing platform, the need for custom hardware has diminished.

The progression of consumer hardware contributed to an increase in ubiquitous computing research. Ubiquitous computing environments can now be created through the combination of consumer hardware, a ubiquitous computing framework and deployment specific software. The results are dynamic smart spaces (Wang et al., 2004c) that enhance a physical space, such as offices, homes, cars, shops and hospitals, with virtual services. However, these deployments are often stovepipes with few, ad-hoc connections between them.

Smart spaces are a partial step towards the realization of Weiser’s concept. However,
research focus must move beyond individual physical environments in order for comprehensive scenarios, such as outlined by Weiser \cite{Weiser1999} in the initial ubiquitous computing vision and by the European Commission’s IST Advisory Group (ISTAG) \cite{Ducatel2001} in the ambient intelligence vision, to become a reality. Such scenarios typically depict a person as they go about their daily lives, moving from one environment to another while being provided with personalised services appropriate for their given situation. The user’s context is constantly changing and with each change of environment a new smart space has access to their context data. Within the multi-environment scenarios it is clear that not all of the context data used by a particular smart space is created by that smart space, some of it is created outside of the smart space’s operational domain. Therefore, the sharing of context data is required between heterogeneous context-aware applications in order for such scenarios to be realised.

Many environments exist that would benefit from the addition of context-aware functionality but cannot provide it due to a dearth of context data originating from their domain of operation. For instance a person entering their local bookshop could be recommended *The Lost Symbol* by Dan Brown because they have a calendar entry stating that they saw *The Da Vinci Code* in the cinema the previous weekend. However, if the bookshop cannot access their customer’s context data then it is not possible for them to provide such functionality. From an implementation perspective many of the components, both hardware and software, required for such scenario functionality are available, the impediment is the access to the necessary remote context data. This ability to exchange context data is one of the problems this thesis addresses.

Due to its highly sensitive nature the ability to exchange context data alone is not sufficient, therefore, *the controlled sharing of context data between heterogeneous third-parties* is required. Controlled sharing with *third parties* is emphasised as the parties involved cannot be assumed to have any control or influence on each other. The ‘controlled sharing’ means that the subjects and owners of the context data must be able to specify who can access the context data and under what circumstances, rather than blindly transferring data to unknown third-parties, with unknown data management procedures and for unknown purposes. This is the second problem this thesis addresses.

### 1.2 Problem Statement

This thesis describes a model for the provision of Context as a Service (CxaaS). CxaaS aims to facilitate *the controlled sharing of context data between heterogeneous third-parties*. The CxaaS model takes a holistic view on the issue of controlled context sharing by considering the implications for individual Context Management Service (CxMS) nodes, the network of CxMS nodes and the users of those nodes. The model is made up of three primary parts: the node architecture, the network architecture and data model.

Current context-aware systems are often stovepipes, sharing little or no context data.
Consequently, context-aware applications are restricted to operating within the boundaries, both physical and virtual, where they can directly access context data concerning their users. The utility of such applications is limited as they are unable to respond to events concerning their users that occur in third-party domains as currently there is no suitable mechanism to handle the mobility of user context between spaces. Additionally, environments that could benefit from the inclusion of context-aware functionality but do not have access to sufficient context information have thus far been largely ignored by context-aware research.

The controlled sharing of context data between heterogeneous third-party applications introduces several challenges, in particular with respect to the privacy of context data, the mobility and discovery of users and the consistency of CxMSs. This thesis addresses the following research issues:

- Discovery of context data from remote sources concerning new and returning users;
- Enabling the privacy sensitive sharing of context data between heterogeneous third-parties;
- Provision of a formal context model appropriate for use with the discovery and privacy mechanisms.

The term ‘context data’ covers a broad spectrum of data-types as it includes any ‘information that characterises the situation of a person’ ([Abowd et al., 1999]) and contained within this information are some highly sensitive data-types that are not suitable for public access. Traditional ‘grant/deny’ access controls are not sufficiently expressive to represent the complexities of context access controls due to the dynamic nature of context data, and are cumbersome to use and prone to errors made by the users. Users may wish to allow access to context data during a particular period of time, while they are at a particular location, while they are attending a particular event or while their heart-rate is within a particular range. Additionally, the distributed and multi-party nature of sharing context data with third-parties adds extra complexity to privacy enforcement.

The ability to share context data in a controlled manner is of limited merit if new and returning users cannot be identified and new context data concerning them cannot be discovered. The implementation of the multi-environment scenarios envisaged by Weiser and ISTAG require such identification and discovery functionality in order to enable the seamless mobility of users between third-party ubicomp environments. Additionally, the discovery mechanism requires the flexibility to handle multiple user identification mechanisms, rather than requiring an individual identification type such as RFID.

Currently there is no standard method for the development of context-aware applications and services. While there have been several toolkits ([Salber et al., 1999]) and middlewares ([Dobson et al., 2007]) designed for the development of context-aware services they have not gained significant traction. Accordingly there are inconsistencies regarding how individual CxMSs handle context data which in turn may cause concern for users regarding the
security of their data. When users approve the sharing of their context data they should be aware of how the context data is handled by the CxaaS node, within its environment. This issue of node consistency is addressed as part of the CxaaS model.

Through the use of CxaaS, access to remote context sources is facilitated and consequently the creation of new context-aware applications is possible as applications have access to critical context data that would ordinarily be unavailable. Existing context-aware applications can be augmented and provide enhanced services to their users as they are no longer limited to context data found within their particular domain of operation. Additionally, application developers can focus on differentiation of their application rather than on ad-hoc integrations with third-parties in search of remote context data.

1.3 Approach

A generic model for the privacy sensitive exchange and of context data between heterogeneous third-parties is provided in this thesis. Context as a Service poses many new challenges due to its distributed nature and its position on the boundary of the physical world, captured by sensors, and the digital world of applications and services.

CxaaS focuses on the core functionality necessary to share context data in a controlled manner between third-parties. The CxaaS model consists of three primary parts: a node architecture, a network architecture and a node data model. The CxaaS Node Architecture defines the structure of software components necessary for an individual application to interact with the CxaaS Network. The CxaaS Network defines how individual nodes locate remote context data, exchange context data and secure the privacy of that data. The final core component of CxaaS is the Data Model, which enables the common understanding of data exchanged between third parties, and includes a formal model for the context exchanged between nodes.

CxaaS uses a combination of ontologies and rules to represent the required privacy policies. The ontologies are used to define a common model for the context data so that each CxMS has a shared understanding of the context data being exchanged. Privacy conditions based on the ontology’s instance data are defined using rules. The rules can be created based on the existing values of one or more pieces of context data. For instance a user can allow his doctor access to his location details only when his heart-rate is above 160bpm. Additionally, policies can specify access at a particular granularity level. For instance a user’s doctor can ordinarily find out what city he is located in but when his heart-rate is over 160bpm the doctor can get his exact latitude/longitude.

The CxaaS Network model defines how CxMSs communicate with each other, how users are identified and context data discovered about them. A lookup service is used to map between real-world user identifiers and URIs, which are used to uniquely identify users within the network. The lookup service forwards queries for a user’s context to the user’s dedicated ‘home node’ which tracks the distribution of a user’s context data throughout the network.
1.4 Hypothesis

Hypothesis: If a generic model appropriate for the controlled sharing of context data between heterogeneous third-parties can be provided then applications and services leveraging this work will be capable of providing enhanced, personalized functionality by adding or augmenting their context-awareness.
The adoption of the CxaaS model may provide numerous benefits to application developers and application users. Access to new sources of context data allows developers to provide enhanced services. Current context-aware services have access to additional context while existing services without context-aware offerings have the potential to add such functionality. Furthermore, the development of complex scenarios, such as those of Weiser and ISTAG, become feasible.

In order to support the proposed hypothesis, a prototype CxaaS implementation and deployment is described. The prototype is used to realize an updated version of the ubicomp and AmI visionary scenarios. Within the implemented scenario context data is shared, in a controlled manner, between third party environments and, based on this context data, the environments provide enhanced, context-aware, services to their users.

1.5 Contributions

This thesis presents the following contributions to the state of the art in the areas of ubiquitous computing and context-awareness:

The CxaaS Architecture is comprised of three primary parts: (i) the node architecture which is incorporated into (ii) the network architecture and (iii) the data model defines the communication between nodes. The node architecture defines a set of components that provide the functionality to manage the entire lifecycle of context data, enable communications with the CxaaS Network (and hence the exchange of context data) and enforce the privacy requirements of the various parties involved.

The CxaaS Network Architecture defines how context data concerning a particular subject is discovered from third-party nodes. In particular, the context data resulting from users moving between smart spaces is made accessible. The enforcement of privacy policies is a key component of the network architecture due to the highly sensitive nature of context data.

The development of an abstract architecture is not sufficient to ensure the interoperability of CxaaS Nodes and hence the exchange of context data between heterogeneous applications. The data model ensures a common understanding of the context data exchanged between CxaaS nodes. Central to the data model is a pragmatic context ontology which utilizes existing popular ontologies where possible in order to harness existing sources of context data and ensure a common understanding of the exchanged context data between services. The ontology adopts a bottom-up approach to defining the model for key context types commonly used in the development of context-aware applications. The node interface descriptions and network communication details are also included.

As part of this thesis an implementation of a CxaaS Node and the CxaaS Network is developed. The implementation is used as part of the evaluation of the research presented in this thesis. Furthermore, in order to demonstrate its practical applicability the implementation is used to realize an updated version of the scenarios of Weiser and ISTAG.
Additionally, an overview of the relevant context-aware research is provided. The progression from the inception of the ubiquitous computing paradigm to its current status is analysed and the key challenges within the domain are identified. Existing lifecycles for context data and lifecycles for enterprise data are analysed and used in the development of a lifecycle that is applicable to CxaaS. This CxaaS Lifecycle encapsulates the progression of the context data, from creation to deletion, within a six stage cycle. It focuses on a distributed environment incorporating the sharing of context data between heterogeneous applications in a privacy-sensitive manner.

Context as a Service contributes to the enablement of networked knowledge through the connection of heretofore unconnected smart spaces. This is achieved by annotating the context data within the smart spaces, using the CxaaS Ontology, in order to ensure a common understanding of the context data between third parties. Subsequently, this annotated context is made available to third parties, thus enabling them to link/network this context with their own context.

This thesis focuses on context data directly associated with users. While context data indirectly associated with users, such as the temperature of the room that a user is located in, is potentially of use to third-party ubicomp applications, the modelling, management, discovery and sharing of such data is outside the scope of this thesis.

1.6 Evaluation

The evaluation of the work presented in this thesis highlights the contributions outlined previously. It consists of three parts: (i) the development of prototypes of the key components of the proposed CxaaS model, (ii) the use of these prototypes to realise the updated scenario developed in Chapter 2 and (iii) the comparison of the proposed solution with the core related work.

The implementation incorporates each of the three parts of the CxaaS model: (i) CxaaS-Net, (ii) CxaaS-Node, and (iii) CxaaS-Data. As part of CxaaS-Net a lookup service is created and WSDL descriptions, annotated with SAWSDL, are provided for the RESTful interfaces exposed by the nodes. The CxaaS-Nodes provide functionality for the management of context data across its entire lifecycle and the enforcement of user, node and legal policies. All aspects of CxaaS-Data, including the ontology, privacy rules and interface descriptions are used within the implementation.

The new scenario incorporates the challenges, motivations, goals and expectations of the vision for the controlled sharing of context data between heterogeneous third-parties. Therefore, the applicability of the CxaaS model to this new set of goals, can be evaluated through the implementation of the scenario. The scenario, and hence its implementation, contains all of the fundamental functionalities, such as user mobility, user identification, context data discovery, context sharing, and privacy protection.

In the background and related work portion of this thesis, Chapter 3, a core set of related
works are analysed with respect to a set of criteria derived from two fundamental scenarios. Within the evaluation, these criteria are also used to evaluate Cxaas and compare it with the core set of related work.

1.7 Thesis Outline

The remainder of the thesis is structured as follows:

Chapter 2 discusses the seminal scenarios from the ubiquitous computing domain and identifies the key challenges contained within them. Based on these challenges a set of criteria are established which can be used to evaluate the related work. Additionally, a new scenario is developed that updates the outdated elements of the original scenarios while incorporating the identified challenges.

Chapter 3 provides the background and related work. This chapter introduces ubiquitous computing and context-awareness and outlines the progression of those paradigms from their inception to the current state-of-the-art. The key applications and frameworks in this area are identified and analysed and the issues currently faced by context-aware and ubiquitous applications is established. The key applications and frameworks are analysed according to the criteria established in Chapter 2.

The requirements for the provision of Cxaas are derived at the beginning of Chapter 4. These requirements are based upon the deficiencies of the related work when compared with the criteria. A set of core requirements and functional requirements are derived. Following this the proposed solution, Cxaas, is introduced and its primary components outlined. The remainder of the chapter concentrates on the network aspects, Cxaas-Net, of the Cxaas model. In particular the architectural requirements for the discovery and exchange of context data.

The node-level aspects of the model, known as Cxaas-Node, are dealt with in Chapter 5. As part of the Cxaas-Node definition a lifecycle for the management of context data is derived, which focuses on how context data is processed by the nodes as it goes through each of the steps from acquisition to disposition.

Chapter 6 formalizes the necessary implementation details to ensure the common understanding of data exchanged between Cxaas nodes. This consists of service descriptions, a formal model for context data and interface descriptions.

Chapter 7 discusses privacy protection which is crucial for user adoption of context-aware services. This chapter outlines the functionality provided within the Cxaas Model to protect the privacy of the involved parties. Three types of policies are defined: legal, node and user. The purpose of each type of policy, its implementation and enforcement details are described.

The Cxaas Model is evaluated in Chapter 8 through the creation of a Cxaas node implementation and the development of the components necessary for the realization of the scenario described in Section 2.4.
Chapter 9 summarises and analyses the contributions of this thesis and presents the future work to be undertaken based on the presented research.
Chapter 2

Context Sharing Scenario

Scenarios are useful tools for portraying how systems are used in daily life. They provide a non-technical narrative of interactions between users and systems while incorporating system goals, expectations, challenges and motivations. Scenarios are particularly suited to the ubicomp domain as it is a user oriented vision, focused on human-computer interaction.

The scenarios discussed in this chapter were constructed at the outset of the ubicomp and AmI visions and date from 1991 and 2001 respectively. They contain the original goals, challenges, motivations and expectations of the ubicomp and AmI visions in a concise, non-technical description. While several aspects of the scenarios have become outdated since their creation, the core motivations and challenges are still valid and a significant number of them are unsolved. Therefore, this chapter will identify those challenges and use them to create an updated scenario and construct a set of criteria that can be used to evaluate the related work in Chapter 3.

2.1 Fundamental Scenarios

This section summarises and analyses one fundamental scenario (Weiser, 1991; Ducatel et al., 2001) each from the areas of Ubiquitous Computing and Ambient Intelligence. The scenario analysis will primarily focus on the context sharing aspects of both scenarios.

2.1.1 Ubicomp Scenario Summary

The scenario described by Weiser (Weiser, 1991) depicts Sal as she goes about her workday. Some of the technological visions mentioned are now commercially available, such as the traffic-aware navigation system mentioned in the ubicomp scenario. However, despite its age, the core issues concerning the sharing of context data are still merely visions, as will be demonstrated by the analysis of the related work in Chapter 3.

The scenario begins with Sal waking in her smart home to the smell of coffee brewed

---

1 The scenarios can be seen in full in Section 9.3 of the Appendix
by her alarm clock which had detected her waking up. Her children had woken up earlier but were playing quietly because they were informed that their mother was still asleep. The smart home is equipped with a range of smart devices such as a smart pen for interacting with newspaper content and windows with image overlay functionality. One such window overlays anonymized electronic tracks left by neighbours, and detected by the local neighbourhood sensing devices, on a neighbourhood map. She commutes to work in a context-aware car which provides Sal with real-time traffic details, information on areas she is passing through and the location of free parking spots close to her office.

Sal’s office is equipped with ubicomp functionality such as indoor localization and smart devices such as interactive wall-mounted screens. Context data can be shared within the office and restricted to a specific period of time if necessary. In her office Sal has an activity indicator that informs her of spontaneous meetings, luckily on this day she does not have any. Sal is collaborating closely with Joe on a project and for its duration she shares a virtual office with him. This includes sharing context data such as location and sharing desktops with each other. While attending meetings Sal and other meeting attendees usually share their contact details with each other for the duration of the meeting. However, Sal is grateful that a client, Mary Hausdorf, has shared her details beyond the meeting as Sal does not have a record of them and needed to contact her.

2.1.2 AmI Scenario Summary

Scenarios for Ambient Intelligence in 2010 ([Ducatel et al., 2001]) was published in 2001 and defines four visionary scenarios for AmI. This section will concentrate on the first scenario entitled ‘MARIA: ROAD WARRIOR’ because the central actor in the scenario is extremely mobile, she is interacting with the smart environments for the first time and the scenario is sufficiently different from that of Weiser.

Maria arrives at a Far Eastern airport after a long flight. She has brought her ‘P-Com’, which is a single integrated communication device, with her on the trip. Her P-Com automatically identifies her to airport immigration and she can walk through without stopping. Her reserved smart car, equipped with a traffic guidance system to directs her to her hotel, is waiting for her. While in the car Maria’s daughter, Amanda, contacts her via the car’s telecommunication system. Maria had detected from the ‘En Casa’ smart home system that her mother is in a place that supports direct voice contact.

When Maria arrives at the hotel the porter is waiting for her, this is her first direct contact with a person since arriving at the airport. In the hotel her room automatically adjusts itself according to her preferences (temperature, lighting and music). At the time of her presentation her communication settings are adjusted to prohibit all but emergency contact. Maria had been wearing a cardio-monitor during the presentation and after the presentation has concluded it recommends that she takes a rest.
2.2 Scenario Analysis

While some of the technological features of the scenarios may appear dated the core components of the scenarios can be adopted and updated as the key challenge of enabling mobility between, and sharing context data with, the third-party environments is yet unaddressed. This section distils the scenarios to identify its key components, their interactions and associated challenges.

2.2.1 Actors and Environments

The scenarios involve multiple actors and multiple environments. Both of the scenarios center on single individuals, Sal and Maria, throughout the course of a particular day. The individuals interact, both physically and virtually, with other actors. Additionally, the primary actors in the scenarios use several different smart environments throughout the course of the day.

Sal and Maria interact directly and indirectly with other actors. Sal indirectly interacts with the people who have passed by her home as she can see the electronic tracks they have left and with her children she can see when they woke up. She directly interacts with her colleague Joe, with whom she is sharing a virtual office. Maria directly interacts with her daughter Amanda, while in the car and the hotel, the hotel porter and the presentation attendees. The AmI scenario does not have any explicit indirect interaction, however, Maria’s seamless progress through immigration and the delivery of the rental car most likely contain some form of indirect human interaction.

Both scenarios develop across multiple environments. The AmI scenario environments include, airport, car, the ‘En Casa’ enabled smart home and the hotel. The ubicomp scenario begins in Maria’s smart home, she then enters her smart car and finally arrives at her smart office. In both scenarios the users move seamlessly between environments, owned by third-parties, without reference to registration. While the ubicomp scenario depicts environments that Sal would interact with on a regular basis, within the AmI scenario it is stated that it is Maria’s first time interacting with the new AmI systems.

2.2.2 Context Types

Across both scenarios there are a range of context types used and there is a significant crossover of types between both scenarios. They are visionary scenarios and implementation specific details are not provided. However, we can infer the use of the following context types:

- Ubicomp Scenario:
  - Location: both indoor and outdoor location are used in the scenario. Indoor location tracking is used at the office and outdoor location tracking is evident in relation to the electronic tracks that are overlayed on the bedroom window.
Chapter 2. Context Sharing Scenario

- Preferences: when Sal wakes up in the morning she is immediately asked if she would like a coffee.

- Identification: the identification of Sal is a piece of context data and it occurs each time she enters an environment. It informs the environment that Sal is in a particular location at a particular time.

- Activity: activity recognition is evident when Sal’s children begin making more noise once they are informed that their mother is awake. The two activities in question are sleeping and awake.

- Schedule: Sal’s schedule is referenced twice in the scenario, initially in relation to spontaneous meetings and secondly regarding a past meeting with Mary Hausdorf.

- Contact Details: contact details were shared by Mary Hausdorf at a prior meeting with Sal.

- AmI Scenario:

  - Location: Maria’s location is used frequently in the scenario, such as advising the car rental company that Maria is in the airport or when immigration are informed of Maria’s location as she passes through the airport.

  - Preferences: when Maria enters her hotel room its settings are adjusted according to her preferences.

  - Identification: immigration required rigorous identification checks at the airport. Maria also identifies herself, via her PCom, to the smart car and the hotel.

  - Presence Status: Maria’s presence status was updated when she entered the rental car as the En Casa enabled home was informed that she was in a car supporting audio communication. Maria could also easily adjust her presence status as she did prior to giving her presentation.

  - Heart Rate: Maria wears a cardio monitor which records her heart rate and informs Maria of any warning signs.

2.2.3 User Mobility and Context Sharing

In the scenarios the actors move freely between environments and the handover from one environment to another is seamless. There is no reference to a need to register with the environments or use particular devices in certain places. This is especially evident in the AmI scenario where Maria is interacting with the environments for the first time. Maria begins in the airport, moves on to the car and then the hotel, each of which operates a context-aware service. The ubicomp scenario depicts Sal interacting with environments that she would encounter on a daily basis. Sal starts at home, then continues to her car and finally her place of work.
2.2. Scenario Analysis

Table 2.1 outlines the instances in the scenarios where context data is shared between environments. The AmI scenario contains more examples of context data sharing between third-parties than the ubicomp scenario as the ubicomp scenario is primarily focused on situations where context data is created and consumed within the same environment. However, one reoccurring sharing of context in Weiser’s scenario is when Sal identifies herself when she begins interacting with an environment. As Sal moves between environments she must be uniquely identifiable as she enters an environment in order to provide her with personalised and context aware services. It is unclear from the scenario what mechanisms are used for the identification, nevertheless the identification process is also fundamental to the seamless mobility within the scenarios.

<table>
<thead>
<tr>
<th>Context Type</th>
<th>Consumer</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ubicomp Scenario</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>Smart Home</td>
<td>Neighbourhood sensing devices</td>
<td>The neighbours electronic tracks are overlaid on the neighbourhood map</td>
</tr>
<tr>
<td>Contact Details</td>
<td>A device belonging to Sal</td>
<td>A device belonging to Mary Hausdorf</td>
<td>Mary provides her contact details to the other meeting attendees.</td>
</tr>
<tr>
<td>Identification</td>
<td>House, Car, Office</td>
<td>Unspecified</td>
<td>Sal identifies herself in order to interact with the smart house, car and office.</td>
</tr>
<tr>
<td><strong>AmI Scenario</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identification</td>
<td>Airport Immigration, Car, Hotel</td>
<td>PCom</td>
<td>Identification details provided to airport immigration, the smart car and the hotel.</td>
</tr>
<tr>
<td>Location</td>
<td>Rental Car Agency</td>
<td>Airport/PCom</td>
<td>Airport/PCom informs car rental agency that Maria is in the airport</td>
</tr>
<tr>
<td>Presence</td>
<td>En Casa Smart-Home</td>
<td>PCom</td>
<td>The PCom informs En Casa that Maria is located in a car supporting audio communication</td>
</tr>
<tr>
<td>Preferences</td>
<td>Hotel</td>
<td>PCom</td>
<td>Maria’s hotel room automatically adjusts according to her preferences when she enters.</td>
</tr>
</tbody>
</table>

Table 2.1: Context data shared between environments

2.2.4 Context Privacy and Control

Due to the absence of implementation details within the scenarios the proposed privacy protection strategies are ambiguous. For instance within the AmI scenario the PCom controls a substantial amount of Maria’s context data, unfortunately the procedures by which it
provides context data to third-parties is unclear. Within Weiser’s scenario there are two references to specific privacy protection mechanisms. Sal’s local neighbourhood monitoring service only shares anonymized tracks, rather than video or other privacy-sensitive types of data, of people moving within the neighbourhood. This indicates that there is a level of granularity or accuracy which the monitoring service can return data at depending on the permissions of the requester. The second example occurs during meeting whereby attendees can decide to only share contact details during the course of the meeting. This indicates that the privacy protection mechanisms are aware of the meeting duration and meeting attendees and, therefore, are context-aware.

Context control and context ownership are distinct but related topics. Context privacy concerns controlling access to context data while context control considers the ownership of the context data and what control the subjects of the context data have over it.

Within both scenarios it is unclear what happens to the context data shared with third-parties, captured in Table 2.1, after that data has been used for the initial purpose. It is not immediately obvious whether Sal and Maria retain any control over their context data once it has been shared. Issues such as the length of time the context is stored by the third-parties, the resharing of context data to other third-parties and the use of context data for purposes beyond the original intention are all related to context control.

2.2.5 Miscellaneous

Additional topics raised in the scenarios include the variety of middlewares deployed. Between the two scenarios seven environments are depicted that are sensing context data concerning the actors. It is unlikely that each of the environments will have deployed the same ubicomp middleware or framework, therefore, it may be unclear to users how the sensed context data will be handled internally by the environments. For instance, when Maria enters the rental car, how can she be confident that the context data stored by the car will only be used for purposes that she deems acceptable?

While the topic of context privacy has been discussed, the related topic of security has been not yet been mentioned. Context types can range from relatively static information, such as profile information, to highly dynamic data, such as location and activity information. Additionally, certain types of context data can be highly sensitive including location, heart-rate, blood pressure and activities and as a result the security of this data is of great importance. As a simple example, if Sal’s location information was to be misappropriated, it could be used to determine when she was in the office and her home could be burgled.
2.3 The Challenges

From a high-level perspective the core challenge outlined in the problem statement is the difficulty providing ‘the controlled sharing of context data between heterogeneous third-party ubicomp services’. This high-level problem can be decomposed into smaller, non-trivial, problems and while solutions may exist for the identified problems in isolation there is no complete solution for all of the problems. This section will identify those challenges by building upon the problem statement and the analysis of the scenarios conducted in the previous section.

There are three facets to the problems which are outlined below, the node issues, the network issues and the interoperability issues. The node level issues are focused on the responsibilities of individual nodes for the provision of ubicomp services in the local environment and the network issues concerns the responsibilities of a node in terms of communicating with other Context Management Services (CxMSs). The network issues centre on the obligations of the nodes as a collective in order to facilitate the controlled sharing of context data. The emphasis is on the external, network, perspective of interactions between nodes. This includes mechanisms such as the discovery of context data and the distributed enforcement of privacy policies. The interoperability issues concern the common understanding of the context data and privacy policies exchanged between nodes as well as the description of services offered by remote nodes.

2.3.1 Mobility and Discovery

User mobility is central to the two scenarios. In order for a user’s context data to seamlessly follow them, services must be capable of identifying users and discovering context about them as they enter a ubiquitous space. Additionally, limiting the ubicomp identification deployment options, for instance requiring users to carry specific hardware such as RFID tags or mobile phones running particular application, is not desirable. It may not be appropriate for users to carry a particular device type in all situations as ubicomp services may have more appropriate identification solution for their environment. These issues map directly to the issues discussed in Section 2.2.3.

2.3.2 Exchange of Context Between Third-Parties

The ability to exchange context data between heterogeneous third-parties concerns more than simply the availability of interfaces for sharing context data. Ubicomp services must be able to locate the interfaces of other ubicomp services, understand the function of the interfaces and how to interact with the interface. Additionally, the context data being exchanged must be understood by the interacting parties and, therefore, must adhere to a common data model. The Exchange of Context Between Third-Parties challenge primarily deals with network, rather than node, issues and it is related to Section 2.2.3 of the scenario analysis.
Chapter 2. Context Sharing Scenario

2.3.3 Context Privacy Policies

The privacy of user’s context data is crucial to the user adoption of ubicomp services. By providing mechanisms for sharing context data between third-parties additional privacy complexity is introduced beyond that of traditional single-environment ubicomp applications. As users move between ubicomp environments they create new context data, for instance the context data could be in the form of new location data, activity data or proximity data. The node, controlling the new context data, must enforce the appropriate privacy policies which could include node, legal and user policies.

The Context Privacy Policies issue deals with the topics discussed in Section 2.2.4. This challenge concerns the creation of a comprehensive privacy protection mechanism that provides suitable protection for users while also being capable of protecting the interests of the context owners and the necessary legal requirements. Enforcement of privacy policies across multiple nodes has both distribution and node level implications. From a network perspective the nodes much be capable of accessing remote user policies, or the appropriate parts of user policies, in order to ensure the protection of their context data. The enforcement of privacy policies is a node operation.

2.3.4 Node Consistency

As discussed in Section 2.2.5 CxMSs come in a range of forms and with a range of functionality. As the background and related work of Chapter 3 will show there is minimal standardisation in ubicomp application and service development and, as a result, there is no single middleware, framework or data model that is commonly used. This leads to issues concerning the consistency of how context data is handled from one CxMS to another. If the stages and processes that a person’s context data is subject to vary between environments then it may be difficult for users to trust an unfamiliar CxMS deployment. Therefore, there must be consistent handling of context data across the entire lifecycle of the context data.

2.3.5 Deployment Restrictions

Ubicomp deployments can vary greatly in terms of devices used, deployment environments and the types of applications interacting within the ubicomp environment. Additionally, the choice of implementation technologies varies from environment to environment as do the types of context data produced and consumed within the ubicomp environment. Therefore, the challenge is to provide mechanisms for the controlled sharing of context data with while introducing minimal deployment restrictions.

2.3.6 Additional Challenges

The topics of context security and device ownership are of increased importance in a distributed setting rather than in the classical deployments. With the larger number of
nodes, devices and users associated with the distributed setting there is also an increased threat of security breaches and device ownership related subterfuge. While these topics are of high relevance to this research topic the amount of work required to solve these issues is beyond the scope this thesis.

2.4 A Motivational New Scenario

To illustrate the benefits of the privacy-sensitive management and sharing of context data a business-oriented scenario is presented in this section. The scenario focuses on an individual user and highlights the deployment of context-aware functionality, incorporating a number of context management services (CxMS), across multiple physical environments. The capture, management and controlled dissemination of context data is demonstrated in a privacy-sensitive business setting.

The scenario depicts multiple elements interacting with each other through the production and consumption of context data, as illustrated in Figure 2.1. The scenario describes a business-oriented narrative and is a pertinent example of a scenario that current ubicomp systems would struggle to implement. It builds on the fundamental scenarios presented in Section 2.1 along with their analysis. The scenario will be implemented as part of the evaluation section of the thesis in order to determine the suitability of the proposed model.
Chapter 2. Context Sharing Scenario

in relation to the identified challenges and comparison criteria.

2.4.1 Scenario Details

Donna has recently purchased a site and plans to build a sustainable timber frame home and she is attending the EcoHome trade fair in order to source a frame manufacturer. Prior to entering the exhibition she decides to have a coffee in UbiCafé, a café close by. The owner of UbiCafé has deployed a range of context-aware hardware devices and software in the café to provide ubicomp services to their customers in order to increase efficiency and customer satisfaction and thus increase sales. Upon entering the café a waitress greets Donna by name and informs her that a window table is free upstairs, just like the one she sat at on her only other visit to the café. Once Donna is seated a waitress comes over and asks her if she would like a latte as before or if she might prefer an iced mocha latte like her friend Susan who was in yesterday.

At the exhibition, Donna is able to identify the position of the various timber frame manufacturers using the exhibition’s indoor location system. Walter is one of many manufacturers with whom she discusses her requirements. Once Donna and Walter begin talking the exhibition smart space automatically provides Walter with Donnas contact details, which she is sharing while she is in the exhibition center and EcoHome is ongoing. Also, as both Walter and Donna are sharing their calendar details with other exhibition attendees at a free/busy level a suitable time can automatically be suggested to Donna and Walter for a follow-on meeting.

The following day, prior to the scheduled meeting, it is clear from Donnas location that she will be late for the meeting as a meeting with another manufacturer went over time. Walter is automatically informed that Donna will be 5 minutes late and his calendar is adjusted accordingly.

Once the meeting commences Walter shows Donna a subset of his companies portfolio. The subset has been chosen based on Donna’s interests and buildings Donna frequently passes nearby. Donna shares details of her interests and a portion of her location history with the company for the duration of the meeting in order for this functionality to be possible.

2.4.2 Scenario Analysis

The new scenario is a distilled and updated version of the two older scenarios. Due to the focus of this thesis on the controlled sharing of context data much of the environmental specific details of the original scenarios are irrelevant and a distillation of the core concepts presented in the scenarios is required.

While it may have been possible to use one of the older scenarios within this thesis, the updated scenario highlights the current issues from ubicomp, while reducing the importance of the issues that have been addressed by existing research. The scenario builds upon the features and challenges identified in the analysis of the two seminal scenarios.
The new scenario shares many of the same key features as its predecessors. The new scenario depicts a user as they move between three different environments. Multiple actors are involved, but the primary focus is on a single actor, Donna. Multiple types of context data are used, such as location, contact details and friends, and each of those context types are also shared between CxMSs. The importance of privacy controls are also highlighted within the scenario, such as the sharing of contact details for the duration of the exhibition.

### 2.5 Comparison Criteria Definition

In addition to being used to define the previous scenario, this section uses the challenges identified in Sections 2.3 to define a set of criteria that will be used to compare the related work that is presented in the following chapter.

The ability to share context data alone is not sufficient. There are several other criteria which must be fulfilled in order for controlled sharing of context data between heterogeneous ubicomp applications. Provided below is a list of the key criteria that will be used in the following chapter to evaluate the related work. This section will elaborate on each of the criteria and describe how each of the criteria was derived.

**C1. Exchange context data with remote heterogeneous third-parties:** This criterion deals with the technical ability to exchange context data with remote third-party heterogeneous ubicomp services and consume that data. Ubicomp systems are designed and deployed with different functionality and requirements, hence it is unreasonable to assume that ubicomp developers will use a single middleware or framework. Accordingly, in order to gather context data from third-parties the ability to exchange context data with heterogeneous services is necessary. Within the new scenario the coffee shop’s ability to retrieve remote context data concerning Donna’s preferences relates to this criterion. In particular, the three key facets of this criterion are:

- **C1.1 Capable of exchanging context data:** Is the system capable of exchanging (providing and consuming) context data in a standardised fashion with standard-compliant heterogeneous third-party systems, without requiring a significant development overhead?

- **C1.2 Interfaces for the exchange of context data:** Are application interfaces available from which third-parties can request or be provided with data?

- **C1.3 Location independent context exchange:** Applications should be capable of exchanging context data regardless of their proximity to each other, for instance, applications should not have to be within Bluetooth range of each other in order to exchange data.

**C2. Privacy protection:** Simple access controls for static resources which simply allow or deny requesters access to resources regardless of the situation are not optimal for
Chapter 2. Context Sharing Scenario

the ubicomp domain. As context data is both sensitive and dynamic, the combination of these two factors requires privacy protection mechanisms where access levels can change in accordance with context. An example of context-aware privacy is present within the exhibition setting from the scenario when Donna shares her contact details for the duration of the exhibition and while she is in the exhibition center. The primary criteria for privacy protection are:

**C2.1 Users can control their own privacy:** The ability of users to specify the access levels they wish to allow according to particular contextual conditions. For instance, a user can specify that they wish to allow all work colleagues access to events in their calendar which occur between 9am and 5pm and only while the user is located in the office. Therefore, the ability to simply ‘grant’ or ‘deny’ an application’s requests for a particular context type is not sufficient.

**C2.2 Users can specify the context granularity level:** It should be possible to grant access to applicable context types at varying levels of granularity. Applicable context types, such as location and events, have an intuitive hierarchy or levels. The example provided above could be extended to allow a user’s work colleagues access to their calendar data at a ‘free/busy’ level for all events outside of working hours.

**C3. Capable of handling user discovery between ubicomp environments:** As users move between ubicomp environments their data should be capable of seamlessly following them. In order for this to be possible ubicomp services must be able to identify both new and returning users and discover context information about them. Within the scenario, when Donna enters both the coffee shop and the exhibition center she is automatically recognised and the environments have access to context data concerning her.

**C4. Ability to manage context data:** Environments must be capable of managing context data in order for it to be available for use by that application and for sharing it with other applications. Therefore, the abilities to share context data and manage context data are intertwined. While the scenario does not explicitly state where and how the context data is managed, each of the environments must have access to a context management service in order to provide context-aware functionality.

**C5 Context Types:** Context data is at the core of ubicomp applications. Therefore, the types of context available, how it is modelled, and the ability to add new types of context data are of importance to system design and operation. There are multiple types of context data mentioned within the scenario. In particular, location, preferences, contact details, friend information, calendar and interests. The following criteria take this into account:
C5.1 **Formal representation of context data:** A common representation or understanding of the context data is crucial as the ability to exchange context is of no use without the ability to understand it.

C5.2 **Support for multiple context types:** Support for location alone is not sufficient for developing ubicomp services. While location is the most common type of context used by ubicomp systems, additional types of context data are required to move from location-based services to fully context-aware services.

C5.3 **Support for extensible context types:** It is unreasonable to assume that all applications will require the exact same set of context types and granularity levels. Therefore, the ability to modify or extend the supported context types is necessary to maximise the number of potential applications.

C6. **Device Agnostic:** The mobile phone is increasingly becoming a hub for user content and context [Prehofer et al., 2009]. However, it is not the sole producer or consumer of context data. Therefore, it is important not to restrict the implementation to a particular device type. Additionally, the assumption that a user will have a particular device with a particular piece of software with them and operating at all times is not tenable. The scenario does not make explicit references to particular device types and hence there is no device-based restrictions for realisation of the scenario.

### 2.6 Summary

Scenarios are important tools that can be used to define the goals, expectations, challenges and motivations of a system. Particular importance is placed on scenarios within this thesis due to the the concise and effective manner they can convey a vision, in particular visions within the ubicomp domain. The focus of a scenario is on the user’s interactions with the system and this is of particular relevance to ubiquitous computing as ubicomp is a new human-computer interaction paradigm.

This chapter began by describing two fundamental scenarios. The first scenario was developed by Weiser and played a key role in defining his vision for ubiquitous computing. The second scenario originates from the overlapping research domain of ambient intelligence and was used as a vision for the future direction of AmI research.

The two identified scenarios contain many different types of issues that are not directly related to the subject matter of this thesis, therefore, some distillation of the scenarios was required. In order to do this the core elements of the scenarios were identified, along with the challenges within the scenarios related to the controlled sharing of context data. These challenges were then used to develop a new scenario and to create a set of criteria by which the related work can be evaluated. The new scenario will be used as a mechanism to evaluate the model developed within this thesis and the criteria will be used to compare it to the related work.
Chapter 3

Background and Related Work

This chapter provides an overview of the related work in the area. Ubiquitous Computing (ubicomp) and context-awareness are presented and discussed. Following on from this, the evolution of ubicomp and context-awareness is outlined and discussed and issues affecting the ubicomp domain are identified. The criteria established in the previous chapter are applied to the main approaches and these approaches are classified according those criteria.

3.1 Ubiquitous Computing and Context-Awareness

Weiser described Ubiquitous Computing (ubicomp) as the ‘third wave of computing’ where technology disappears into the background.

**Definition 1:** Ubiquitous computing enhances computer use by making many computers available throughout the physical environment, while making them effectively invisible to the user. [Weiser, 1993]

In addition to this definition, Weiser provided two quotes that provide an insight into his vision of ubiquitous computing:

‘The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.’

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

In order for technology to recede into the background, while still providing users with their desired services when required, the ubicomp applications must be aware of information
concerning their user’s current situation, known as context information. The term ‘context-awareness’ emerged from the Ubiquitous Computing paradigm in the 1990s (both concepts originated from the Xerox Pablo Alto Research Center (PARC) under the supervision of Mark Weiser) and describes applications or services which use context data to provide personalised services. Context information encompasses a wide variety of data such as a user’s location, free/busy status, heart-rate and friends nearby.

The original context-awareness vision inspired domains beyond the ubiquitous computing such as user interfaces, search engines, wearable computing and business process management while simultaneously flourishing in the ubicomp field under various guises, such as Pervasive Computing, Ambient Intelligence, Sentient Computing or Calm Technology.

3.1.1 Definition of Context

From the inception of the term context-awareness by Schilit et al. (Schilit et al., 1994) there has been disagreement about the definition of context. There are many definitions for context which vary depending on application functionality, user goals, deployment environment etc. Schilit et al. (Schilit et al., 1994) were the first to discuss ‘context-aware computing’ and they defined context as:

**Definition 2:** Location, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time.

Chen (Chen, 2004) takes the definition of context a step further by including environmental attributes:

**Definition 3:** Context is information about a location, its environmental attributes (e.g., noise level, light intensity, temperature, and motion) and the people, devices, objects and software agents that it contains. Context may also include system capabilities, services offered and sought, the activities and tasks in which people and computing entities are engaged, and their situational roles, beliefs, and intentions.

Dey’s definition (Dey, 2001) is more general than that of Chen:

**Definition 4:** Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

For the purpose of this thesis we shall adopt a restricted version of the definition of context provided by Dey. The primary focus of this thesis will be on personal context, such as a user’s free/busy status or body temperature, rather than environmental context, such as a room’s free/busy status or temperature. Personal context is viewed as a collection of information that characterizes a person and which is of interest to a significant number of ubicomp applications and which can be modelled, shared and understood by those applications. The sharing of
3.2. The Historical Evolution of Context-Aware and Ubiquitous Computing

Environmental context data between third-parties is not as critical as the sharing of user's context. The mobility of users and their interaction with multiple ubicomp environments is the primary factor behind this difference. Contrastingly, environmental context data is typically only of interest to the smart space operating within that environment. For instance, a smart office may know that Jim is in meeting room 3, where the temperature is 15 degrees and the projector is turned off. Out of the 3 pieces of contextual information (Jim’s location, temperature in meeting room 3, status of room 3 projector) only Jim’s location will be of interest to the majority of third-party ubicomp applications.

3.2 The Historical Evolution of Context-Aware and Ubiquitous Computing

The three distinct periods in the evolution of context-aware and ubicomp, as shown in Figure 3.1, are:

- The Formative Period, which includes the development of the ubicomp vision and the initial custom hardware-oriented deployments.
- The Classical Period, which involves software-oriented ubicomp deployments primarily using commercial hardware.
- The Post-Classical Period, which is a context-aware period, moving away from traditional ubicomp smart space deployments, centered on the synergy between the mobile phone and Web 2.0 applications.

Figure 3.1: A timeline of context-aware research and development

- The Formative Period, which includes the development of the ubicomp vision and the initial custom hardware-oriented deployments.
- The Classical Period, which involves software-oriented ubicomp deployments primarily using commercial hardware.
- The Post-Classical Period, which is a context-aware period, moving away from traditional ubicomp smart space deployments, centered on the synergy between the mobile phone and Web 2.0 applications.
Figure 3.1 shows the development in terms of major approaches and systems and Figure 3.2 shows the development in terms of hardware.

Figure 3.2: A timeline of commercial hardware and software related to context-awareness

This section focuses on general context-aware and ubicomp systems in order to provide a frame of reference for the research presented in this thesis and to identify some of the key problems currently faced by the domain. Ten key approaches closely related to the topic of this thesis are then identified and evaluated in detail in Section 3.3.

3.2.1 The Formative Period

At the end of the 1980s and beginning of the 1990’s the commercial hardware available was not compatible with Weiser’s vision of computers receding into the background. At that time mobile phones and laptops were large and cumbersome and PDAs were not available until 1993 when Apple launched the MessagePad [Kounalakis and Menuez 1993]. Accordingly, early ubicomp applications required the development of custom hardware to fulfil its requirements. Two of the leading centers for ubicomp research at that time were Xerox PARC, under the guidance of Weiser, and Olivetti Research Laboratory (ORL) Cambridge.

The Active Badge [Want et al. 1992], developed by ORL, was one of the first ubicomp applications. It consisted of a badge worn by a user which transmitted a unique five bit code every 15 seconds which was then captured by nearby networked sensors. The Active Badge System made this location information available via the “WWW Active Badge Service” and enabled telephone receptionists to route calls to the phone closest to the intended recipient.
3.2. The Historical Evolution of Context-Aware and Ubiquitous Computing

Throughout the 1990’s ORL continued to grow and introduce improved localization devices and software. The Active Bat (Jones et al., 2002) used ultrasound instead of infra-red, which resulted in much greater accuracy. Active Bat enabled the localization of a transmitter, “Bat”, to an accuracy of up to 3cm. The trend for localization systems continued with “The ORL Active Floor” (Addlessee et al., 1997) which identified a person walking on the floor based on their gait.

![Figure 3.3: The Xerox ParcTab](image)

The ParcTab (Want et al., 1996), which began development in 1992, was the seminal context-aware computing platform and was used as the foundation for much of Xerox PARC’s research in the area of Ubiquitous Computing. The ParcTab, shown in Figure 3.3 was a palm sized mobile computing device which could communicate wirelessly via infra-red. The device was designed to be extremely portable and have constant connectivity. The controlling software was designed to know the location of all ParcTabs at all times. The ParcTab was a significant improvement over the Active Badge as it allowed the user interaction via buttons or a pen and it included a collection of utilities for email, memo, sketch-pad, calendar and a remote drawing tool.

The ParcTab was complimented with other devices such as the MPad and LiveBoard (Weiser et al., 1999). The MPad was a multi-user, sketch-pad style, mobile X terminal. It was capable of performing multiple tasks but designed to be left in one location, if a user moved rooms they would not bring the MPad with them but simply use one in their next location. The LiveBoard was a large wall display whose primary function was as a collaborative drawing system. The ParcTabs could interact with the LiveBoards and enable remote drawing on the LiveBoards.

The trend of developing custom hardware for ubiquitous and context-aware applications was also evident elsewhere. The InfoPad (Truman et al., 2002) developed at UC Berkeley was also a wireless terminal and had additional multimedia capabilities which the MPad lacked.
3.2.2 The Classical Period

As the 1990’s progressed an increasing amount of consumer hardware which was suitable for context-aware and ubiquitous computing application development was becoming available and this hardware was becoming progressively cheaper. These developments caused an increase in context-aware ubiquitous computing research as access to suitable hardware was no longer limited to organisations which could design and manufacture their own.

One of the original ubicomp projects to begin using commercial hardware was Classroom 2000 [Abowd 2000] from Georgia Tech which aimed to ‘facilitate the easy capture of the traditional lecture experience’. Cyberguide [Long et al., 1996] was another context-aware service developed at Georgia Tech around the same time. Cyberguide’s aim was to replace human tour guides with an automated version running on a PDA. Users could locate places nearby with certain properties such as having special offers or offering free parking.

Involvement of Large Companies

This period saw several large multi-national companies enter the ubicomp area of research. Hewlett Packard developed Cooltown [Barton et al., 2001] which was an initiative to enable people, places and things to connect with each other via the Web. By providing a web presence, identified by a URL, to every entity they were able to communicate with each other. For instance a PDA could control the lights in a room by loading the URL associated with the room, or a projector could load a presentation by receiving the URL of the file.

Philips developed their version of ubicomp called Ambient Intelligence and produced several demonstrators based on this vision [Aarts 2003]. In 1999 Philips joined the Oxygen alliance, which was connected to the Oxygen project which will be discussed in Section 3.2.2. The Philips HomeLab was opened in 2002 and was designed to study how people interacted with ambient intelligent technology, such as smart mirrors with embedded displays, context-aware mood lighting and context-aware sound systems.

IBM developed a middleware infrastructure for context collection and dissemination called the Context Service [Lei et al., 2002]. They adopted a service-based approach resulting in a high level of abstraction for application developers. Clients, which may be users or programs, could either query for recent context data or they could register a request for notification upon matching of a certain condition. Access to context data was controlled by a controller which could be either the subject or the object of the context statement.

Development Frameworks and Middlewares

The difficulty and repetition involved in the development of ubicomp applications was also recognised during this time. Context Toolkit [Salber et al., 1999], CoBrA [Chen 2004], Construct [Dobson et al., 2007] and the Context Service [Lei et al., 2002] are examples of such initiatives aimed at helping the development of ubicomp and context-aware applications.
While the formative systems tended to focus on location awareness, classical frameworks such as the Context Toolkit moved context-aware research beyond location alone.

**Context Toolkit:** The Context Toolkit in particular played a significant role in the evolution of ubicomp and context-awareness as it was one of the original development frameworks and inspired many later frameworks. The toolkit provides a basic set of sensor units, or “widgets”, for application developers to use as part of their context-aware system. Context is stored using attribute-value tuples and is encoded as XML for inter-widget communication. Privacy is implemented through the use of context ownership, this means that only the owner of the context can access the data.

**Construct:** Construct is an open-source platform for developing ubicomp applications. It simplifies tasks such as sensor integration, data acquisition and network programming. Construct is a fully decentralised platform in which nodes can discover each other to form an overlay network. Sensors and applications can automatically discover and connect to Construct nodes. Semantic web technologies are used for data representation and data management. Both real and virtual sources of data are supported by the platform and all information sources are treated as sensors.

**SOCAM:** Service-Oriented Context-Aware Middleware (SOCAM) (Gu et al., 2005) is a middleware aimed at the rapid prototyping of context-aware mobile services in pervasive computing environments. An ontology is used to model the context data stored in the system and the resulting context data can be reasoned over. A top-level ontology is defined for modelling general context concepts and specific ontologies are used for individual environments such as office, home etc. The SOCAM architecture is made up of five components: Context Providers which abstract from heterogeneous context sources; Context Interpreters which provide context reasoning, querying and consistency functionality; a Context Database which stores the ontologies and the associated instance data; Context-aware Services which provide services to users and modify their operation according to the current context; and a Service Location Service which enables Context Providers and Context Interpreters to advertise their services and users or applications to locate them. The Context Interpreter is a central server that mediates the transfer of context data between context providers and clients.

**Large-scale Projects**

During this classical period several large-scale interdisciplinary ubicomp collaborations were launched. This section examines two main projects in detail, Project Oxygen and Gaia.

**Project Oxygen:** Developed as a collaboration between the MIT Laboratory for Computer Science, the MIT Artificial Intelligence Laboratory and several commercial partners,
Project Oxygen [Rudolph, 2001] strove to deliver “Human-Centered Computing” by combining certain user and system technologies. Project Oxygen was made up of many smaller projects from MIT within the ubiquitous computing and artificial intelligence area which were combined to create a pervasive, human-centric computing experience. Initially, Project Oxygen was to develop custom hardware to fulfill the required functionality, however, it was later deemed more pragmatic to use commercially available hardware.

Oxygen consisted of sensor-rich environments consisting of stationary devices such as cameras, projectors, microphones and various sensors and personal hand-held devices. The hand-held devices could accept speech and visual input and could act as mobile phones, beepers, radios, GPS devices, cameras or PDA’s. Access to information and services was provided by the Oxygen network which was capable of adapting to changing configurations of the devices. By supporting multiple communications protocols collaborative regions could dynamically be created at local, building, or campus levels.

Some of the smaller projects which combined to create Oxygen include the self-certifying file system (SFS) [Mazieres, 2000], a secure file-system which was used to provide global access to information within Oxygen; the Cricket Location System [Smith et al., 2004] which was a privacy sensitive location tracking system used for tracking users; the Intentional Naming System [Adjie-Winoto et al., 2000] which provided a mechanism for addressing objects by their use rather than by their location; and Haystacks [Ádar et al., 1999] which was used to find files.

Gaia: The Gaia Operating System [Roman and Campbell, 2000] used the paradigm of an operating system and applied it to interactive physical spaces, known as ‘active spaces’. Much like a traditional operating system has multiple inputs and outputs, Gaia had active spaces (e.g. wall-mounted displays, phones, speakers and monitors). Active spaces also provided program execution, file-system access, error detection and resource allocation.

The Presentation Manager demonstrator [Roman et al., 2002] illustrated Gaia’s capabilities. It provided slideshow presentation functionality within the active space and allowed the displaying of the slides on multiple displays within the space. The slideshow could be controlled from multiple input devices within the active space. The presentation could be followed remotely via a web-cam which tracked the speaker as he/she moved around the room. If the speaker left the room he/she could be viewed remotely via the closest camera to his/her current location.

3.2.3 Post-Classical Trends

Recently there have been several efforts to provide web-based context management with the focus primarily being on location data. These initiatives include Fire Eagle [fir], Google Latitude [goo], IYOUIT [iyo] and the OSLO Alliance (Open Sharing of Location Objects) [Siegler, 2009]. These services receive updates from applications running on smart
phones, manual user inputs or by using the JavaScript geolocation API within the user’s browser.

The popularity of smart phones, such as the Apple iPhone and Android-based phones, has provided web-based applications with access to new data types and data sources. The combination of high levels of handset connectivity, new types of sensors embedded into the handsets and inexpensive applications has resulted in increased application access to context data. By developing clients for these smart-phones web-applications can access data such as a user’s location, orientation, acceleration and ambient light/sound surrounding the user. Additional external sensors can be used with the smartphones to provide other types of context data such as heart rate (spu), speed and cadence while running (nik), or power and cadence while cycling (smh).

The post-classical period is characterized by the combination of Web 2.0 services with smart phone applications for the provision of context-aware services. This new phase of context-aware applications has moved away from the ubicomp domain. These applications have not “disappeared” nor are they “indistinguishable from the fabric of everyday life”, however, they do react to the user’s situation and modify the provided service based on the user’s context.

At the heart of this new category of applications and are a set of services which act as brokers for user’s context data, in particular their location data. The context brokers take this data and make it available to other applications and other users. Three brokers in particular are of interest, Fire Eagle due to its early entry into the field, Google Latitude because of its potentially large user base and IYOUIT as a result of its focus on other types of context data rather than just location.

**Location Brokers:** Fire Eagle is a location broker which enables users to centrally store their location and to share it with applications or services that can then offer geo-aware services. It is primarily focused on geo-enabling third-party applications rather than sharing location between individual users. Privacy controls and different levels of access granularity are provided. Google Latitude is another example of a location broker which, due to its inclusion in Google’s Mobile Maps application, has a large potential user-base. Unlike Fire Eagle it is more focused on sharing location data between users rather than with third-party applications. These two services will be discussed in greater detail later in this chapter.

**Location-aware Recommendations:** There are a growing number of applications in the area of location-aware recommendations, such as the tagging, reviewing and sharing of places. These services typically combine a web-service with a mobile phone application to tag and share location-based events or locations with others. Services such as Gowalla (gow), Loopt (loo), Whrrl (whr), Locle (loc), Hotlist (the), Brightkite (bri), Rummble (rum) and Foursquare (fou) are a small selection of the offerings currently available in this domain. The functionality of Foursquare and Rummble are summarised to provide examples of the services
Foursquare is a location-based social networking website which provides friend-finding and a social city-guide functionality. Applications are provided for the major mobile phone platforms which allows users to update their location or leave location-based hints and tips to others. It is also possible to interact with the service via SMS and the website. Foursquare includes some game functionality to encourage users to use the service by rewarding them with ‘Badges’ when they frequently use the service or providing titles such as ‘Mayor’ when they provide the most updates from a particular location.

Rummble is a location-aware discovery service and social search platform that allows users to easily find content in their current location. Rummble allows users to tag anything with a location and provide a rating for it, this rating is then used to calculate a trust metric between the user and others who have tagged the same thing. From this trust metric people with similar interests and opinions are automatically identified and recommendations can be provided to the user from those identified people.

Mobile Phone Context Harvesting  CenceMe (Miluzzo et al., 2007) and IYOUIT are two examples of services that harvest as much context data as possible from mobile phones and make that available via social networks. CenceMe gathers noise, location, accelerometer and activity data from iPhones or iPod Touches. This data is made available to others via popular social networking sites such as Facebook, MySpace and Twitter or via the CenceMe web application. IYOUIT, which will be described in greater detail later in this chapter, collects a similar level of data from Symbian-based phones and makes it available via the IYOUIT website.

3.2.4 Summary

At the outset of ubiquitous computing, the amount of context available to applications, and the number of context data sources used by the applications, were limited. Context data was gathered using custom hardware and the ubicomp services based on that context used bespoke software. Gradually, ubicomp applications were able to leverage the increased capabilities of mobile consumer hardware, and as a result service development became primarily a software task. During this period ubicomp research was expanding and several large companies began experiments and large inter-disciplinary projects were started.

As the computing power, connectivity and range of sensors available on mobile phones increased, some of the functionality previously provided by ubicomp applications moved to mobile phone applications and their associated web applications. Functions such as friend finding, sleep monitoring and location-based reminders could be easily deployed on smart phones. This new class of context-aware application is not limited by physical operational boundaries in the same way as ubicomp smart spaces are. While smart spaces may be limited to gleaning context data from within their operational building or room, mobile phones can
provide context data from anywhere worldwide with suitable connectivity.

However, this new wave of context-aware applications cannot replicate all the functionality provided by ubicomp services. Ubicomp services are generally associated with a particular physical space in which they have “disappeared”. These environments can contain a greater range of sensors and actuators than a mobile phone and, therefore, can collect and respond to a wider range of context types.

Ubicomp environments can also harness the data coming from mobile phones and location brokers to expand their operational domain. While this limited sharing of context data between the ubicomp applications and the mobile phones and brokers is of benefit to the operation of ubicomp applications, the sharing of the wider selection of context data directly between ubicomp applications would be of greater benefit.

### 3.3 Controlled Sharing and Management of Context Data

To date the majority of ubicomp research has focused on enhancing individual environments with virtual services. Homes, offices, cars, libraries and hospitals, among others, have been augmented with context-aware applications. However, the sharing of context data between heterogeneous applications has received little attention thus far and as a result ubicomp services are restricted to using data available directly within their domain of operation. For instance, a smart office may be able to postpone a meeting if an attendee located in the office is delayed by a phone-call. However, the smart office may be unaware that an attendee located in their smart-car is caught in a traffic jam as currently there is no standard way for sharing context data between heterogeneous third-party applications.

Having access to new sources of context data allows application developers to provide new context-aware functionality. Services can react to actions and events which occur in separate locations, for example, if there is no milk in the fridge a smart home can suggest that a resident buys some milk when they are driving home and they are in close proximity to a supermarket. Additionally, complex scenarios, such as those described by Weiser and ISTAG in their visions for Ubiquitous Computing and Ambient Intelligence, can be realized thanks to the flow of context data between services.

#### 3.3.1 Architecture Types

The related work, which will be discussed in the following sections, is selected based on the existing research which best fulfils the criteria identified in Section 2.5. There are five classes of architecture within the identified systems:

**Single Environments:** Such environments are typical of the ‘Classical Period’ of ubiquitous computing applications. Single environment based systems do not share context data with other ubicomp systems and accordingly do not have any mobility functionality. However,
the remaining criteria are applicable to such systems. While single environment deployments
often use bespoke software to provide context-aware functionality, such systems are excluded
from selection as they are not appropriate for reuse by others.

Distributed Deployments: Distributed deployments operate in physically disconnected
environments, either within the same organisation or across organisational boundaries. Context
sharing capabilities are integral part of the functionality of this category of ubicomp
system and, unlike the single environment category, the entire set of criteria is applicable
when analysing these systems.

Software Bridge: Software bridges enable the sharing of context data between heteroge-
neous Context Management Services (CxMSs) through the integration of two CxMSs at both
an interface and context model level. Once bridged the two CxMSs should function as one
system and, therefore, can access context information from each other.

Mobile Phone Based: Mobile phones are a rich source of context data from which infor-
mation such as a user’s location, nearby devices, gait, people who are frequently contacted
and calendar can be gathered. As a result several research efforts have focused on turning
the mobile phone into a personal hub for a user’s context data. Interfaces can be provided
for ubicomp services to interact with the phone in order to retrieve context data concerning
the user.

Web-based Brokers: Users have ever increasing web-connectivity options, mobile broad-
band allows Internet on the move and it is available across a range of devices, from laptops
to mobile phones to Internet tablets. With this increased connectivity the use of web-based
technologies to store and broker context data is a natural fit. Web technologies are well
supported, cheap to deploy and highly accessible; therefore, the management of context data
on the web provides application developers with an easily usable source of context data and
users are provided with a central point at which to manage their context data. The web-based
broker approaches typically do not originate from the ubicomp domain, rather they emerge
from a combination of Web 2.0 and mobile phone based development. This architecture type
is typical of the ‘Post-Classical Period’ presented in Section 3.2.3

3.3.2 Core Related Work

The ubicomp research domain contains a vast amount of context-aware systems, frame-
works and middlewares. In this section ten examples of the most relevant research approaches
will be described and analysed. The systems have been chosen based on their support for the
criteria established in the previous section. While the sharing of context data with third-party
context-aware applications is not explicitly supported by all of the systems, their support for
the other criteria identified is of relevance. The systems are presented according to their architectural categories.

**Single Environments**

**CoBrA:** Context Broker Architecture (CoBrA) ([Chen et al., 2004a](#)) is an agent-based architecture for supporting context-aware systems in smart spaces. At the core of CoBrA is a context broker that manages context data for all of the CoBrA agents operating within the same smart space. The broker is made up of four primary components: (i) **Context Knowledge Base** manages the context data and provides interfaces for querying, adding and modifying context data; (ii) **Context Reasoning Engine** creates new information by reasoning over recently acquired context data; (iii) **Context Acquisition Module** is a set of procedures for acquiring context data from external sources such as sensors or web services; and (iv) **Privacy Management Module** defines the set of rules which the broker follows when performing privacy management tasks. The agents which interact with the context broker can come in many forms, from applications running on mobile phones to intelligent services operating within a particular environment.

The CoBrA architecture is focused on deployments in single environments; however, the brokers are not completely isolated and can consume data from third-party web-services and replicate their context data across teams of brokers. In order to remove the single-point-of-failure problem associated with having only one broker, it is possible to deploy a team of brokers for the purpose of replicating context data across multiple instances ([Chen et al., 2003](#)). The broker can process context data retrieved from third-parties, such as a weather data web-services, however, these are tightly-coupled to the web services and unsuited to consuming context from third-parties.

CoBrA uses a set of ontologies ([Chen et al., 2004a](#)), which are based on the higher level SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) ontologies ([Chen et al., 2005](#)), to maintain the context model. The use of ontologies in CoBrA provides it with the same flexibility as an object-oriented context model, but with the additional benefits of a high level of formality, thus enabling easier interoperation with looser coupling. The ontologies model many different types of context data, such as user profiles, devices, geospatial aspects, events and meetings, much of which is optimised for a smart meeting and office-oriented scenario. Due to the use of OWL for the formal representation of the context ontologies they can easily be extended in order to include any additional context types which may be required.

CoBrA agents and context brokers communicate using a combination of the Foundation for Intelligent Physical Agents - Agent Communication Language (FIPA-ACL) and the SOUPA and CoBrA context ontologies. The agents primarily operate within a single network, however, individual agents can be configured to communicate with remote brokers, such as having a context provider or context consumer at the user’s home while the context broker is...
located in the user’s office. An external interface is not provided for remote communication with the broker, rather developers must create a proxy for each remote agent which requires broker interaction. Additionally, CoBrA is not capable of handling discovery of context data between ubicomp environments.

From a privacy protection perspective the CoBrA policies are comprehensive and allow users to specify access conditions based on their current context. The privacy policies also enable users to specify the level of granularity at which to allow the requester access to their location data.

Distributed Environments

Context Fabric: Context Fabric (ConFab) ([Hong and Landay 2004]), is a context-aware development framework. It consists of three parts: a context data store, a context specification language and privacy protection mechanisms. The data store manages how and where the context is stored and distributes the administration, maintenance and protection of the context data. The context data model consists of three parts: entities which can be people, places and things, attributes which are properties associated with an entity, such as a name, and aggregates which is a mechanism for grouping entities.

The Context Specification Language (CSL) of ConFab is a mechanism for stating the context requirements at a high level. CSL statements allow for the specification of requests such as ‘What are the five nearest movie theaters to me?’. Additionally, event notification requests can be specified by CSL, such as ‘Notify me every time a person enters the room’.

ConFab places heavy emphasis upon the privacy protection aspect of the architecture. The identified challenge is how to provide a balance between the requirements of the user and those of governments and societies. Three operational modes are provided: optimistic where the application shares information by default and then later changes based on abuse detection, pessimistic where the emphasis is placed on prevention of abuse, and finally mixed-initiative where the decisions to share or not share information are made interactively by the user. ConFab allows for features such as restricting the execution of certain queries to when the user is physically located nearby. It also allows for intentionally ambiguous answers to be returned, which is the ConFab equivalent to privacy policy granularity.

Two application scenarios have been identified which highlight ConFab’s benefits. The first scenario is a friend finder scenario, whereby employees can update their location in a company’s location server at different levels of granularity. Employees are then notified of any queries against their data and their location details are only returned if they are located within the same office building. The second scenario is an automatic tour guide system which provides different data depending on the granularity of location data shared with it. For instance if the user’s current city location is provided then queue lengths for famous city museums are provided, at neighbourhood level, local points of interest are suggested and at street level, real-time navigation maps and a route finder can be provided.
3.3. Controlled Sharing and Management of Context Data

ConFab is heavily based on web technologies, such as HTTP for network communication, the Tomcat web server and the XPath query language. Operations on the InfoSpace, which stores context data, are based on a modified version of the RESTful paradigm. The basic HTTP operations (PUT, GET, POST, DELETE) are extended to include ‘Query’, ‘Subscribe’, ‘Unsubscribe’ and ‘Notify’ functionality. Service descriptions are incorporated into the framework, which allows applications to describe their functionality, their information requirements and the options which the end-user can choose from.

Unfortunately, due to ConFab’s modification of HTTP methods, it is difficult for third-party applications to interact with ConFab InfoSpaces due to the non-standard approach adopted. Therefore, while interfaces are provided for interaction, they are not usable by third-parties without significant ConFab specific modifications.

ConFab is capable of managing context data through the use of InfoSpaces and also has mechanisms for the discovery of context data. The privacy policies provided by ConFab are comprehensive and during the development of the Context Fabric privacy policy framework, the legal requirements were analysed and parts of these requirements were integrated into the framework. Regrettably, ConFab lack of semantics and ontologies for modelling its context data means it cannot be guaranteed that interacting services will understand the context data. The context data itself is represented in XML and the context types are extensible.

The Nexus Platform: The Nexus Platform focuses on how context-aware applications can be supported by the federation of local context models (Lehmann et al., 2004) and builds upon the research from the Aware Home Project (Kidd et al., 1999). While the Aware Home Project is focused on the individual home environment, the Nexus Platform adopts a global perspective and attempts to provide applications with uniform context information from different administrative and application domains.

A three layer architecture is used with applications at the top layer, the middle layer controls the federation and the bottom layer is made up of context providers. When Nexus receives a request for context from an application, it initially determines the spatial area of the request, it then forwards the request to the appropriate context providers. Nexus integrates the received results and returns them to the requesting application.

A custom data model is used within Nexus, the Augmented World Model (AWM) (Nicklas and Mitschang, 2004), which is made up of two parts, the Standard Class Schema (SCS) and the Extended Class Schemes (ECS). The SCS contains a set of types which the Nexus developers consider relevant to context-aware applications, and ECS provides a mechanism for application developers to include specialised types which are not included in SCS. A top-down approach in the creation of the AWM data model is used. The result is a heavy-weight ontology in which the SCS alone contains 250 classes (Lehmann et al., 2004). Such heavy-weight ontologies are difficult to use and often find it difficult to gain traction in comparison with lighter, bottom-up, approaches. AWM is developed using a custom XML-based language called Augmented World Modelling Language (AWML). However, AWML is not a formal or
standardised representation for context data and, therefore, cannot be guaranteed to be understood by third-parties.

The Nexus Extended Architecture \cite{Lange2009} is a proposed extension to the Nexus Platform outlined above. It adopts the same three layer structure, however, multiple layers can be created and distributed across different locations. This allows applications to get data from context producers controlled by third-parties. The data from the context producers is made accessible to the applications via the distributed layer of federation nodes.

The Nexus Platform and Extended Architecture recognises the need for sharing of context data between applications, however, the interfaces which are provided for the acquisition of context data are custom interfaces and do not adopt commonly used standards, e.g., the custom Augmented World Query Language (AWQL) is used by Nexus nodes to query the AWQL interface for spatial data.

Discovery of context data is possible through the use of the Nexus Distributed Query Processing Service. This service operates across the entire distributed Nexus network and is located within the platform’s Federation Layer. Additionally, Nexus is not constrained to a particular platform and location independent context exchange is possible between nodes.

The privacy side of context sharing has been ignored by Nexus. The concept of offering a “global, consistent view” on an application’s context data without offering privacy protection does not reflect the realities of how third-party applications interact. Application A may wish to share all of their context data with Application B, however, A may only wish to share context-data with Application C during working hours and they do not ever wish to share context-data with Application D. Additionally, the user’s role in the securing of context data concerning them is not dealt with nor are the legal implications of enabling such unrestricted data sharing.

Software Bridge

Context Bridge: Context Bridge \cite{Hesselman2007} is an architecture for facilitating the interoperation of pairs of context management systems (CxMS). A software bridge is used to make the functionality provided by a foreign CxMS available to the native CxMS. The primary focus of Context Bridge is the discovery and exchange of context data between heterogeneous CxMSs. Each Context Bridges can include six categories of functionality, which are:

- Identity Mapping: map between the scheme used for identifying users in one CxMS to the scheme of the other CxMS
- Context Discovery: the ability to discover context producers under the control of foreign CxMSs
- Context Forwarding: the technical ability to exchange context data from one CxMS type to another in terms of the protocols used by each CxMS
• Context Format Mapping: the ability to map between the context representation formats used by each of the CxMSs

• Context Adaptation & Reasoning: the ability to adapt the context arriving from the foreign CxMS to fit the native CxMS’s requirements, e.g., data rate, context type adjustments and context aggregation

• Privacy Controls: enforce the privacy controls required by a CxMS when interacting with a foreign CxMS

In the creation of Context Bridge, Hesselman et al. have highlighted one of the major issues with ubicomp systems: they recognised that the lack of interoperation between CxMSs is restricting the potential of ubicomp services, particularly regarding user mobility. However, the approach they adopted to solve this problem is cumbersome and inefficient. Their solution is suited to individual pairs of CxMSs, however, the development overhead required to integrate multiple CxMSs from multiple environments, using multiple vocabularies and communication protocols is significant. With each new CxMS or CxMS variant added the number of bridges required grows quadratically; therefore, this approach is not scalable and is an example of a classical $N^2$ data integration problem. For each pair of CxMSs two separate bridges are required as communication within a bridge is uni-directional; therefore, for bi-directional communication two separate bridges are required.

Each bridge is a unique piece of software containing a significant amount of bespoke code for that particular CxMS pair. Details such as the interfaces provided, privacy protection mechanisms, context representation and the types of context managed are different from bridge to bridge. Therefore, it is not possible to exchange context data between a particular pair of previously unbridged CxMSs without a significant development overhead. Additionally, Context Bridge does not expose any interfaces by default and each interface within a deployed bridge is a custom implementation.

The bridges do not store the context data themselves. This is the responsibility of the bridged CxMSs. Discovery of context data across the CxMSs is an integral part of the bridging functionality as is the protection of user privacy. However, specific mechanisms for allowing, and mediating between, varying levels of granularity or accuracy are not available.

The internal representation used for context data is unknown and the external representation is dependent on the bridged CxMSs; therefore, the formal representation of context data (Criterion 5.1) is not applicable to Context Bridge. Context Bridge is capable of bridging multiple types of context data and is not restricted to a particular set of context types.

Ubicomp Integration Framework: Blackstock et al. (Blackstock et al., 2006) have identified two impediments to the wider deployment of ubicomp applications: (i) system designers have primarily focused on applications within one administration or domain and do not integrate with applications from other domains, and (ii) there is no shared model for
Chapter 3. Background and Related Work

ubicomp environments and, therefore, there is no shared understanding between heterogeneous applications. The approach adopted by Blackstock et. al. is to create bridges between existing ubicomp systems and a common model for inter-application interaction. The model in question is the Ubicomp Common Model (UCM) which has been implemented in the Ubicomp Integration Framework (UIF) [Blackstock et al., 2008].

UCM is an extensible model which supports multiple-types of context data, but is not a complete representation of all context types, rather it is a common core which is intended to be specialised for a specific environment. It is described using OWL, combined with a set of Jena rules, and is made up of seven abstractions:

- Environment: encapsulates the available components, context types, services types, and entities in a given environment
- Entities: abstractions representing people, places, computing devices, groups and activities
- Entity relationships: Relationships between individual entities, such as location, social and activity relationships
- Context: context data concerning an entity, such as location, temperature and direction
- Services: sensor, actuator or software services provided by a device
- Events: occurrences which result in a change in state for an entity, for example a change of location of a person or a door closing
- Content: data related to an entity, such as a person’s photo, documents etc.

The UIF is a meta-middleware implementation of UCM. It bridges existing ubicomp platforms from their internal model to the UCM and it can be used as a standalone middleware for ubicomp environments. It serves three primary functions: (i) It is a repository for context data concerning the operational environment, (ii) it routes application requests to the appropriate ubicomp adapters or internal components, and (iii) it manages event subscriptions.

UIF exposes SOAP-based interfaces which are described using concepts from UCM. Exchange of context data with third-parties is possible with UIF thanks to its support of commonly supported standards-based interfaces and the use of formal OWL-based descriptions using UCM. Context data exchange does not have any location-based restrictions and, therefore, can occur in a location independent manner.

Unlike Context Bridge, UIF provides both bridging functionality and internal context management functionality. The discovery of context data is supported by UIF across bridged CxMSs, as once bridged the systems operate as though they were one.

UIF, and the associated model UCM ignore the issue of context privacy and as a result are not is not suited to the sharing of context data between third-parties. The evaluation provided [Blackstock et al., 2008] demonstrates the UIF ability to integrate different services,
3.3. Controlled Sharing and Management of Context Data

in particular Context Toolkit, iROS, Equip Component Toolkit and MUSEcap. However, the evaluation is situated in a single domain where privacy is not as major a concern as when dealing with multiple domains. While UIF is technically capable of exchanging context data with third-parties, in reality it is not a feasible option due to its privacy limitations.

Mobile Phone Based

Hydrogen: The Hydrogen Project \textsuperscript{(Hofer et al. 2003)} provides a three layer architecture for the development of mobile device based context-aware applications. Within Hydrogen there are two primary types of context, ‘remote context’ is information another device has and ‘local context’ is information that the device itself has. When two devices come in close proximity to each other their remote and local contexts are exchanged by first converting them to XML streams and then sending them via Bluetooth or WiFi.

Hydrogen uses an object-oriented approach for modeling context such as time, location, device, user, network. These context objects are managed by the Context Server in the management layer which is responsible for providing and retrieving context objects. The context types can easily be extended to include application-specific context types. While object-oriented context modelling is far more flexible than the attribute-value tuples of older approaches, it lacks the level of formality which an ontology based representation provides and as a result it is not the most suitable solution for a context sharing scenario.

The Hydrogen architecture does not enable the sharing of context data with heterogeneous applications due to its heavy use of non-standard interfaces and context representations. While interfaces are provided for the provision of context data to hydrogen applications and P2P connections are used for sharing context data with other Hydrogen devices, they are not suitable for third-party applications to retrieve context data without significant customisation.

As a user moves from location to location their Hydrogen device, which manages the user’s context data, carries their context data with them. Therefore, the discovery of context data concerning a new user is straightforward as the enquiring service simply has to query the user’s Hydrogen device for the desired data. However, this model requires the mobile device to be physically co-located with the application or service it is exchanging context with and location independent context exchange is not possible.

The issue of privacy within the Hydrogen Project is an open issue and is not taken into account within the present architecture. Accordingly, there is also no granularity mechanism within Hydrogen. As with Nexus, the lack of privacy controls is of great concern given the potentially sensitive nature of the data being shared.

Pervasive Web: The Pervasive Web is a concept developed by Prehofer et al. \textsuperscript{(Prehofer et al. 2009)} at Nokia Research Center, Helsinki. As mobile devices are increasingly becoming hubs of content and context information Prehofer et al. wish to support the use of mobile
devices for pervasive applications. In particular, they wish to use web technologies on the mobile devices to enable the provision and consumption of pervasive services within dynamic smart spaces. REST is used for the creation of pervasive mashups, OpenID and OAuth for authentication and a search engine for locating resources within a smart space. The use of DCCI (Delivery Context: Client Information) [W3C 2007] for sharing the context within a device is also investigated.

The focus of this work is on the development of a platform to enable practical smart space deployments through the use of web technologies. Web technologies are used because there are many mature tools and libraries available for developer use, the majority of devices can consume RESTful web services and certain devices can also provide RESTful web services.

The approach adopted by Prehofer et. al. revolves around the mobile device and thus assumes that all potential users will have suitable phones with the appropriate services installed. If a user did not have their mobile device with them or if they do not have a compatible phone they cannot access the services offered by the smart space as location independent context exchange is not possible. It is unfortunate that such an assumption is made as the system is innovative and pragmatic. However, the extensive use of widely supported web technologies and the provision of interfaces for the exchange of context data results in a system which third-parties could interact with minimal effort.

The privacy mechanisms provided are based upon the OpenID and OAuth standards. Their crude ‘grant’/‘deny’ resource access options are unsuited for application to sensitive and dynamic context data. Furthermore, the concept of context granularity is not considered within Pervasive Web.

Pervasive Web supports several types of context data such as location, profile, friends and preferences. Due to the extensibility of DCCI it is also possible to extend the types of context supported by Pervasive Web. However, the context data used within Pervasive Web does not conform to a formal ontology and hence the context data shared between applications lacks formal semantics.

Web-based Brokers:

This section details three web-based brokers. Google Latitude and Yahoo!’s Fire Eagle are of interest as they manage and share location data, which is a major type of context data, and make that data available to third-party applications. Both applications provide similar functionality, but have a different focus. IYOUIT deals with a wider range of context types than the other two brokers and also provides a mobile phone application for gathering context data.

Google Latitude and Yahoo Fire Eagle: Google Latitude was initially released in February 2009. Its primary focus is the sharing of location data between users rather than sharing location data with third-party applications. Latitude has a location history feature
which is used to visualize the user’s movement over a given period of time. Additionally, it includes ‘Alerts’ which can send notifications when a users is in an unusual location or when friends are nearby.

Google Latitude does not offer a formal API for application developers. However, location information can be gathered from ‘Latitude Badges’, these are widgets that can be embedded into webpages and display the location of the user on a map. An XML or JSON feed can be retrieved from these badges, thus allowing a user’s location to be used by third-party applications. There are no complex privacy controls available, badges are simply turned on or off. As a result, the privacy controls when sharing location data with applications is extremely limited, as users can only specify full access to all applications or no access to any application. Third-party applications cannot update a user’s location, location is updated by the user using Google Mobile Maps on their mobile phone or via the Google Latitude web interface. Therefore, the full exchange of context data with third-parties is not supported, rather they allow only the sharing of context but not the updating of context. In contrast the level of functionality provided when sharing location data directly with friends is much greater. A user can hide their location for an individual friend, allow certain friends access only to their location at city level or allow others full access to their location.

Fire Eagle takes a more application-oriented focus to location sharing than Google Latitude. The concept of friends does not exist within Fire Eagle and, therefore, users do not have any direct method for sharing their location information with their friends using Fire Eagle. However, third-party applications, such as ‘Friends on Fire’ ([3]), can be used to provide this functionality. When an applications requests access to a user’s location information, users can choose to reject their request or allow access at one of seven different granularity levels, as shown in Table 3.1.

Fire Eagle offers developers a comprehensive API for managing and gathering location data. A user’s location can be updated or retrieved by applications. Applications can make requests for user data within a certain geographical area such as within the range of a cell tower, in a particular location or at a particular venue. In order for an application to be able to query a user’s location the user must provide permission at the granularity level appropriate to the query.

As both Latitude and Fire Eagle are centralized, developers are forced to rely on a single third-party services for the management of their location data, thus risking the creation of ‘Big Brother’. Developers are forced to abide by their terms of service, which are subject to change, even for data which they collect and are using the broker to manage. The centralized approach may be of benefit to certain applications, however, it can also introduce avoidable performance inefficiencies for applications wishing to avoid the additional remote communication overhead.

The privacy policies of both Fire Eagle and Google Latitude do not offer adequate functionality to represent the complexities of location-specific access permissions. The specification of a singular access level for a given friend or application is not sufficient to model the user’s access levels. For instance, if a user wishes to allow their boss access to their location
only during working hours, neither Fire Eagle or Google Latitude provide the functionality necessary for such permissions.

The discovery criterion (C3) requires that a user’s context data must be capable of following them as they move between ubicomp environments. While third-parties may be able to consume context data from both Fire Eagle and Latitude, there is no mechanism available for identifying users as they enter a new ubicomp environment. Users must provide account details to the environments rather than being automatically identified through the use of, for example, face recognition, gait recognition, Bluetooth MAC-address or RFID.

As both services focus on location, the utility of their respective service to interacting applications is limited when compared with services such as IYOUIT which offers similar functionality but for a far wider range of context types. However, Google Latitude and Fire Eagle have demonstrated that there is a demand for location-based services both from a user and a developer perspective.

**IYOUIT:** IYOUIT, which evolved from the ContextWatcher project [Koolwaaij et al. 2006], is comprised of a mobile phone application capturing context data and a web-based context broker that aggregates the data. The mobile phone application captures data such as nearby WiFi and Bluetooth devices, GPS-based location of the phone, and photos and sound recordings taken by the user. This context data is augmented with weather information from third-party services and users directly entering their moods or information on products they are consuming. IYOUIT supports a far greater number of context types than Fire Eagle and Google Latitude and it represents a significant step towards the creation of a comprehensive personal context data repository. Information such as user location, nearby Bluetooth and WiFi access points, mood and activity (which are manually entered), photos and nearby weather can be made available.

Interfaces are provided for third-party applications to access context data about users, via the the web-based context broker. The updating of context data is restricted to the Symbian mobile phone application, which gathers the user’s context data and uploads it to the IYOUIT web service. As a result, the full exchange of data with third-parties is not possible. Third party applications cannot provide context data, they can only retrieve context data from the
web service. Furthermore, due to its tight coupling with Symbian phones and the inability to update the service without using the mobile application, IYOUIT is restricted.

IYOUIT suffers from the same discovery issues as Fire Eagle and Google Latitude. The inability to relate a user within a physical space to their IYOUIT profile requires direct interaction between the user and the ubicomp service which could be avoided if a user identifier lookup service was provided.

In a similar fashion to Pervasive Web, the user privacy settings provided with IYOUIT are crude 'grant/deny' permissions for particular context types. It is also not possible to share the gathered context data with friends at varying levels of granularity. While a formal ontology is used to model the context data, unfortunately, this ontology has not been made publicly available. Additionally, it is not possible to add new context types to the service.

### 3.3.3 Summary of Existing Approaches

The main related work have been identified and analysed in this section. Table 3.2 outlines their support for the criteria defined in Section 2.5. It is clear that none of the systems provide complete support of the identified criteria. However, no single criterion is left uncovered by the systems. This coverage of each criterion demonstrates that there is a need for fulfilment of each criterion within ubicomp systems related to the controlled sharing of context data.

The exchange of context data between third-party, heterogeneous applications is supported by Pervasive Web, UIF and Fire Eagle, while the remaining two web-based brokers support the sharing, rather than the exchange, of context data. Context Bridge and CoBrA are the only systems which do not expose some class of interfaces for external data access. The mobile phone-based solutions cannot provide location independent sharing of context data and as a result of being a single environment system without interfaces for sharing data CoBrA also cannot provide location-independent sharing of context data with third-parties. Overall UIF is the sole system which provides full coverage of Criterion C1.

With the exception of Context Bridge, the remaining nine systems support the management of context data. Context Bridge does not directly support the management of context data, however, the bridged ubicomp application may support it.

The web-based brokers do not have support for discovery among ubicomp environments, as they do not incorporate the concept of inter-environment mobility. This deficiency is due to the fact that they are not ubicomp applications and ordinarily do not require such functionality. Ubicomp environments relying on Fire Eagle or Google Latitude have no way of recognising new users or mapping users to their web-based context store. This is not an issue when the services are used with a web application. However, when used within a ubicomp environment it introduces the need for user input. CoBrA does not require resource discovery across third-party environments as it is a classic example of a single environment ubicomp framework and, therefore, lacks any inter-environment mobility capabilities. The mobile phone-based systems, Pervasive Web and Hydrogen, can seamlessly move between
environments as their user’s context data is transported with them on their phone. The bridge approaches support discovery as the bridged systems are treated as one.

While context privacy is crucial for the adoption of ubicomp services, many systems ignore the issue and do not provide suitable privacy control mechanisms. UIF and Nexus do not provide any mechanisms for securing the privacy of context data. This is of concern given that their primary function is the sharing of context data between heterogeneous systems. Google Latitude provides extremely limited support for securing privacy with respect to applications. Users can merely grant access to all applications or deny access to all applications. No mechanism is provided to control the access of individual applications. IYOUIT, Hydrogen and Pervasive Web are marginally better than Google Latitude by providing ‘grant’/‘deny’ access on a per-application basis. The remaining systems provide sufficient user privacy controls. Context Fabric in particular places emphasis on the protection of context privacy. During the development of the Context Fabric privacy policy framework the legal requirements were analysed and parts of the requirements were integrated into the framework.

Granular access to context data is directly connected to the provision of privacy controls and as UIF does not support privacy policies it also does not support the specification of access at particular levels of accuracy. CoBrA, Context Fabric and Fire Eagle allow users to specify what accuracy or granularity of access they wish to grant applications. The remaining seven systems do not have such functionality.

CoBrA, UIF and IYOUIT provide formal representation of the context data used in their systems. As a result services consuming this data can understand its meaning. Fire Eagle and Google Latitude are solely location brokers and, therefore, do not support any types of context other than location data. The remaining eight systems support a wider range of context data, however, the exact number and type of contexts varies from system to system. The web-based brokers do not offer the possibility to extend the types of context data used. Unlike the other frameworks and middlewares which can be customised by individual developers, the web-based brokers are third-party services and thus are outside of the control of developers using them.
### 3.3. Controlled Sharing and Management of Context Data

<table>
<thead>
<tr>
<th>Name</th>
<th>C1: Exchange</th>
<th>C2: Privacy</th>
<th>C3: Discovery</th>
<th>C4: Manage</th>
<th>C5: Context Types</th>
<th>C6: Device Agnostic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1.1: 3rd Party</td>
<td>C1.2: Interfaces</td>
<td>C1.3: Location independent</td>
<td>C2.1: User Controlled</td>
<td>C2.2: Granularity</td>
<td></td>
</tr>
<tr>
<td>CoBrA</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Context Fabric</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Nexus (extended)</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Context Bridge</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>n/a</td>
</tr>
<tr>
<td>UIF</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Hydrogen</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Pervasive Web</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Fire Eagle</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Google Latitude</td>
<td>P</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>IYOUIT</td>
<td>P</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

* Table 3.2: Comparison of Related Work

* N=no support, P=partial support, Y=full support, n/a=not applicable
Three of the systems, Pervasive Web, Hydrogen and IYOUIT, require the user to carry a mobile phone and have the appropriate software running in order to interact with the ubicomp service. This restriction is unreasonable as it reduces the number of potential users, as not all users will carry the required phone with the required software, and restricts the type of environment which the ubicomp application can be deployed in. While Google Latitude and Fire Eagle also have mobile clients, it is also possible to interact with them from any device with a web browser.

3.4 Analysis of Related Work

Table 3.2 summarises how the core related work compares with the criteria established in Chapter 2. This section analyses the findings to identify the key features and omissions of the core related work relative to the criteria.

Initially the individual criteria will be examined separately and subsequently a number of interdependent criteria will be combined and analysed together. Throughout the criteria analysis two high-level categories of related work are frequently referenced, web-based brokers and ubicomp systems. These two approaches have attracted particular attention due to the contrasting respective strengths.

3.4.1 Individual Criteria

Not all of the identified criteria are of equal importance to the overall vision of the controlled sharing of context data between heterogeneous third-parties. The three key criteria, in order of precedence, are: (i) the exchange of context data with third-parties, (ii) the protection of the privacy of the context data, and (iii) the discovery of context data.

C1. Exchange context data with remote heterogeneous third-parties

As previously indicated criterion C1 is the most important of the criteria. It is clear from the table that the web-based brokers provide better support for the exchange of context data than the remaining, more traditional, ubicomp systems. This is primarily due to the Web 2.0 background of the web-based brokers where the provision of APIs for services to interact with is common. In the case of the web-based brokers they each have APIs for providing context data and Fire Eagle can also receive context from authorized third-parties. With the exception of UIF, the remaining systems do not offer full support for C1. While the provision of interfaces is common they are typically for use by applications and services within the same framework and not by third-parties.

C2. Privacy protection

The privacy of context data has poor support across both categories and most of the surveyed applications. This is of particular concern given the sensitive nature of context
3.4. Analysis of Related Work

data, in particular when dealing with the sharing of context data with third-parties as is the case with UIF and Pervasive Web. Privacy is often cited (Cardoso and Issarny 2007; Hong and Landay 2004) as the primary reasons for the low number of ubiquitous computing deployments and, therefore, the importance of the protection of user’s context data cannot be over-emphasised. The issue of user privacy is challenging and frequently ignored in ubicomp research, such as in UIF, in order to concentrate on other aspects of ubicomp research.

C3. Capable of handling user discovery between ubicomp environments

The web-based brokers are weak in terms of the discovery of context data (C3) and conversely the more traditional ubicomp approaches provide widespread support for this feature. This disparity is because in their traditional setting the web-based brokers would not need to discover remote sources of context data concerning a particular user. For instance, when users wish to share their Fire Eagle location data with a third-party the user would provide them with the details and permissions necessary to access that data. As a result there is no need for discovery in such scenarios.

C4. Ability to manage context data

The management of context data is a widely supported feature which emphasises its importance. From a research perspective the novelty of C4 is limited. However, the provision of context management functionality is necessary for providing applications and services with access to new and old context data.

C5 Context Types

Criterion C5, and its sub-criteria, concern the context representation capabilities of the surveyed systems. It can be seen that the majority of the systems support multiple types of context data and those types of context data are extensible, thus allowing developers to include application specific context types. The web-based brokers have poor support for context data beyond location data and do not allow the addition of application-specific context data. There is a clear disparity between the focus of the web-based brokers, whose primary concern is sharing location data with location-aware applications and services, and the ubicomp systems, who provide more complete context functionality.

The use of a formal model to represent the context model (C5.1) is not common practice within the traditional ubicomp systems or the web-based brokers. The reason for this directly relates to the sharing of context data with heterogeneous third-parties (C1.1). Formal models, in particular ontologies, provide a common understanding of a shared conceptualisation, which is of particular importance when sharing context data with third-parties. This is not of great importance when sharing context data within the same application or service as the semantics of a particular context type, such as location, is inherently understood. However, if, for example, the three variables of a location object (x, y, z) were shared with a third-party
it would not be clear what exactly the variables represented and what coordinate system was being used. Therefore, while a formal context model is not widely used within the surveyed systems, it is of particular importance in relation to the controlled sharing of context data between heterogeneous third-parties.

C6. Device Agnostic

The majority of the systems are hardware agnostic and this is clearly a desirable feature as it does not restrict the user’s interaction methods with the system. The three systems that are not device agnostic have integral mobile phone applications that users must utilize in order to interact with the context-aware applications or services. This type of approach introduces an unnecessary constraint in the deployment options for such systems.

3.4.2 Combined Criteria

Section 3.4.1 examined how the core related work supported the individual criteria. The following section analyses their support for combinations of criteria. While there is no single criterion that is unsatisfied by all of the core systems, when criteria are combined, as shown in Table 3.3, deficiencies across all of the system categories in the related work become clear. The criterion combinations were decided upon by identifying complimentary criteria, where the satisfaction of one criterion within a pair would only provide marginal benefits without the satisfaction of the second criterion. This section will identify and discuss these key combined criteria.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1</td>
<td>Exchange of context data with privacy</td>
<td>C1.1, C2.1</td>
</tr>
<tr>
<td>CC2</td>
<td>Discovery of context data with a common context model</td>
<td>C3, C5.1</td>
</tr>
<tr>
<td>CC3</td>
<td>User controlled privacy protection with a formal context model</td>
<td>C2.1, C5.1</td>
</tr>
</tbody>
</table>

Table 3.3: The Key Combined Criteria (CC)

C1.1 & C2.1: Exchange of context data with privacy

When considering the controlled sharing of context data between heterogeneous third-parties the two criteria C1.1 and C2.1 are interconnected. C2.1 considers the controlled portion of the statement while C1.1 considers the sharing with heterogeneous third-parties. Table 3.4 shows that only one of the surveyed systems, Fire Eagle, supports the exchange of context data with third-parties while providing users with mechanisms for securing their context data, this is a significant concern given the sensitive nature of context data. The sharing of context data with third-parties without mechanisms for protecting that data results in a system that is not suitable for real-world deployments. The web-based broker, Fire Eagle,
is the only system to provide functionality for the sharing of context data with third-parties while offering privacy protection mechanisms to the user.

<table>
<thead>
<tr>
<th></th>
<th>C1.1: 3rd Party</th>
<th>C2.1: User Controlled</th>
<th>C1.1 ∧ C2.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoBrA</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Context Fabric</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Nexus (extended)</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Context Bridge</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>UIF</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Hydrogen</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Pervasive Web</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td><strong>Fire Eagle</strong></td>
<td><strong>Y</strong></td>
<td><strong>Y</strong></td>
<td><strong>Y</strong></td>
</tr>
<tr>
<td>Google Latitude</td>
<td>P</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IYOUIT</td>
<td>P</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 3.4: Criteria C1.1 and C2.1

**C3 & C5.1: Discovery of context data with a common context model**

The use of device based identifiers is common in ubicomp systems (Hervs and Chavira, Siegmund and Florkemeier, 2003), and a variety of wireless device types can be used for the identification of users, such as RFID, NFC, Bluetooth and WiFi MAC addresses. When a user enters a ubicomp environment for the first time the environment must be able to discover context information about this new user. In a distributed heterogeneous setting, as is being considered in this thesis, the use of a formal model to represent the user’s device identifier is important. In order to make queries to third-parties concerning a user’s identifier there must be a common understanding of how to represent and interpret those identifiers, otherwise an incorrect or no result may be returned. Therefore, the device context model must be an integral part of the discovery context mechanism.

Table 3.5 summarises the support for the second combined criteria by the surveyed systems. As can be seen in the table, only UIF provides a formal model for device identifiers that is used in their discovery process. This omission on the part of the other system illustrates their unsuitability to deployment in distributed heterogeneous scenarios. The web-based brokers are particularly poor in relation to CC2. Once again this is due to their focus on Web 2.0 oriented scenarios, where interacting websites are directly provided with the location of the user’s context data rather than being provided with a device-based identifier. For use within a distributed heterogeneous ubicomp scenario this failing on the part of the web-based brokers would require addressing.

The majority of the traditional ubicomp systems do support the discovery of context data, however, they fall short regarding the provision of a formal context model. As outlined in Section 3.4.1 this is primarily due to their shortcomings regarding the sharing of context
data with third-parties.

<table>
<thead>
<tr>
<th></th>
<th>C3: Discovery</th>
<th>C5.1: Formal Context Model</th>
<th>C3&amp;C5.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoBrA</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Context Fabric</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Nexus (extended)</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Context Bridge</td>
<td>Y</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>UIF</strong></td>
<td><strong>Y</strong></td>
<td><strong>Y</strong></td>
<td><strong>Y</strong></td>
</tr>
<tr>
<td>Hydrogen</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Pervasive Web</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Fire Eagle</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Google Latitude</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IYOUIT</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

\(N=\text{no support}, \ P=\text{partial support}, \ Y=\text{full support}\)

Table 3.5: Criteria C3 and C5.1

**C2.1 & C5.1: Privacy protection with a formal context model**

As can be seen in Table 3.2 the majority of traditional ubicomp applications do not provide functionality for exchanging context data with third-parties (C1.1), as a result the use of a formal context model is not a priority for them as the semantics of the internally exchanged data can be incorporated programmatically. This thesis focuses on distributed heterogeneous scenarios where the use of a formal context model is important. In particular, the use of a formal context model is relevant to the user’s privacy protection preferences. As mentioned in Section 3.4.1, user’s privacy concerns are cited (Cardoso and Issarny, 2007; Hong and Landay, 2004) as the primary reasons for the low number of ubiquitous computing deployments. In order for exchanged context data to be understood by third-parties, a formal model must be used to represent the data. Directly related to this is the topic of context privacy. Users utilize privacy policies to specify who can access what context data, under what circumstances and at what level of accuracy / granularity. Specifying what context data, what circumstance and what level of accuracy/granularity all incorporate aspects of context data. What context data specifies a particular type of context data, what circumstance is a set of conditions based on context values and what level of accuracy/granularity is a hierarchy of related context types. For third-parties to understand and enforce access to a user’s context data, as specified in a their privacy policy, the use of a formal model is required.

Table 3.6 outlines the support the surveyed systems have for criteria C2.1 and C5.1. CoBrA is the sole surveyed system to provide its users with control over their context data and a formal context model for describing that context data. Of the remaining nine systems, four of them neither support C2.1 or C5.1. UIF and IYOUIT are the only two systems to support C5.1 but not C2.1. IYOUIT internally makes use of a formal context model, however, this is of limited use to third-parties as the model is not made publicly available.
3.5. Summary

The remaining three systems only provide user controlled privacy protection but not a formal context model.

<table>
<thead>
<tr>
<th></th>
<th>C2.1: User Controlled</th>
<th>C5.1: Formal Context Model</th>
<th>C3&amp;C5.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoBrA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Context Fabric</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Nexus (extended)</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Context Bridge</td>
<td>Y</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>UIF</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Hydrogen</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Pervasive Web</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Fire Eagle</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Google Latitude</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IYOUIT</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

N=no support, P=partial support, Y=full support

Table 3.6: Criteria C2.1 and C5.1

3.5 Summary

Given that every criterion is supported by a minimum of three systems indicates that an appropriate subsection of systems was chosen for analysis based on the identified criteria. Some of the criteria, such as the management of context data (C4), are widely supported signifying that, while there may be limited research novelty associated with such features, they are critical for ubicomp systems.

No single system exists that satisfies all of the identified criteria. In previous sections we have demonstrated that there is a need for such functionality. Consequently it is clear that deficiencies exist in the related work. While looking at individual systems does provide some insights into the related work, analysing the criteria according to the categories of related work provides additional insights. In particular, the divide between web-based brokers and the ubicomp systems is informative.

The web-based approaches excel at the criteria related to the exchange of context data (C1). This is of particular relevance to the issues addressed by this thesis, as described in the Problem Statement in Section 1.2 as C1 is the highest priority of the criteria. Due to their Web 2.0 background, and its principle of Openness (Anderson 2007), the web-based brokers provide interfaces for services to interact with. The combination of the interfaces with the highly accessible nature of web-based services means that the web-based brokers provide location independent sharing by default as part of their model.

Web-based approaches are weakest at the criteria where the ubicomp systems are strongest. In particular, they struggle with privacy protection, supporting multiple context types and allowing the extension of their supported context types. While their strength with respect to C1 pertains to their Web 2.0 background, their deficiencies with regard to the other criteria
also relates to their background. The current trend within many Web 2.0 services and applications is for the integration of location-based functionality, as a result the surveyed systems have poor support beyond location data. Features such as Discovery (C3) are not typically relevant to the web-based brokers as they are not designed to be used within ubicomp applications.

The strengths of the classical systems analysed are generally the inverse of the web-based systems, they are weak at the exchange of context data (C1) but strong in terms of the traditional ubicomp system attributes. Their weakness in relation to C1 is as a result of the typical lack of support for third-party software within ubicomp applications, in particular the exchange of context data with third-parties. While interfaces may be provided they are often unsuited to use by third-parties due to the lack of a formal context model (C5.1) or lack of privacy controls (C2.1) as is the case with Pervasive Web and Nexus respectively.

Unsurprisingly, the ubicomp systems surpass the web-based brokers in their support for the ubicomp specific criteria. Features such as discovery and extensible multi-type context are supported across the majority of the ubicomp systems surveyed. The support for privacy and a formal context model is less uniform across the ubicomp systems. However, both features are necessary for the addressing of the issues presented in this thesis.

Overall it is clear that no existing system meets all the criteria for the controlled sharing of context data between heterogeneous third-parties. When examining Table 3.2 a clear distinction between the strengths of ubicomp systems and web-based brokers becomes obvious. We can see that neither category unequivocally supports all of the categories. However, a hybrid approach that chooses the strongest parts of each approach may be the most appropriate for addressing the previously established issues.
Chapter 4

Context as a Service and the Network Model

The previous chapters have established the benefits of, and need for, the controlled sharing of context data, as well as the deficiencies in the related work concerning this objective. By sharing the context data currently contained within isolated ubicomp environments, applications and services can react to events that happen outside of their domain of operation and provide enhanced service to their users. However, as yet there is no system that meets the criteria necessary for this goal. The two primary categories of related work examined, ubicomp applications and web-based brokers, provide two different, but incomplete, approaches to this issue. A solution to the controlled sharing of context data is introduced in this chapter and is elaborated upon in the following chapters.

This chapter begins by describing the approach adopted, designated Context as a Service (CxaaS), and the reasoning behind it. A set of requirements are then defined based on the adopted approach. Following this, an overview of the three primary parts of CxaaS is provided and the latter part of this chapter will focus on one of those parts, the CxaaS Network Model (CxaaS-Net). The remaining two CxaaS parts, the node model and the data model, will be discussed in detail in subsequent chapters.

4.1 Requirements

4.1.1 Adopted Approach

CxaaS adopts a hybrid approach built upon a web-based broker foundation. It focuses on ubicomp scenarios and leverages the existing ubicomp research where appropriate. This hybrid approach harnesses the beneficial aspects of both types of approaches to provide a model for the controlled sharing of context data between heterogeneous third-parties.

From the analysis of the core related work in Section 3.4 it is clear that no existing system or architecture type from the related work meets all of the criteria for a system
supporting the *controlled sharing of context data between heterogeneous third-parties*. A disparity between the two categories of related work is apparent based on their strengths in relation to the identified criteria. The web-based brokers excel at the sharing of context data while, unsurprisingly, the ubicomp systems are superior in terms of the discovery of context data and supporting multiple types of context data; which are traditionally associated with ubicomp systems.

Of the identified criteria *C1 Exchange* has the highest precedence and this is the criterion at which the web-based brokers excel. The web-based broker’s deficiencies in relation to the ubicomp specific criteria are not related to an inherent problem when applying a web-based broker model to a ubicomp scenario. Rather, the surveyed systems were not originally designed for ubicomp applications, but for the sharing of location data with friends or location aware-services. Hence the identified failings can be rectified by focusing on ubicomp scenarios from the outset.

The web-based brokers adopted a centralised approach whereby all the context data was pushed to a central, web-based, service. This structure is not appropriate for deployment in ubicomp scenarios as it requires ubicomp environments to push potentially sensitive context data to a third-party. Additionally, it does not complement the traditional, *classical*, structure of ubicomp applications whereby all context data is stored locally and reliance on third-parties is minimal. The adopted approach treats the network of ubicomp nodes as a distributed set of web-based brokers, where each broker stores the context data of that node and when necessary requests context data from other brokers. This approach compliments the *classical* systems and minimises disruption when adapting such systems.

This distributed web-based broker approach, known as Context as a Service, introduces several problems that the centralised web-based approaches do not have to deal with, in particular the identification of users and the discovery of data concerning those users. These issues are addressed by some ubicomp systems, however, their approaches are not suitable for a privacy sensitive web-based broker approach due to fundamental differences between their models. Additionally, a context model is required that is suitable for use within a network of distributed web-based brokers as the brokers must have a clear understanding of the data being exchanged.

Another new issue is the distributed protection of user’s context data. The privacy protection mechanisms currently used in the centralised web-based broker approach and the *classical* ubicomp systems are not applicable to a distributed version. Both approaches are centralised and do not have to enforce privacy controls across multiple parties. It would not be practical for users to register and create privacy policies with each environment they interact with. Therefore, a solution to this issue is fundamental to the realisation of the controlled sharing of context data.
4.1.2 Core Requirements

Using a distributed web-based broker approach as a foundation, this section will establish the requirements that must be fulfilled in order to address the issues outlined in Chapter 3. The core requirements focus on the failings of the existing web-based approaches when applied to a ubicomp scenario, once these failings are addressed the outcome should be a complete model for the controlled sharing of context data between heterogeneous third-parties. The shortcomings, as taken from the analysis in Section 3.4, are the discovery of context data (C3), privacy protection (C2) and context types (C5). Table 4.1 summarises the core requirements for the provision of CxaaS and the following chapters discuss those requirements in more detail.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.1</td>
<td>Mechanisms for the discovery of new context data concerning new or returning users</td>
</tr>
<tr>
<td>R.2</td>
<td>Mechanism available for sharing context data in a controlled, privacy sensitive manner</td>
</tr>
<tr>
<td>R.3</td>
<td>A formal context model suitable for the sharing of context data between third-parties, appropriate for use within the discovery and privacy mechanisms</td>
</tr>
</tbody>
</table>

Table 4.1: Core Requirements

**Context Discovery:** Ubicomp environments detecting new or returning users must be provided with the functionality necessary to discover existing context data concerning those users. When a user enters a ubicomp environment they could be carrying one of a range of devices capable of uniquely identifying them, such as RFID, NFC, Bluetooth or WiFi MAC addresses. From these device based identifiers the ubicomp environment must be able to locate existing context data that was detected by third-parties and is stored remotely.

This requirement relates to criterion C3 which is one of the deficiencies of web-based brokers.

**Controlled Context Sharing:** Following on from the initial core requirement, R.1, once the required context data has been located it must be possible for the querying node to request the required context data. The request must be answered in a manner that respects a user’s privacy wishes as represented in the user’s privacy policy. Context data can be very personal and sensitive and, therefore, the protection of the privacy of the data is extremely important. The majority of the surveyed systems were weak in terms of the privacy protection aspect of the identified criteria.
This requirement relates to criterion C1, which is one of the strengths of web-based brokers, and C2, which is one of its deficiencies.

A Formal Context Model: Continuing from the previous requirement, R.2, once the context data has been retrieved by the requesting node they must be able to understand the shared context. A formal model for the shared context ensures a common understanding of the shared content and it can capture the relationship between different types of context data. Additionally, formal models allow nodes to reason over their context data and infer new knowledge from the initial context information.

The formal context model must be appropriate for use in the discovery of context data as nodes will be looking for a particular type of context data from a particular user across all of the relevant nodes. Additionally, the privacy policies of users will be enforced across all nodes sharing the user’s context data. The privacy policies will specify access rules for particular types of context data, where the conditions may depend on the values of other types of context data. Therefore, it is crucial that nodes have a clear understanding of the context types and values mentioned in the privacy policy, this is possible through the use of formally modelled context within the policies.

This requirement relates to criterion C5 and is being applied to discovery and privacy which are criteria C3 and C2 respectively.

4.1.3 Functional Requirements

Expanding from the core requirements a set of functional requirements, can be derived where the emphasis is on features required to satisfy the core requirements rather than on novelty and differentiation from the related work. These functional requirements are classified under three headings: Exchange of Context Data, Discovery and Mobility and Privacy Protection. Each of the headings are discussed in the following paragraphs, while Table 4.2 summarises and enumerates the requirements and sub-requirements.

Exchange of Context Data: The exchange of context data not only concerns context data but also the interfaces which handle the context data exchange. In order for context data to be shared between heterogeneous third-parties, discoverable interfaces must be provided through which the context data can be accessed. While those third-parties may be able to locate remote interfaces they must also be capable of understanding the functionality offered by the interfaces and how to interact with them.

Discovery and Mobility: The discovery of information concerning users initially requires the identification of users. As users enter ubicomp spaces, the services they interact with must be capable of uniquely identifying them, regardless of whether they are new or returning users. Without the ability to identify its users, the ubicomp environments would be unable to discover context data about them and provide personalised services to the users. Existing
research often ties implementations to one particular user identification mechanism, such as RFID. Such implementation restrictions are not practical as a single identification mechanism will not be suited to all ubicomp environments. Once a ubicomp environment has identified a user they must be able to locate context data concerning that user. That context data could potentially be located across a number of CxMSs, depending on the number of environments the user has interacted with.

**Privacy Protection:** Comprehensive privacy protection mechanisms, for both users and ubicomp environments, are essential to the adoption of the proposed model for context sharing. Given the distributed nature of the CxaaaS model it must be possible to enforce a user’s privacy policy across all nodes accessing their data rather than requiring users to create a new policy each time they enter a new ubicomp environment.

Simply being able to grant/deny access rights to a service is not expressive enough to deal with the dynamic nature of context data. The ability to specify access permissions at varying levels of granularity based on context values is necessary. Additionally, users are not the only actors involved when data is shared between ubicomp services. The ubicomp service providing the data will also have privacy requirements that must be enforced.

**Individual Nodes:** The requirements for the successful sharing of context data are not restricted to the network level, there are also requirements at the local node level. The most suitable programming languages to use and interfaces to provide will vary from one ubicomp deployment to another. However, the manner in which nodes handle context data must be consistent from node to node, across the entire lifecycle of the context data, in order that users can be confident that their context data is secure and handled appropriately. Context data privacy protection must be incorporated as part of the lifecycle. Therefore, nodes must be able to enforce the user’s policies, from both local and distributed sources.

### 4.2 Introduction to Context as a Service

In order to overcome the challenges detailed in Section 2.3 and fulfil the requirements provided in Section 4.1 this thesis proposes a service based model for the provision of context data which is called Context-as-a-Service (CxaaaS).

#### 4.2.1 CxaaaS Primary Components

The service based model, illustrated in Figure 4.1 is comprised of three high-level segments: (i) the network architecture (CxaaaS-Net), (ii) the node architecture (CxaaaS-Node) and (iii) the data model (CxaaaS-Data). At a node level CxaaaS represents the concept of embodying the entire lifecycle of a piece of context data, from creation to deletion and any steps in between, in a single service. While at a network level CxaaaS provides a model for transparent discovery and acquisition of context data, in a privacy sensitive manner, from
### Table 4.2: Functional Requirements

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR.1</td>
<td><strong>Exchange of Context Data</strong></td>
</tr>
<tr>
<td>FR.1.1</td>
<td><em>Interfaces for the provision of context data to third-parties:</em> Interfaces must be provided through which the context data can be accessed.</td>
</tr>
<tr>
<td>FR.1.2</td>
<td><em>Formally described interfaces:</em> Remote nodes must be capable of understanding the interface’s functionality.</td>
</tr>
<tr>
<td>FR.2</td>
<td><strong>Discovery and Mobility</strong></td>
</tr>
<tr>
<td>FR.2.1</td>
<td><em>Identification of new and returning users:</em> All users entering a ubicomp space must be uniquely identifiable.</td>
</tr>
<tr>
<td>FR.2.2</td>
<td><em>Discovery of context data about identified users:</em> Once a ubicomp environment has identified a user they must be able to locate context data concerning that user.</td>
</tr>
<tr>
<td>FR.2.3</td>
<td><em>Multiple identification mechanisms:</em> Existing research often ties implementations to one particular user identification mechanism, e.g. RFID, such restrictions are not practical.</td>
</tr>
<tr>
<td>FR.3</td>
<td><strong>Privacy Protection</strong></td>
</tr>
<tr>
<td>FR.3.1</td>
<td><em>Distribution of Privacy Policies:</em> It must be possible to enforce a user’s privacy policy across all relevant nodes.</td>
</tr>
<tr>
<td>FR.3.2</td>
<td><em>Expressivity and Granularity:</em> The ability to specify access permissions at varying levels of granularity based on context values is necessary.</td>
</tr>
<tr>
<td>FR.3.3</td>
<td><em>Multiple Policy Types:</em> The ubicomp service providing the data will also have privacy requirements that must be enforced.</td>
</tr>
<tr>
<td>FR.4</td>
<td><strong>Individual Nodes</strong></td>
</tr>
<tr>
<td>FR.4.1</td>
<td><em>Implementation Agnostic:</em> The model should be implementation agnostic and allow the system developers as much implementation flexibility as possible.</td>
</tr>
<tr>
<td>FR.4.2</td>
<td><em>Manage the entire lifecycle:</em> Nodes must be capable of managing the entire lifecycle of context data in a privacy-sensitive manner.</td>
</tr>
<tr>
<td>FR.4.3</td>
<td><em>Policy Enforcement:</em> Nodes must enforce the policies, from both local and distributed sources.</td>
</tr>
</tbody>
</table>
heterogeneous third-parties. The data model provides a shared vocabulary that nodes can use to ensure the common understanding of data transferred, and use in the description of the services offered by nodes. Each of the primary parts of the CxaaS model have unique but complementary goals and they are all necessary in order to solve the problems and meet the requirements outlined in Chapter 3.

Figure 4.1: Overview of CxaaS Architecture

Figure 4.1 provides a high-level overview of the components of the CxaaS model. The figure shows a lookup service and two CxaaS nodes. The nodes provide interfaces for other nodes to access their context data, while the lookup service provides interfaces for discovering the primary source of a particular user’s context data. The diagram shows the three primary elements of the CxaaS model: (i) CxaaS-Net, which is made up of the lookup service and the interface portion of the CxaaS nodes; (ii) CxaaS-Node, which manages the context data and provides the functionality behind the interfaces; and (iii) CxaaS-Data, which defines the structure of the data exchanged between the nodes.

4.2.2 Node Level

Within the CxaaS model the CxaaS-Node is an integral component in the provision of the context-aware services within a smart spaces. It performs two primary tasks, (i) servicing the node’s local ubicomp environment and (ii) implementing the functionality exposed by the CxaaS network interfaces. Chapter 5 is dedicated to CxaaS-Node and it elaborates upon the
node level topics introduced in this section.

As the services offered by ubicomp environments can vary significantly it is not feasible to encapsulate all the potential services and related interfaces within the CxaaS model. Therefore, the CxaaS Model will focus on the core functionality required by nodes to enable the controlled sharing of context data. A CxaaS-Node can be viewed as a CxMS that provides a set of core interfaces and related functionality required to manage the entire lifecycle of context data and exchange context data with other CxaaS nodes in a privacy sensitive manner. CxaaS nodes can be extended with additional services that provide the application-specific functionality required for that particular deployment. The exact type of architecture (e.g. SOA, Agent-based, P2P) to be used internally within the environment is not constrained, however, externally the CxaaS node must adopt a web service-based approach. The type of web service used externally is not constrained, provided a Web Services Description Language (WSDL) 2.0 ([Moreau et al., 2006]) binding is available.

A CxaaS node’s ubicomp environment typically consists of users, context providers and context consumers interacting with a single CxMS, as depicted in Figure 4.2. In the image a CxaaS node is shown with a set of services that are operating within its local ubicomp environment. There is an indoor location system providing positional information within the environment. There is also a meeting manager, which consumes context data for the provision of context aware meeting services, and also provides meeting-related context data to the node. Finally there is an activity monitor that provides details of user activity within the environment and a ubicomp service provider that consumes context data in order to provide a context-aware service within the environment.

The CxaaS node must support the functionality required by the local environment, such as the one depicted in Figure 4.2, as well as providing the functionality behind the CxaaS interfaces. In isolation a CxaaS-Node and its ubicomp environment could be viewed as a system from the classical period, as outlined in Chapter 3. However, the addition of CxaaS-Net functionality expands the potential operational boundaries of the ubicomp services and moves the CxaaS beyond the classical period. The CxaaS node will be discussed in detail in the following chapter.

4.2.3 Network Level

CxaaS-Net builds upon CxaaS-Node functionality and provides a standardised mechanism for the discovery and exchange of context data, in a privacy sensitive manner, between physically distributed CxaaS nodes. This section provides a brief introduction to the network level of the CxaaS model, however, it will be discussed in greater detail in Section 4.3 of this chapter.

While CxaaS-Node is primarily focused on supporting its local ubicomp services and related context lifecycle, CxaaS-Net is focused on the external interoperation of CxaaS nodes,
4.2. Introduction to Context as a Service

in particular the architectural requirements for supporting the discovery and exchange of context data. The data model provides a commonly understood representation for information exchanged between the nodes.

The CxaaS Network is comprised of a set of interconnected CxaaS nodes, sharing context data, and a lookup service to facilitate the mobility and discovery functionality that is central to CxaaS. The CxaaS lookup service provides the methods for mapping user identifiers, such as RFID, to a URI representing that user. Based on that URI a user’s home node can be identified.

The home node is a CxaaS Node that is the primary storage point for a user’s privacy policies. Each user has a single home node and it is the only node that is aware of the location of all of its user’s context data. While the home node may not have a copy of each item of its user’s context data they will be aware of where queries should be redirected in order for the required information to be found. Along with storing the user’s privacy policies the home node also distributes the policies to nodes that share some of that user’s context data. If a user updates their privacy policy the home node must distribute the updated policy, or part thereof, to the appropriate third-party nodes with information concerning that user. Section 4.3 will discuss CxaaS-Net in more detail.
4.2.4 Data Model

The data model is composed of three parts: (i) an extensible OWL context ontology, (ii) a WSDL-based interface description specification, and (iii) privacy policy rules. Chapter 6 provides details of the context ontology and the service descriptions, while Chapter 7 deals with the privacy protection mechanisms within the CxaaS model. The CxaaS Context Ontology defines a core set of commonly used context types and re-uses existing commonly used ontologies where possible. The ontology is also an integral part of the two other CxaaS-Data components.

Service Descriptions

The service descriptions make use of the CxaaS ontology. The service descriptions provide a flexible mechanism for developers to describe the interfaces made available by their CxMS and how to interact with those interfaces. The specification uses the context ontology to specify the types of the context data inputted to or outputted from a particular interface. The CxaaS ontology and WSDL interface description is central to the sharing of context data between heterogeneous third-parties.

Privacy Policies

The ‘controlled’ portion of the ‘controlled sharing of context data’ is achieved through the use of a rules-based user privacy policy. The rules are used to specify conditions, based on the context types defined in the ontology, under which access is granted or denied to a user’s context data. CxaaS defines three types of privacy policies, each of which are shown in Table 4.3. Node and legal policies are not shared between nodes. However, user policies can be shared by the user’s home node to third-party nodes that wish to share context data concerning that user.

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>User policies define the conditions under which the subject of the context data is prepared to share context data concerning them</td>
</tr>
<tr>
<td>Node</td>
<td>Node policies define how CxaaS nodes wish to share context data that they own; for instance user location data that the node sensed</td>
</tr>
<tr>
<td>Legal</td>
<td>Legal policies define the legal requirements for the sharing of context data, these legal requirements may vary between jurisdictions</td>
</tr>
</tbody>
</table>

Table 4.3: Policy Types
4.2. Introduction to Context as a Service

Resource Identifiers

Uniform Resource Identifiers (URIs) are used to identify resources across the CxaaS-Net. The resources can be anything that must be referenced by nodes when sharing data; context data, users, CxaaS-Nodes or the lookup service can all be identified using URIs. The use of HTTP URIs is recommended in order to facilitate easy dereferencing and interlinking of the context data from multiple nodes.

4.2.5 Harnessing Existing Research

The proposed model provides significant opportunities for harnessing existing research across each of the three segments of the CxaaS Model.

Network Level

Adopting a service based approach for the sharing of context data between third-party heterogeneous context aware service provides many benefits. The loosely coupled nature of service oriented approaches is highly applicable when dealing with heterogeneous third-parties due to the wide variety of ubicomp implementations. Additionally, the provision of context in a service format allows for the leveraging of widely supported service-oriented development tools and libraries \cite{Hansson2004} for the development of interacting services.

Node Level

Due to the similarities between CxaaS nodes and classical ubicomp systems the existing research can be leveraged for the provision of local context-aware services and supporting the local ubicomp environment. In particular, the requirement for implementation agnosticism enables developers the flexibility to choose the most suitable implementation for their particular deployment, and where appropriate they may reuse existing research.

Data Model

The data model makes use of W3C standards for representing data exchanged between nodes. When appropriate existing data models, or parts thereof, are reused and external sources of context data are harnessed. For instance ontologies such as Friend Of A Friend (FOAF) \cite{Dodds2004} can be used for representing users, while Annotation CReatiON for Your Media (ACRONYM) can be used to represent devices \cite{Monaghan2007}. The reuse of such ontologies facilitates the reuse of existing data while increasing the understanding and interoperability of the concepts within the ontology. Finally, standardised technologies such as Web Services Description Language (WSDL) and Rule Interchange Format (RIF) \cite{Kifer2010} can be used to describe the exposed interfaces and policy rules respectively.
Chapter 4. Context as a Service and the Network Model

4.3 CxaaS Network

The CxaaS Network Architecture (CxaaS-Net) tackles the inherent problems of ubicomp systems outlined in Chapter 3, while adhering to the requirements established in Section 4.1. The CxaaS-Net defines the components and functionality necessary for heterogeneous applications to collaborate, share context and enable user mobility between smart spaces. CxaaS-Net is a distributed architecture and consists of a lookup service and a number of nodes providing the CxaaS core functionality.

CxaaS-Net differentiates the CxaaS model from classical ubicomp stove-pipes. It enables the mobility of users between environments, this includes the identification of new and returning users, the discovery of context data concerning those users and finally, the sharing of that context data. Additionally, it enables ubicomp environments to access context data originating from third-parties, therefore, they can react to events that occur outside of their direct domain of operation.

4.3.1 CxaaS Network Challenges

This section discusses the key challenges of the CxaaS network and describes the solutions adopted within the CxaaS model. The components mentioned as part of the solutions will be described in the following section.

Context Control

The control and ownership of context data have increased importance in a distributed setting where the context data is exchanged between nodes. In the classical ubicomp deployments the context data is detected by a single node and consumed by the same node, this simplifies the issue of context control and context ownership. Single environment creation and consumption of context data cannot be assumed in CxaaS. The issues of context ownership and context control are distinct but related and have effects on issues such as privacy and discovery.

When a ubicomp environment senses a user, and creates context data concerning that user, the environment is the owner of that context data. A user’s historical context data can be owned by a wide number of geographically distributed environments. Consolidating this in a single location could simplify the enforcement of the user’s privacy policies as all requests would be handled by a single node. Unfortunately, this type of model is not compatible with classic ubicomp systems, where the environment retains total control over the context data.

By allowing the environments to continue storing their context data the CxaaS Model is similar to a connected group of classical period systems rather than an entirely new paradigm for distributed ubiquitous computing, as would be the case if a consolidated approach was adopted. The privacy wishes of individual nodes can easily be enforced as they directly manage their own context data, however, complication is added to enforcement of user’s
privacy preferences as their context data is distributed across multiple nodes. The distribution of a user’s context data also requires the distribution of the user’s privacy policy to all nodes wishing to share the user’s context data. Any subsequent changes made to the privacy policy necessitate the updating of the previously distributed privacy policies.

**User Identification**

The registration of users at each environment they interact with is not appropriate as this overhead may deter users from interacting with the ubicomp services. Therefore, environments must be capable of automatically identifying users that enter their domain in order to discover some context data concerning that user.

One potential approach to the identification issue would be for the devices carried by the user to provide the ubicomp environment with URI that can be dereferenced to access context data concerning the user. However, such approaches introduce undesirable device restrictions; for instance, a mobile phone implementation would most likely require the installation of a particular application in order to share the URI.

The adopted approach uses a *lookup service* that contains mappings between a user’s identifier, including device identifiers, and the user’s dereferencable URI. This approach is flexible and allows multiple identifiers to be associated with a single user. The *lookup service*, described in detail in Section 4.3.2, is a peripheral service which can be bypassed if necessary, thus allowing the flexibility to use approaches, such as the previously mentioned device-restrictive one if necessary.

**Context Discovery**

With a user’s context data being dispersed over multiple nodes the locating of particular pieces of context data becomes more challenging than in a centralised approach. For instance, a node should be capable of finding out where a user was located at this time one week ago. Potential solutions to this issue must respect the control and ownership of context data.

The proposed solution to context discovery involves the use of a *home node*, which has complete knowledge of the location of all of a user’s context data. Nodes can identify a user’s *home node* thanks to an additional mapping, which is stored by the *lookup service*, that associates a user with a *home node*. By having access to an identifier for the user and the *home node* a third-party node has the potential to locate all context data concerning the user within the CxaaS network.

**Policy Enforcement**

The distributed approach adopted within CxaaS makes the enforcement of node policies easier, but the enforcement of user policies is made more complicated. It is not practical to require a user to create or provide a privacy policy each time they interact with an environment for the first time. For instance, a tourist visiting Dublin for the first time could
be presented with context-aware tourist information at various locations around the city. However, if the tourist was required to create a privacy policy for each environment they interacted with they may decide to ignore the available context aware functionality because of the inconvenience involved for a single interaction with the systems. Therefore, a solution is required that maximises privacy protection while attempting to minimise the interaction obstructions for the user.

To deal with this issue Cxaas makes the home node the central point from which a user’s privacy policies are distributed. The policies, or parts thereof, can be made available to other nodes when required.

4.3.2 Cxaas Network Components

Cxaas-Net comprises of the components within the Cxaas model that are network accessible, in particular the lookup service and the web-service interfaces provided by the Cxaas nodes. The following sub-sections elaborate on these components. Figure 4.3 provides an example of the interactions between the primary members of the Cxaas model. The sequence diagram shows a user entering an environment and being detected by an RFID scanner. The user’s RFID details are provided to the environment’s Cxaas node, which in turn sends a request to the lookup service to find out the user’s Cxaas URI and home node. The sequence continues with the node discovering the user’s last known location. This location data is not stored on the home node but, as the home node was recently informed of a new source of the user’s location data, the home node notifies the environment’s Cxaas node of a source of recent location data. Based on this location data a context-aware recommender service is provided. The sequence concludes with the Cxaas node informing the user’s home node of the user’s new location and requesting the user’s privacy policies.

Lookup Service

The lookup service stores information on Cxaas nodes, users and user’s identifiers. The lookup service enables mapping between a user’s real-world identifier, such as their RFID details, and a URI. Using the URI, services can search for the user’s home node mapping stored by the lookup service.

Table 4.4 outlines the operations supported by the Cxaas lookup service. There are three categories of operations and they are based around the three object classes stored within the lookup service: (i) Identifier Mappings, (ii) Users, and (iii) Cxaas Nodes (shown as Services). Figure 4.3 provides an example of interactions between a Cxaas node and a lookup service in order for the node to identify a new user and find out the location of their home node in order to discover additional information concerning the user.

It is possible that some minimal context information could be inferred from the home node associated with a particular user. For instance if the user’s home node has the URI http://acmecorp.com/cxaas it could be inferred that the user works for Acme Corp. However,
the amount of information that can be inferred is minimal and it is similar to the level of information that can be inferred from an email address, where the place of work can be inferred from the domain of the email address. This issue can be overcome through the use of ambiguous domain names in the home node URIs.

**User Registration:** After a CxaaS nodes register with the lookup service they can then register users. The registration process is typically handled by the home node. The inputs to the registration process are the URI of the user’s home node and the URI of the user on the home node. Ordinarily, the URI of the user is dereferencerable to a domain under the control of the home node, for instance the home node may be located at http://example.org while the URI of the user could be http://example.org/user/123#user. The registration of users cannot occur without consent from the user themselves. If an unregistered user enters a ubicomp environment then the environment must proceed without access to context data from third-parties.

The first set of sequences in Figure 4.4 show the communication sequence between a CxaaS node and a lookup service when registering a user. In the example provided only one identifier is associated with the user, for instance an RFID. Both the user registration and policy upload sequences of Figure 4.4 would have to occur prior to the sequence depicted in Figure 4.3 in order for the user’s remote context data to be discoverable.
User Identifiers: After user registration the lookup service is used to associate identifiers, such as their RFID tags, Bluetooth MAC Addresses or email addresses, with the user’s URI. Using this mapping, ubicomp environments can identify new and returning users and provide the user with context aware functionality. For instance, when a user enters a smart space and their RFID is detected, the smart space can find the user’s URI from the lookup service and they can also find out the user’s home node. Web-based identifiers, such as email, are useful for situations where a CxaaS-Node has access to context data originating from the web where the user may be identified by an email address. Requirement ‘R.1 Discovery’ is addressed by the aforementioned functionality.

Home Nodes and Node Interfaces

The node interfaces are the point at which nodes expose their, privacy-sensitive, context sharing functionality. The key interfaces are those associated with the exchange of context data and those associated with the exchange of privacy policies. Additionally, the functionality of home nodes are relevant to the operation of CxaaS-Net.

Home Nodes: Much of the functionality of the home nodes has been covered by the previous sections. In summary, the home node is viewed as the central point from which a user’s context data can be located and their privacy policies retrieved. In addition to this the home node it is typically the point where relatively static context data, such as calendar data, can be retrieved. The home node is explicitly chosen by the user and at any given time they may only have one home node. Users that do not have a home node cannot have their context data shared.
### Identifier Mappings

<table>
<thead>
<tr>
<th>Operation</th>
<th>Identifier Mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_mapping(identifier_type, user_URI)</td>
<td>identifier, identifier, user_URI. Create a new mapping between a user and a identifier, the URI of the new mapping is returned.</td>
</tr>
<tr>
<td>search_mappings(identifier_type, identifier)</td>
<td>Identify mappings to find an entry and return a user_URI where possible.</td>
</tr>
<tr>
<td>read_mappings(mapping_URI)</td>
<td>Read a particular mapping, specified by its URI.</td>
</tr>
<tr>
<td>update_mapping(identifier_type, old_user_URI, new_user_URI)</td>
<td>identifier, identifier, identifier. Update an existing mapping entry (a device owner may have changed) the method returns a true or false.</td>
</tr>
<tr>
<td>delete_mapping(identifier_type, user_URI)</td>
<td>identifier, identifier. Deletes a mapping between a user and an identifier based on the unique combination of identifier and identifier type.</td>
</tr>
<tr>
<td>delete_mapping(mapping_URI)</td>
<td>Deletes a mapping between a user and an identifier based on the URI of the mapping.</td>
</tr>
</tbody>
</table>

### Users

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_user(user_URI, home_node_URI)</td>
<td>Create an entry for a new user in the lookup service, providing it with the user’s home node and the user’s URI. The home node must have an associated services entry on the lookup service.</td>
</tr>
<tr>
<td>read_user(user_URI)</td>
<td>Read a user’s details in the lookup service, including details of their home node.</td>
</tr>
<tr>
<td>update_home_node(user_URI, new_home_node_URI)</td>
<td>Update the URI of the home node of a user. This action is used when a user wishes to change their home node.</td>
</tr>
<tr>
<td>delete_home_node(user_URI)</td>
<td>Delete the home node associated with a user.</td>
</tr>
<tr>
<td>delete_user(user_URI)</td>
<td>Delete the entry for a user on the lookup service.</td>
</tr>
</tbody>
</table>

### Services

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_service(service_URI, WSDL_location, name)</td>
<td>Create a service entry in the lookup service. The parameters provided are the URI of the service, the location of its WSDL file and its name.</td>
</tr>
<tr>
<td>read_service(service_URI)</td>
<td>Read the service’s details from the lookup service.</td>
</tr>
<tr>
<td>update_service(service_URI, new_WSDL_location, new_name)</td>
<td>Update the service’s WSDL location and/or its name.</td>
</tr>
<tr>
<td>delete_service(service_URI)</td>
<td>Delete the service’s details on the lookup service.</td>
</tr>
</tbody>
</table>

Table 4.4: Lookup Service Operations
Chapter 4. Context as a Service and the Network Model

Figure 4.3 provides an example of the purpose of the home node within the CxaaS model. In the diagram an ubicomp environment wishes to discover the last known location of a user that has just entered it’s domain. Once the user’s home node has been identified, via the lookup service, the environment then sends a request to the home node for the required information. There are four possible responses to the request: (i) the home node has the requested information and returns it, (ii) the home node does not have the requested information but redirects the request to a node that does, (iii) the home node does not have the requested information or a node that does, and (iv) the user’s privacy policy forbids the returning of the context data. In the situation depicted the home node redirects the request to a third-party node.

The final interaction with the home node depicted in Figure 4.3 shows the original CxaaS node informing the home node of the location of its user. In future, the home node will redirect appropriate requests to that node. In order for the node to share the context data it gathers concerning that user it requests a copy of the user’s privacy policy.

**Privacy Policy Interfaces:** At some point after registration the user must provide a privacy policy, represented using RIF, to their home node in order for their context data to be sharable. This privacy policy, or the relevant parts of the policy, will be provided to remote nodes that wish to share context data concerning the user.

The CxaaS-Net privacy policy functionality is outlined on the lower part of Table 4.5. As previously mentioned there are three types of policies facilitated within the CxaaS Model: (i) User Policies, (ii) Node Policies and (iii) Legal Policies. However, the CxaaS-Net is solely concerned with User Policy as it is the only policy that is shared between nodes. The privacy policy involves both the CxaaS-Net and the CxaaS-Node. The CxaaS-Net provides the interfaces for the sharing of the privacy policies, while the CxaaS-Node controls the sharing and enforcement of the privacy policies.

**Core Context Interfaces:** The CxaaS-Node core functionality provides five methods for handling context data, as show on the upper part of Table 4.5. These methods represent the minimum set of operations necessary to handle context data within CxaaS. If required by specific deployments, the CxaaS Model can be extended to provide additional functionality that is not possible with the core functionality alone.

The distribution of context data throughout the network is not completely unrestricted. A node that receives context data from another node should not re-distribute that context data. This constraint reduces the number of policy types that must be enforced in a distributed fashion without reducing the functional capabilities of the model. Without this constraint the original node’s node policy would also have to be distributed to ensure its privacy wishes were adhered to when the secondary sharing was taking place.
4.3. CxaaS Network

### Operation Description

#### Context Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_context(user_URI, context_subject, context_type, value)</td>
<td>This operation enables the creation of a context triple for a particular user. This can be used by local context services or by remote CxaaS nodes to inform a home node of the location of new context data.</td>
</tr>
<tr>
<td>read_context(context_URI)</td>
<td>Retrieve the context data from a previously known URI.</td>
</tr>
<tr>
<td>update_context(context_URI, context_type, value)</td>
<td>Update the value of an existing piece of context data. The user associated with the context or the URI of the context cannot be changed.</td>
</tr>
<tr>
<td>delete_context(context_URI, delete_inferred)</td>
<td>Delete a piece of context data from a node. If the ‘delete_inferred’ boolean is set to true then any data inferred from the data to be deleted will also be deleted.</td>
</tr>
<tr>
<td>search_context(user_URI, context_type, parameter(s))</td>
<td>Search the a particular type of context data, for a particular user and according to some parameters.</td>
</tr>
</tbody>
</table>

#### Policy Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_policy(user_URI, policy)</td>
<td>Create a privacy policy for a user.</td>
</tr>
<tr>
<td>read_policy(user_URI)</td>
<td>Retrieve the the privacy policy of a particular user.</td>
</tr>
<tr>
<td>update_policy(user_URI, new_policy)</td>
<td>Update the privacy policy of a user.</td>
</tr>
<tr>
<td>delete_policy(user_URI)</td>
<td>Delete the privacy policy of a user.</td>
</tr>
</tbody>
</table>

Table 4.5: CxaaS Node Core Operations

---

4.3.3 CxaaS-Net Component Interaction Examples

This section contains two interaction examples oriented around CxaaS-Net’s identification and discovery functionality.

State Transition Example

In order to illustrate the operation of CxaaS-Net this section provides details of a typical program flow that occurs when a user enters a ubicomp environment. Figure 4.5 illustrates the state transitions that occur when a user enters a ubicomp environment and attempts to discover some context data about that user.

The diagram begins with a user entering the environment, after which a check is made to see if the user is a new user or a returning user. If it is a returning user then a copy of the
user’s CxaaS URI and home node details will be available locally, otherwise the CxaaS node will have to check if the user is registered with the lookup service, and if so, retrieve their CxaaS URI and home node.

Once the home node is known, the CxaaS node sends a request to it for the context data it requires. If the context data is not available on the home node but it is available on a third-party node then then home node redirects the request to the third-party. Each piece of context data will only be accessible from a single node, however if a range of context data is requested then several nodes may have to be queried. Finally, the requested context data is returned in accordance with the user’s privacy policy. In this example it is assumed that the user’s privacy policies have previously been provided to the third-party node, in order for it to be able to share the user’s context data.

Figure 4.5: User Entering Smart Space State Transition Diagram
Use Case Example

Figure 4.2 provided a high-level overview of the CxaaS Network Architecture, while Figure 4.6 provides a related use case diagram which focuses on the identification of a user and the discovery of context data concerning that user. The use case diagram provides a typical example of the functionality that CxaaS is designed to provide.

In the diagram the ‘Ubicomp Service’ is a service running within the CxaaS Node’s local ubicomp environment. It wishes to provide a context aware service to the user, and in order to do so it requests context data from the CxaaS node concerning an as yet unknown user with a particular identifier. The CxaaS node initially identifies the user and its home node either by using existing user details or, if it is a new user, the lookup service is used. If the requested context data is not available locally the node requests it from the user’s home node.

If the home node has the requested context data then it will return it, otherwise, if it is available elsewhere then the request will be redirected to a third-party node. Both the returning of locally stored information and the redirection to a third-party node are executed subject to approval by the user’s privacy policy. Therefore, if the user’s privacy policy forbids the requester accessing that particular type of context data, then the home node will not redirect the request as the redirection alone could provide some context information concerning the user. For instance, if the user’s last location was requested and the request was redirected to a node controlled by a particular shop then it is quite likely that the user was last located in that shop.

4.3.4 CxaaS-Net Requirements Mapping

Table 4.6 illustrates CxaaS-Net’s contributions to the core and functional requirements. This table is provided, and added to, at the end of each of the core chapters in order to summarise which of the requirements are covered by what parts of the CxaaS model. The blank cells that currently exist in the table will be covered by subsequent chapters. While certain requirements may involve more than one part of CxaaS, a significant contribution to
the fulfilment of the requirement is required in order for the contribution to be added to the table.

Of the core requirements, CxaaS-Net is designed to contribute towards Requirement \textit{R.1 (Discover)} and the ‘sharing’ part of \textit{R.2 (Controlled sharing of context data)}. The ‘controlled’ part of requirement \textit{R.2} involves privacy policies, the structure of which are handled by the data model and the execution of the privacy policies is a task for individual CxaaS nodes.

In terms of the functional requirements CxaaS-Net deals with \textit{FR.1.1 Interfaces for the provision of context data to third-parties} but not with the formal description of the interfaces, this is handled by the data model. \textit{FR.2 Discovery and Mobility} and related sub-requirements are included within CxaaS-Net.

\begin{table}[h]
\centering
\begin{tabular}{ |c|c| }
\hline
\textbf{Requirement} & \textbf{CxaaS-Net} \\
\hline
\textbf{Core Requirements} & \\
R.1 Discovery & • \\
R.2 Controlled Sharing of Context Data & • \\
R.3 Formal Context Model & \\
\hline
\textbf{Functional Requirements} & \\
FR.1 Exchange of Context Data & \\
FR.1.1 Interfaces for the provision of context data to third-parties & • \\
FR.1.2 Formally described interfaces & \\
FR.2 Discovery and Mobility & \\
FR.2.1 Identification of new and returning users & • \\
FR.2.2 Discovery of context data about identified users & • \\
FR.2.3 Multiple identification mechanisms & • \\
FR.3 Privacy Protection & \\
FR.3.1 Distribution of Privacy Policies & \\
FR.3.2 Expressivity and Granularity & \\
FR.3.3 Multiple Policy Types & \\
FR.4 Individual Nodes & \\
FR.4.1 Implementation Agnostic & \\
FR.4.2 Manage the entire lifecycle & \\
FR.4.3 Policy Enforcement & • = central involvement, \circ = partial involvement \\
\hline
\end{tabular}
\caption{CxaaS Requirements Support}
\end{table}

78
4.4 Summary

This chapter began with a high-level description of the approach adopted, called Context as a Service (CxaaS), for the controlled sharing of context data between heterogeneous third-parties. Based on the adopted approach, and the deficiencies in the related work identified in the previous chapter, a set of requirements are defined. They consist of a set of three core requirements and a related set of functional requirements.

The chapter continues by introducing CxaaS and its three components: (i) the network model (CxaaS-Net) is focused on the discovery and sharing of context data between nodes; (ii) the node model (CxaaS-Node) is concerned with the management of context data across its entire lifecycle, servicing its local ubicomp environment and enforcement of the privacy requirements; and (iii) the data model (CxaaS-Data) ensures the understanding of the exchanged context data, the interfaces provided by the nodes and the privacy policies.

The remainder of the chapter provides the details of CxaaS-Net. The issues being addressed by CxaaS-Net are discussed initially and following this the functionalities provided to address those issues are outlined. Of particular importance to CxaaS-Net is the lookup service and its operation is described within the functionality section.
Chapter 5

CxaaS-Node

CxaaS nodes are the primary point at which users and applications interact with the CxaaS Network. The nodes have two parts: (i) the external, network accessible, that is comprised of the service descriptions and service interfaces, (ii) the internal part provides the functionality behind the interfaces for the management of context data and the dissemination of privacy policies. While the previous chapter discussed the external part, this chapter focuses on the internal portion of the nodes.

The core purpose of the node architecture is the privacy-sensitive management of the entire lifecycle of the context data under its control. In addition to the management of context data, the enforcement of privacy policies are of key importance within the node architecture. Three types of privacy policies are facilitated, user policies, legal policies and node policies, a policy execution component controls the execution of the policies. The context store allows nodes and applications access to historical context data.

This chapter describes the architecture and models underpinning the CxaaS nodes. Initially a lifecycle for the provision of CxaaS is developed to ensure consistent handling of context data across all nodes. Following this, the functionality of the CxaaS nodes are described. Finally, the architecture of the individual nodes, which encompasses the lifecycle and how they relate to the requirements derived in Section 4.1 is outlined.

5.1 A Lifecycle for the Provision of CxaaS

The realization of CxaaS requires the encapsulation of the context lifecycle within a web-service. In order for this to occur a formal model for the lifecycle of context data within a CxaaS node must be developed. The lifecycle presented in this section is based upon related work on enterprise data lifecycles and ubiquitous computing lifecycles. While there exists some related lifecycle work that can be leveraged in the development of the CxaaS lifecycle there is no individual lifecycle that is suited to adoption within the CxaaS model. This section analyses the existing related lifecycles, which is divided into two classifications of lifecycles: (i) Enterprise Lifecycle approaches and (ii) Context Lifecycle approaches.
Several lifecycle diagrams are presented as part of the analysis of the related work and the presentation of the CxaaS lifecycle. These diagrams provide multiple stages linked by arrows that go in a clockwise direction. These arrows, and the order of the stages, represent the typical order in which the stages progress throughout the life of the data. However, strict sequential execution cannot be assumed to occur for each data item, for instance, steps may be omitted in particular situations.

5.1.1 Existing Lifecycles

Enterprise Lifecycles

Enterprise Lifecycles are developed for enterprise applications and do not explicitly target context data. In comparison to context data, the data stored within the enterprise applications is relatively static. However, these lifecycles are robust and well-established industry standard strategies and are formulated to control the lifecycle of sensitive data in enterprise applications. While the lifecycles in their entireties are not suitable for application within a CxaaS node, certain lifecycle stages are suitable for reuse.

Two of the more popular enterprise lifecycle approaches are Information Lifecycle Management (ILM) (Peterson and St.Pierre, 2004) and Enterprise Content Management (ECM) (AIIM, 2010), both of which are shown in Figure 5.1. Each approach consists of more than just a lifecycle, but also the tools, practices and methods that surround the lifecycle. ILM is defined by the SNIA Information Lifecycle Management Initiative to consist of “the policies, processes, practices, services and tools used to align the business value of information with the most appropriate and cost-effective infrastructure from the time information is created through its final disposition” (Peterson and St.Pierre, 2004). While ECM is said to be “the strategies, methods and tools used to capture, manage, store, preserve, and deliver content and documents related to organizational processes” (AIIM, 2010). The details of the steps in the ILM lifecycle are provided in Table 5.1, while the details of the ECM lifecycle are provided in Table 5.2.
5.1. A Lifecycle for the Provision of CxxaaS

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation and</td>
<td>This stage can involve the local creation of business records or the receipt of the records from an external source. The exact type of business record is not specified as the lifecycle applies to all types of business records.</td>
</tr>
<tr>
<td>Receipt</td>
<td></td>
</tr>
<tr>
<td>Distribution</td>
<td>Once the information has been gathered it is distributed to the point where it is required. This distribution can be internal or external to the organisation.</td>
</tr>
<tr>
<td>Use</td>
<td>This step applies to information that has been distributed internally. The information can be used for a wide range of business purposes.</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Maintenance concerns the management of the information after it has been used. This includes the archiving of the information in such a manner that it can be efficiently retrieved.</td>
</tr>
<tr>
<td>Disposition</td>
<td>In general, the period directly after the creation of a piece of information is when it is most valuable to the business. After this period expires its utility gradually declines and the information will be disposed of once its retention period has lapsed. The length of the retention period will vary from organisation to organisation and should incorporate the organisation’s legal requirements.</td>
</tr>
</tbody>
</table>

Table 5.1: ILM Lifecycle Stage Descriptions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capture</td>
<td>Capture involves the collection of data from disparate sources and producing a consistently structured data for the management step. The data can include scanned paper documents, electronic files.</td>
</tr>
<tr>
<td>Manage</td>
<td>The management step incorporates the functionality for organisation, searching, retrieval and versioning of the captured data.</td>
</tr>
<tr>
<td>Store</td>
<td>The store step concerns the temporary storage of data that is not currently suitable for long-term storage. Long-term storage is handled by the ‘preserve’ step and is separate from the ‘store’ step.</td>
</tr>
<tr>
<td>Preserve</td>
<td>Data that is suitable for long-term storage is processed in this step. A primary use-case for this step is the long-term storage of data in order to comply with statuary data-retention requirements.</td>
</tr>
<tr>
<td>Deliver</td>
<td>The ECM lifecycle is not require a strict sequential following of the lifecycle steps. Therefore, the deliver step can provide data from the any of the three previous steps (Manage, Store and Preserve).</td>
</tr>
</tbody>
</table>

Table 5.2: ECM Lifecycle Stage Descriptions
A third lifecycle in the enterprise domain, developed by Hayden (Hayden, 2008), provides an extensive data lifecycle management strategy containing ten steps. Hayden strikes a balance between ICM and ECM by having a better distribution of steps before and after the Release of the data. Some of Hayden’s steps, shown in Figure 5.2 and described in Table 5.3, are of relevance to the CxaaS lifecycle. Hayden places particular emphasis on the security and integrity of the business data within each stage of the lifecycle.

**Context Lifecycles**

Within the ubicomp domain many papers mention the lifecycle of context data, however, very few provide an actual lifecycle. Three context lifecycles have been identified and analysed, each of which takes a different approach depending on the application requirements. The details of the stages for each of the three lifecycles are provided in Table 5.4. The first lifecycle is by Chantzara and Anagnostou (Chantzara and Anagnostou, 2005), they have identified four separate stages in the lifecycle, which is shown on the left of Figure 5.3. The stages are distributed across three types of actors; Context Providers, a Context Broker and context aware Service Providers. This approach is of relevance to the CxaaS lifecycle as the interacting actors in both systems are similar.

Ferscha et. al. (Ferscha et al., 2005) provide details of their lifecycle for context information in wireless networks, shown on the right of Figure 5.3. They begin with the Context Sensing stage, which can be a time triggered or event triggered acquisition of low level context data, followed by the Context Transformation stage, which involves the aggregation and interpretation of the sensed information, the next stage is Context Representation, whereby the transformed context is applied to an ontology to create high level contextual information, Context Rules are then applied to the semantic context data which then sends commands to the Actuation stage.
5.1. A Lifecycle for the Provision of CxaaS

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>The lifecycle begins by selective collecting of the data required within the organisation’s business process. An effort is made to avoid the collection of unnecessary sensitive data, such as the collection of social security numbers or credit card numbers if they are not required.</td>
</tr>
<tr>
<td>Relevance</td>
<td>Following the collection of the data its relevance to the business process is established. Non-critical data is dropped at this point in order to minimise the data loss if a security breach occurred.</td>
</tr>
<tr>
<td>Classification</td>
<td>The classification stage sorts the data according to the following categories: (i) level of business sensitivity, (ii) if it is personally identifiable information, (iii) sensitive personal information (such as health records), and (iv) whether the information is publicly available or not.</td>
</tr>
<tr>
<td>Handling and Storage</td>
<td>This step ensures that the data is appropriately stored and handled, in accordance with its classifications, as it progresses through the lifecycle stages.</td>
</tr>
<tr>
<td>Transmission and</td>
<td>Incorporating both the physical and digital movement of data, this stage is designed to ensure that the most appropriate transmission and transportation mechanisms are used for the data as it progresses through the lifecycle stages. The classification of the data will effect the choice of the most appropriate mechanism.</td>
</tr>
<tr>
<td>Transportation</td>
<td></td>
</tr>
<tr>
<td>Manipulation and</td>
<td>At this point in the lifecycle the data is prepared for release to a third-party. Up to this point the data is predominantly unchanged from the initial collection stage. The exact details of this stage will vary between organisations.</td>
</tr>
<tr>
<td>Conversion/Alteration</td>
<td></td>
</tr>
<tr>
<td>Release</td>
<td>The release stage is the point where the manipulated data is released to a third-party. There are strict security requirements surrounding the release of the data to third-parties.</td>
</tr>
<tr>
<td>Backup</td>
<td>Backup involves the copying of data to a separate location so that the data copies can be used to restore the original data in the event of a data loss.</td>
</tr>
<tr>
<td>Retention</td>
<td>The retention stage of the lifecycle concerns the statutory requirements for businesses to retain data for certain periods of time. These requirements are enforced in this stage.</td>
</tr>
<tr>
<td>Destruction</td>
<td>The destruction of data involves the deletion of digital information once it is no longer of use and the retention requirements have been met. Destruction also applies to physical copies of data.</td>
</tr>
</tbody>
</table>

Table 5.3: Hayden’s Lifecycle Stage Descriptions
As part of the MOSQUITO project Wrona et. al. [Wrona and Gomez 2005] developed a simple three stage context life cycle which was made up of “Context Information Discovery”, “Context Information Acquisition”, “Context Information Reasoning”. The Context Information Acquisition stage is applicable to the CxaaS lifecycle as it is more generic than the Context Sensing stages used by the other lifecycles and hence applies to pushing/pulling of context data from many different sources, not only sensors.

Lifecycle Analysis

The context lifecycles are oriented around their particular applications and not optimal for use within CxaaS. There is no reference to the privacy protection of the context data, which is a fundamental requirement for CxaaS. While the enterprise lifecycles do contain significant security measures they do not offer any privacy protection functionality to the subjects of their data. For their types of deployments such functionality is not necessary, however this is not the case for CxaaS.

When compared with the enterprise lifecycles, it is clear that the context lifecycles do not provide any stages for managing the context data after it has been used. Context data can be highly sensitive and therefore its deletion/destruction/disposition should be an important part of its lifecycle.

The enterprise lifecycles are optimised for handling sensitive business data and therefore are not directly applicable to use within CxaaS, while the context lifecycles have some deficiencies that also make them unsuitable. Therefore, it is necessary to develop a new lifecycle which is optimised for distributed context sharing applications. While the lifecycles in their entirety are not suited for adoption, particular aspects of the lifecycles are applicable.

5.1.2 CxaaS Lifecycle

The CxaaS lifecycle, shown in Figure 5.4, borrows aspects from the previously described lifecycles. The result is a formal context management lifecycle optimized for deployment in
## 5.1. A Lifecycle for the Provision of Cxaas

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chantzara and Anagnostou’s Lifecycle</strong></td>
<td></td>
</tr>
<tr>
<td>Sense</td>
<td>Sensing consists of the acquisition of context from context information sources e.g. sensors or network resources</td>
</tr>
<tr>
<td>Process</td>
<td>Context processing includes the generation of high-level context information from primitive data by enforcing functions of interpretation, filtering, aggregation and inference.</td>
</tr>
<tr>
<td>Disseminate</td>
<td>Context dissemination refers to the efficient distribution of context to context-aware services</td>
</tr>
<tr>
<td>Use</td>
<td>Context usage describes the utilization of context by context aware services in order to trigger the appropriate actions</td>
</tr>
<tr>
<td><strong>Ferscha’s Lifecycle</strong></td>
<td></td>
</tr>
<tr>
<td>Sensing</td>
<td>Time or event triggered acquisition of low level context information</td>
</tr>
<tr>
<td>Transformation</td>
<td>Aggregation or interpretation of low level context information</td>
</tr>
<tr>
<td>Representation</td>
<td>The application of data structures representing abstract context information</td>
</tr>
<tr>
<td>Rule Base</td>
<td>The application of implicit or explicit rules for the triggering of control events</td>
</tr>
<tr>
<td>Actuation</td>
<td>Triggering context-based the actuation of the environmental actuators</td>
</tr>
<tr>
<td><strong>MOSQUITO Lifecycle</strong></td>
<td></td>
</tr>
<tr>
<td>Discovery</td>
<td>A context-aware system discovers available context information providers. The discovery can be performed either in a push or a pull mode.</td>
</tr>
<tr>
<td>Acquisition</td>
<td>A context-aware system collects context information from the discovered context information providers and stores it in a context information repository for further reasoning. Similar to the process of discovery, the acquisition is performed either in a pull or a push mode.</td>
</tr>
<tr>
<td>Reasoning</td>
<td>Reasoning mechanisms enable applications to take advantage of the available context information. The reasoning can be performed based on a single piece of context information or on a collection of such information.</td>
</tr>
</tbody>
</table>

Table 5.4: Context Lifecycle Stage Descriptions
a service-based environment. The lifecycle defines the stages that a typical piece of context data within the CxaaS network progresses through from the time it is initially created to when it is deleted. The arrows between the stages represent the typical execution order of the stages, however it is possible that for particular pieces of context they may be executed in a different order or particular stages omitted. The lifecycle has been separated between three actors in a similar manner to Chantzara and Anagnostou. The remainder of this section elaborates upon the actors and stages of the CxaaS lifecycle.

**Context Provider:** Context providers are sources of context data, including, but not limited to, sensors, web services, mobile phones and other CxaaS nodes. The context sensing and context transmission stages within the context provider are not part of the CxaaS lifecycle. The two stages provide a minimal representation of the stages the context is subjected to by the context provider and may be replaced by entirely different stages depending on the details of the context provider. For instance, if the context provider is another CxaaS node then the two stages will be replaced by an entire CxaaS lifecycle. Context sensing is a generic stage involving the retrieval of context data. Following this the context prepared for, and transmitted, in the context transmission stage. The context data can be made available to the CxaaS node via an interface from which it can be pulled or it can be pushed to the node.

**Context Consumer:** The context consumer can be any context aware service, such as a mobile phone application, a web application, a traditional ubicomp environment, a desktop application or another CxaaS node. Figure 1.1 of Chapter 1 provides a high level overview of some of the potential context providers and consumers interacting with a CxaaS node. As with the context provider, the lifecycle stages within the context consumer are a minimal representation of the lifecycle steps the context data is subjected to within the consumer.

**Context Acquisition:** The context acquisition stage, which is inspired by MOSQUITO, provides mechanisms for the pulling and pushing of context data. By allowing context data to be pushed and pulled it prevents the lifecycle from imposing any limitations on how the context data is acquired. For instance pushed data could either be manually entered via the web-interface or pushed via an interface, while calendar event data could be pulled at regular intervals to check for updates. Security must be incorporated into the acquisition stage as it may not be appropriate to allow all/any third-parties to create context data.

The acquisition stage begins with the retrieval of the context data. This can take two forms: (i) the provision of interfaces for the context to be pushed to, or (ii) the pulling of context data. Once the context has been retrieved, the security permissions of the context provider are checked to ensure they are permitted to provide context data to the node. Following this, the context data is passed on to the classification stage.
5.1. A Lifecycle for the Provision of CxaaS

**Context Classification:** The context classification stage is based on the *representation* stage from Chantzara and Anagnostou’s lifecycle. It is of interest to the CxaaS lifecycle as classification of acquired context data is important due to the variety of context types and the importance of knowing the relationship between individual pieces of context data. The *classification* stage classifies unstructured data according to a schema or ontology. Inputted structured context data may be reclassified according to a new schema or ontology if necessary. For example, if a user’s profile is available on Facebook and it is represented using a particular schema it may require conversion to a separate schema if the original one is not supported.

Once the context data arrives at the classification stage the appropriate schema is applied to the context data. If a schema cannot be found then context data will not progress to the handling or dissemination stages as the semantics of the context data may not be understood by the consumer. Upon application of an appropriate classification the context data is provided to the handling stage.

![Figure 5.4: The CxaaS Lifecycle](image-url)
Chapter 5. Cxaas-Node

**Context Handling:** The *handling* stage expands on the classified data to infer new context data from the original data and subsequently manages the storage of that data. For instance, upon finding out that John Smith is located in Brooklyn it should be possible to infer that John is also located in New York City, New York State and the United States. The additional information created in this handling stage can be used within each of the subsequent steps, for instance when specifying context access permissions used within the dissemination phase. The additional context data is materialised prior to receipt of a query for the context as the context data is relatively static and provides faster query processing than a query-rewriting approach. The handling stage is of particular relevance to privacy policies as the extra data inferred in this step can be used in the privacy policies to permit access of different levels of accuracy. For instance, access to a user’s location data could be restricted to levels of accuracy coarser than that of a city.

The storage of context data is required in order for context to be available for dissemination when requested. The storage of the context data occurs at the end handling stage.

The handling stage begins by analysing the classification of the context data. If possible, and appropriate, more context data can be inferred from the newly classified context data. This new context data will have an associated classification and will join the lifecycle at this point. For context data that does not result in the inferring of new context, this stage does not result in any changes. Upon completion of this stage the context data can be stored until such point as it is ready for dissemination, or immediately disseminated to appropriate context consumers.

**Controlled Dissemination:** Controlled *dissemination* is inspired by the ‘disseminate’ stage from Chantzara and Anagnostou. This stage is the point where the privacy policies are enforced prior to the release of data. If the appropriate user policies are not available then they must be retrieved from the user’s home node.

The dissemination of context data to the context consumer begins with the receipt of a request for context data. The identification of the consumer is then established and their permissions defined in the privacy policies are checked. The appropriate context data is retrieved from the context store and returned to the consumer.

**Context Maintenance:** The *maintenance* stage is similar to the *maintenance* stage in ILM, the *preserve* stage from ECM, and the *backup* and *retention* parts of from Hayden’s lifecycle. It is not possible to assume, as is the case with the context lifecycles, that once a piece of context data has been used that it is no longer of use. The maintenance stage consists of context data archiving and backup.

The backup of context data involves the copying of context data to a separate location in order to have a copy of the data available in situations where the original data is lost. The backup is done on the data placed in the context store by the handling stage.

Context data of limited utility to the Cxaas node can be archived in order to free resources.
This archived data should still be accessible to context consumers, however, it is possible that the queries on this data will not be as fast as the non-archived data. The archival of context data may be of particular use to nodes operating in jurisdictions that have data retention requirements.

**Context Disposition:** The *disposition* stages are based on the equivalent stage from ILM. Once the context data is no longer of use, and there is no longer a legal requirement to retain the data, the data can be deleted. Deletion can be an automatic process that occurs after a specified period of time, or it can be manually initiated by the context owner. It can occur directly after the classification stage if the context data cannot be handled, or after the handling stage if the data is not used, or after the dissemination stage if there is no maintenance requirement, or finally, after the maintenance stage.

### 5.2 Node Architecture

This section provides details of the node architecture and how it relates to the lifecycle developed in the previous section. To begin with, the element categories are summarised and subsequently, a description of how the architecture supports the node’s local context services is provided. Each of the elements within the architecture is then described and details of the element interactions are provided. Afterwards, the relationship between the elements and the lifecycle is discussed.

#### 5.2.1 Overview

The node architecture, depicted in Figure 5.5, defines the core elements of a Cxaas node. The Cxaas Lifecycle stages are mapped to the node elements, as shown in Figure 5.10. The node architecture is implementation agnostic and, therefore, provides developers with implementation flexibility regarding their choice of language and hardware.

**Element Categories**

There are five main types of elements within the Cxaas architecture:

- **Service Descriptions** provide a formal description for the services presented by the node.
- **Service Interfaces** are the point of interaction for third-party nodes and they are described by the service descriptions.
- **Components** provide the functionality behind the interfaces.
- **Privacy and Security** deal with the enforcement of the privacy policies and the node’s security requirements.
- **Context Store** is the point where historical context data is stored.
While each of the elements will be described in this chapter, detailed information on particular categories is available elsewhere in the thesis. The Service Descriptions and the privacy policies are part of the CxaaS data model and will be dealt with in Chapters 6 and 7 respectively. The Service Interfaces, while mentioned in this chapter, are described in detail in Chapter 4. The implementations and context store will be described later in this section.

![Figure 5.5: CxaaS Node Architecture](image)

### 5.2.2 Service Interfaces and Descriptions

The service interfaces and descriptions are described in detail elsewhere (interfaces in Chapter 4 and service descriptions in Chapter 6), however, a summary is provided in this section to illustrate how they fit into the node architecture. The service descriptions, as shown with the interfaces in Figure 5.5, describes the services offered by a CxaaS node, what the relevant interfaces are and how to invoke them. WSDL 2.0 is used to describe the interfaces and SA-WSDL is used to provide the semantics of the interfaces and the inputted and outputted data. The core CxaaS functionality is accessible to third-parties via the Core Interfaces, the application specific functionality is accessible via separate interfaces.

The core interfaces represent the minimum functionality required to manage lifecycle of the context data, including the exchange of context data with third-party nodes, and the exchange of privacy policies. They are based around the CRUD operations (Create, Read, Update, Delete), which are the four basic operations of persistent storage. Each interface is mapped onto a component that is in charge of performing the action associated with
the interface. There are no restrictions in terms of how the core web-service interfaces are implemented, provided they are adequately formalised in the service description section.

5.2.3 Components

The CxaaS node architecture, show in Figure 5.5 includes six core components. Two of the core components are associated with the core interfaces (CRUD and Policy Manager), three are associated with a step in the CxaaS Lifecycle (Classification, Handling and Maintenance), while the final step, Mobility Manager, is associated with the context discovery functionality within the CxaaS network. The remainder of this section describes each of the components in detail.

CRUD

The CRUD component handles the Creation, Reading, Updating and Deletion of context data. Each of the CRUD actions is executed in a privacy sensitive manner according to the conditions provided by the Privacy and Security elements.

Figures 5.6, 5.7, 5.8 and 5.9 provide examples of interactions between components during the creation, reading, updating and deletion of context data respectively. These interactions will be explained in detail in Section 5.2.5. Even without a detailed explanation it can be
Chapter 5. CxaAS-Node

seen from the diagrams that the CRUD component orchestrates the execution of the create, read, update and delete requests within the node.

The CRUD portion of the core interfaces provides the CRUD component with context data from authenticated sources. For the creation of context data, shown in Figure 5.6 the context data is provided to the classification stage and returned with associated schema information. The CRUD component then provides the classified context to the handling stage, which returns a URI for the stored context data, which is then returned to the interface. When the new piece of context data is a new location for a user, as is the case in the example in the diagram, the mobility monitor component must be informed.

The deletion and updating of context data follows a similar sequence to the creation of context data. However, as the initial input to the interface contains the URI of a classified piece of context data the classification stage is not required. The updating and deleting is handled by the handling component.

For the reading of context data, shown in Figure 5.7, the CRUD component receives the URI of the required context data. The appropriate access permissions of the requester are established and returned by the Policy Execution element. Based on these access permissions the appropriate context data is retrieved by the CRUD component from the context store and returned to the interface.

Classification

In this component raw context data received by the CxaAS node is classified according to the schema used by the node. Additionally, context data originating from a third-party and modelled according to a different schema can be transformed by this component into the schema used by the node.

The classification component provides the functionality behind the classification stage in the lifecycle. It is called by the CRUD component when it receives a context creation request, as shown in Figure 5.6. When raw context data is received by the CRUD component the classification component is called and associates the context data with some concepts from an ontology such as the CxaAS Ontology, which will be described in the following chapter. The classification component also is charged with the transformation between data types, such as the transformation between coordinate systems or units of measure.

Handling

The handling component performs two primary tasks: (i) the inferring of new knowledge from new context data, as is described in the handling stage of the lifecycle, and (ii) management of the storage of the context data. Handling is involved in the creation, updating and deletion of context data.

For the creation of context data, the handling process takes the classified context data and, if appropriate, infers new context data from it, and then stores that context data in the
context store. The approach adopted for the inferring of new context data is not restricted and developers could choose a rules-based approach or develop some custom inferencing.

The updating of context data in the handling process must also take into account the previously inferred context data, and where appropriate this additional information should be applied to the inferred context data also. The deletion of context data is also managed by the handling component. The context data and its related inferred context can be deleted individually or as a whole.

![Sequence Diagram for a Context Read Event on a CxaaS Node](image)

**Figure 5.7: Sequence Diagram for a Context Read Event on a CxaaS Node**

**Mobility Monitor**

The movement of users between smart spaces is managed by the Mobility Monitor component. This involves the identification of new users through the use of the lookup service and the informing of users' home nodes when context concerning their users has been created.

When a person moves from one smart space to another that person’s home node is notified and thus any node attempting to locate that person can query their primary node. Figure 5.6 provides an example of when the mobility monitor is used. The initial invocation of the mobility monitor is to establish what the URI of the detected user is. In the case in the figure the mapping between the user’s identifier and its URI was already known and there was no need to access the lookup service.

The second time the mobility monitor is called in the figure is to inform the user’s home node of the location of new context data concerning their user. This is done through the
creation of a new piece of context data on the home node of the same type as the original piece of context data. The new piece of context data can link to the original using the ‘seeContext’ ontological property, or link to the node containing the original context data through the use of the ‘seeNode’ property. Both of these ontological properties will be discussed in further detail in Section 6.3.9 of the following chapter.

Upon informing of the user’s new source of context data the mobility monitor calls the policy manager. This is done to ensure the node has a copy of the user’s privacy policy.

Maintenance

The maintenance component performs the functionality outlined in the maintenance stage of the CxaaS lifecycle. This consists of the archival of historical context data of limited utility, the backing up of context data and the adherence to the data retention rules that apply to that node’s jurisdiction. The backup, archival and retention requirements will vary between nodes.

Policy Manager

The policy manager is in charge of the management of all aspects of the privacy policies, including the implementation of the logic behind the policy exchange interface and the dissemination of privacy policies to remote nodes when appropriate.

If a request is received for a privacy policy the policy manager can return the entire policy or the parts of the policy that relate to the context data store by the remote node. The privacy policies are defined using rules, which will be described in detail in Chapter 7 and if desired only the rules relevant to the remote node may be returned.

The policy manager is also tasked with the retrieval of policies from remote nodes. In the example outlined in Figure 5.6 the detected user’s location is updated on its home node by the mobility monitor and then the policy manager is called to establish whether the privacy policies must be retrieved from the user’s home node. If the node already has a copy of the user’s policy then this will not be necessary. Additionally, the policy manager must periodically check to see if its current copies of the privacy policies are up to date.

5.2.4 Privacy and Security

Prior to the execution of the component associated with the requested interface the requester must be identified and authenticated. From this it can be established if a local application or service is making the request or if it is a third-party. Local requests do not have the same level of privacy restrictions as remote requests as the local service is under the control of the same individual or organisation as the CxaaS node, thus the context data is not being shared with third-parties. Based on the identity of the user the Policy Execution component decides which of the three policy types are applicable to the request. For instance
if a user requests their own data it is not necessary to apply the User Policy, if a local service requests context data it may not be necessary to apply the Node Policies.

The node architecture supports three types of privacy policies: (i) user, (ii) node, and (iii) legal. The privacy policies contains the set of conditions under which the user/node/law is prepared to share context data and at what level of granularity. Each of the policies must be successfully executed in order for the context data to be shared. Details of the privacy policies and their structure will be covered in Chapter 7.

Privacy Policy Dissemination

The dissemination of the privacy policies is a task which the CxaaS-Node portion of the architecture fulfils. This task can take two forms: (i) pushed by the home node to the remote nodes or (ii) pulled by the remote node from the home node. Two approaches to the dissemination are required because it cannot be assumed that the home node supports the interfaces provided by remote node; in such cases pushing the privacy policies would not be possible. Therefore, in situations where the remote node does not receive pushed policy updates from the home node they must periodically poll the home node for new versions of the privacy policies.

5.2.5 Element Interactions

Context Creation

Figure 5.6 outlines the sequence of interactions that occur between architectural elements when a piece of context is created. In this example the context data that is being created is
a detection event, where the user’s RFID tag was detected entering the environment. The context data is pushed from a location sensor in the local environment to the CRUD interface. Additionally, the node in the example is not the user’s home node.

The sequence begins with the ‘Create’ portion of the CRUD interface receiving the raw context data from the location sensor. The location sensor is then authenticated and its permissions pertaining to the creation of context data are checked. Once successfully authenticated, the raw context data is provided to the CRUD component which manages the remainder of the interactions. It begins by asking the classification component to classifying the raw context data. The raw context data is formatted as a ‘detectionEvent’ and returned to the CRUD component. Subsequent to classification, the subject of the detectionEvent must be established using the mobility monitor. In the example in the figure, the mapping between the user’s RFID and its URI was already know and there was no need to consult the lookup service prior to returning the user’s URI. The detectionEvent is then passed to the handling component which infers some new context data concerning the location of the user, at coarser levels of accuracy than the original detectionEvent. The original and new context data is then stored by in the context store and a URI for the original detection is returned to the location sensor.

The sequence concludes with the mobility monitor informing the user’s home node of the presence of new context data on the node and the calling of the policy manager. The policy manager checks to see if there are policies for the user stored locally, otherwise they must be retrieved from the user’s home node. In this case the policies already exist on the node as the user was previously in the environment.
5.2. Node Architecture

Context Reading

The typical steps involved in the reading of context data are outlined in Figure 5.7. The sequence begins with the authentication of the requester. Upon successful authentication the read request is provided to the ‘Read’ portion of the CRUD component.

Prior to retrieving the requested context data the permissions of the requester must be established. To begin with the policy execution component retrieves the relevant policies from the policy store and any context data relevant to the execution of those policies. The requester’s access permissions are then established and the appropriate context data is requested by the CRUD component from the context store. This context data is then returned to the requester.

Context Updating and Deletion

Figures 5.8 and 5.9 outline the interactions involved in the updating and deletion operations respectively. Both sequences are similar and involve similar element interactions. Both sequences begin with an authentication and permission check prior to the request being passed to the CRUD component. The CRUD component then provides the request to the handling component.

For the updating sequence the handling component updates the original context data and if appropriate any inferred context data. The deletion of the context data depends on whether the a single context item was requested for deletion or if the related inferred context was also included. Both sequences return ‘success’/‘failure’ depending on the outcome of the operation.

5.2.6 Peripheral Nodes

The overhead of deploying and maintaining a CxaaS node may be prohibitive to certain environments; nevertheless, these environments may benefit from access to context data. The concept of a Peripheral Node is designed for such circumstances. A peripheral node is a context-aware service that consumes context data but does not make context data available to third-parties. As a result, not all of the CxaaS-Node functionality and lifecycle stages must be implemented by the node.

The differences between a peripheral node and a standard CxaaS node are:

• A peripheral node can access the CxaaS Lookup Service but cannot have users associated with it.

• A peripheral node cannot share user’s context data with third-parties, with the exception of the user’s home node. When possible the peripheral node must inform the user’s home node when they are located in their environment.
• As peripheral nodes do not share context data they do no have to enforce user privacy policies and as a result are not involved in the privacy policy distribution process.

• Service descriptions are not required as requests will not be received from third-parties.

• Where additional user context data is available within the domain of the peripheral node, it may provide this context data to the user’s home node.

Peripheral nodes are suited to context-aware applications where a minimal level of context data is available locally, thus having limited interest to third-parties. The café from the scenario in Section 2.4 is a good example of an environment that would be suited to implementation using a peripheral node due to its basic context functionality and minimal amount of context data originating locally.

5.3 Lifecycle and Requirements Mappings

This section discusses how the node elements relate to the lifecycle stages. Afterwards the requirements that are satisfied by CxaaS-Node are described.
5.3. Lifecycle and Requirements Mappings

5.3.1 Lifecycle Mapping

Figure 5.10 provides an overview of the mapping between the CxaaS lifecycle and the node architecture. As can be seen, not all of the lifecycle stages have a direct mapping to individual architectural elements.

- Acquisition: Context is acquired by a CxaaS node either by being pushed to the node via the CRUD interface or through the use of applications specific code within the node, such as pulling context data from a local sensor. Additionally, the privacy and security components are invoked as the creator of the context data must have the appropriate permissions. The steps in the sequence diagram shown in Figure 5.6 prior to the provision of the context creation request to the CRUD component, are associated with the acquisition phase of the lifecycle.

- Classification: The classification of context data is handled by the classification component. Data supplied by sensors may be classified prior to reaching the node, as a result it may not require further classification. The classification component is invoked by the CRUD component, when necessary, upon receipt of a new piece of context data. Figure 5.6 shows the CRUD component providing the classification component with some raw data for classification and a classified ‘detectionEvent’ is returned.

- Handling: The handling stage is implemented within the handling component. The handling component is called by the CRUD component upon receipt of a create, update or delete request. The sequence in Figure 5.6 shows the handling component being provided with the detectionEvent and storing the context data in the context store. The URI for the detection event that is returned can be used to read or reference to that detection event.

- Dissemination: The dissemination lifecycle stage incorporates several architectural elements. Initially the request is received by the CRUD interface, authenticated, and then forwarded to the CRUD component. The privacy policies are then invoked and if successful the context store is accessed and the context data is returned to the requester at the appropriate privacy level. The sequence diagram shown in Figure 5.7 shows the elements of the node architecture that are related to the dissemination portion of the lifecycle. The details of this sequence have been described in Section 5.2.5.

- Maintenance: The maintenance stage is handled by the maintenance component. The operational and implementational details of the maintenance component will differ from node to node because of their varying backup, archival and retention requirements.

- Disposition: The disposition stage is controlled by the deletion portion of the CRUD component and also involves the handling component. Figure 5.9 illustrates the elements that are involved in an external request to delete context data. The disposition
process related to the disposition stage of the lifecycle begins when the CRUD component receives the deletion request.

5.3.2 Requirements Mapping

Of the requirements established in Section 4.1 the ‘Individual Node’ functional requirements are concerned with internal node details, as is shown in Table 5.5. The table combines the requirements supported by CxaaS-Net, established in Section 4.3.4 of the previous chapter, with requirements supported by CxaaS-Node, which was presented in this chapter. CxaaS nodes are also involved in the realisation of the majority of the other requirements through the provision of functionality necessary at a network level. The three ‘Individual Node’ requirements in question are: FR.4.1 Implementation Agnostic, FR.4.2 Manage the entire lifecycle, FR.4.3 Policy Enforcement. Additionally, CxaaS-Node is involved in the distribution of the privacy policies, FR.3.1.

Implementation agnosticism (FR.4.1) enables ubicomp environment developers the flexibility to use the programming language and hardware most suited to the needs of their deployments. A node architecture is defined that establishes the core elements and their functionality required for CxaaS node operation. The architecture allows for the inclusion of deployment specific functionality beyond the capabilities of the core elements.

As no single framework or middleware is required for use within CxaaS the issue of node consistency regarding the handling of context data could arise. In order to handle this issue a lifecycle for context data management is defined. The lifecycle specifies the stages and processes that a user’s context data is subjected to by CxaaS nodes. By conforming to the lifecycle, regardless of implementation details, a consistency can be guaranteed regarding how heterogeneous nodes handle context data.

While the provision of interfaces for the sharing of context data and privacy policies between nodes may primarily be a CxaaS Network issue the enforcement of those policies is a node issue. The enforcement of policies, originating both locally and remotely, and the management and dissemination of policies are all included requirement FR.4.3.

5.4 Summary

This chapter outlined the CxaaS nodes, particular emphasis was placed upon the importance of the context lifecycle within the CxaaS node. Rather than requiring the use of a particular middleware in order to ensure node consistency, the approach adopted requires the incorporation of the CxaaS lifecycle steps into the deployed nodes. This ensures a consistent handling of context data across heterogeneous CxaaS nodes.

As there was no existing context lifecycle suitable for adoption within CxaaS the development of one was required. The CxaaS lifecycle strikes a balance between the robust, heavily deployed, enterprise lifecycles and the existing context data lifecycles that are not
Table 5.5: CxaaS-Node Requirements Support

used outside of their original application. The new lifecycle is focused on distributed context sharing scenarios and this is done through the incorporation of three different actors: a context provider, a CxaaS node and a context consumer. Unlike the existing context lifecycles, the privacy of context data is emphasised in the CxaaS lifecycle. Additionally, the maintenance of the context data after it has been used is included, whereas existing context lifecycles assume that context data is not longer of use after its initial use.

Following on from the development of the lifecycle a node architecture is provided and the mapping between the architecture and lifecycle is described. The architecture includes the service descriptions required for the CxaaS-Net operation in addition to the privacy and security, interface, context storage and lifecycle related components required for node operation.
Chapter 6

Data Model

The CxaaS data model consists of a formal context model, a formal description of node services, and a privacy policy model. This chapter deals with the context model and service descriptions, while the privacy policy will be covered in the following chapter. As part of the context model the notion of granularity will also be dealt with.

The ontology developed in this chapter is used in both of the other two parts of the CxaaS data model, i.e. in the privacy policy and service descriptions. Due to the distributed and heterogeneous nature of CxaaS Nodes, ontologies are well suited to the task of representing the data exchanged between the nodes. Ontologies provide a high level of expressivity, they are extensible and allow for knowledge reuse and inferencing. They provide a common understanding for the modelled data and, therefore, are appropriate for knowledge sharing between third-parties. Additionally, they allow for context to be described in a manner that is independent of the middleware used.

6.1 Ontology Abstract Conceptualisation

Many different datatypes can be classed as ‘context data’, due to the broad nature of the definition of the term. It is not feasible to incorporate all potential types of context data into a single model. As a result a core set of frequently used context types will be identified in this section. Additional application specific context types can be added thereafter, by individual developers, as necessary through the creation of new ontological concepts that link to the existing CxaaS ontology. The focus is on personal context data and, therefore, the environmental aspects of existing work will be disregarded.

6.1.1 Frequently Used Context Categories

Abowd et al. (Abowd et al., 1999) have identified categories of context data based around ‘Who, What, When, Where’. They are termed: Identity, Activity, Time and Location. Schmidt (Schmidt et al., 1999) identified three primary categories for ‘Human Factors’ within context data: User, Social Environment and Task. Schmidt’s categories are directly related
to those of Abowd with the exception of ‘Time’ which is absent from Schmidt’s categories. It can be seen in the following surveys that these basic categories of context are among the most frequently used within context aware systems and frameworks.

In 2000 Chen and Kotz ([Chen and Kotz, 2000]) surveyed twelve systems and identified 5 types of personal context data, as shown in Table 6.1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>10 out of 12 systems</td>
</tr>
<tr>
<td>Time</td>
<td>5 out of 12 systems</td>
</tr>
<tr>
<td>User’s Activity</td>
<td>3 out of 12 systems</td>
</tr>
<tr>
<td>User’s Schedule</td>
<td>2 out of 12 systems</td>
</tr>
<tr>
<td>Proximity to other users</td>
<td>2 out of 12 systems</td>
</tr>
</tbody>
</table>

Table 6.1: Chen and Kotz Context Types

Bolchini et al.’s ‘Data-oriented Survey of Context Models’ ([Bolchini et al., 2007]) analysed sixteen context models, from which they identified six ‘modeled aspects’. The modelled aspects are classes represented within the models. Of these six modelled aspects there are three context types: (i) location, (ii) time and (iii) user profile. Table 6.2 outlines the number of occurrences of the identified context types.

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>12 out of 16 systems</td>
</tr>
<tr>
<td>Time</td>
<td>12 out of 16 systems</td>
</tr>
<tr>
<td>User Profile</td>
<td>14 out of 16 systems</td>
</tr>
<tr>
<td>(both role and feature based)</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: Bolchini’s Context Types

Bisgaard et al. ([Bisgaard et al., 2008]) have analysed the definitions used for context data in an attempt to identify the most frequently used context categories. Based on seven context definitions they have identified ten context categories, of which five are related to personal context data. Table 6.3 summarises the personal context component of Bisgaard et al.’s findings. The ‘Social Setting’ context category refers to Schilit’s ([Schilit et al., 1994]) research, in which he gives an example for a social setting as ‘whether you are with your manager or with a co-worker’. Therefore, the ‘Social Setting’ category can be related back to the ‘Proximity To Other Users’ context type identified by Chen and Kotz ([Chen and Kotz, 2000]).

From the three tables it is clear that Location and Time are the most frequently used context categories. The User Profile/Identity is the next most common category followed by User Activity, Proximity and User Schedule. The proximity context type differs from
other context types as it can relate to two individuals, i.e. the proximity of two individuals to each other. From a privacy prospective, this may result in additional overhead regarding the sharing of this context data because it can relate to two users and thus requires two sets of user policies. An alternative approach is adopted within the CxaaS ontology that maintains the single piece of context to single user mapping of the other context types. User location data is used instead of proximity as similar information can be constructed through the use of two pieces of location data as one piece of proximity data. Therefore, the core common context categories required within the CxaaS model are:

- **Location**: Indoor and outdoor location.
- **Time**: Instant time and duration.
- **User Profile**: Identifiers, profile and contact details.
- **Activity**: Common daily activities.
- **User Schedule**: User’s daily calendar including past, present and future schedules.

### 6.1.2 CxaaS Specific Context Categories

In addition to the common context categories identified in the previous section some CxaaS specific context types are required. These additional context types are necessary to ensure the operation of the CxaaS model in a distributed manner. The following additional concepts are required for the distributed setting:

- **Nodes**: Model the profile of the nodes, such as owner, contact details, service offered.
- **Device**: Model the devices carried by users and used to identify them.
- **Events**: Model the events that concern a user, including events that occur at a particular location and calendar events.
- **Online Presence**: This is not a typical ubicomp context-type, however, the web-oriented nature of CxaaS makes its inclusion possible.
The combined set of concepts is not a complete model for supporting context-awareness in all applications. It is a core set that is applicable to a broad range of applications and which can be extended by individual applications for their particular needs.

6.2 Analysis of Existing Ontologies

This section examines existing ontologies in order to identify their applicability for use within CxaaS. Two classes of ontologies will be considered: (i) integrated context ontologies, which are designed for use within UbiComp deployments, and (ii) ancillary ontologies that are not specifically designed for use within UbiComp deployments but support one or more of the context categories previously identified.

Within this thesis the definition that ‘ontologies are shared models of a domain that encode a view which is common to a set of different parties’ [Bouquet et al., 2004] is adopted. Therefore, where possible ontologies should be in use beyond the application of the ontology creator and the ontologies should be publicly available for use by third-parties.

6.2.1 Integrated Context Ontologies

While many context ontologies exist they have gained minimal traction both within the UbiComp domain and outside of the domain. The exact reasons for the lack of traction are not clear however, some influencing factors include:

- Many of the ontologies are not publicly available and only sections of the ontologies are available in publications, as is the case with IYOUIT [iyoi].
- Researchers tend to develop completely new ontologies, rather than expanding upon existing ones, because of differences between the application deployment, as is the case with CoDAMoS [Preuveneers et al., 2004].
- The ontologies have limited application to domains beyond UbiComp.
- The ontologies are overly holistic and, therefore, not useful in practice. Such heavy-weight ontologies are difficult to use and often find it difficult to gain traction in comparison with lighter, bottom-up, approaches.

The following is an overview of a selection of context ontologies. The chosen ontologies originate from heavily cited ubicomp research projects that have developed context ontologies as part of the research.

- CoDAMoS: The CoDAMoS (Context-Driven Adaptation of Mobile Services) project was an Ambient Intelligence project focused on mobile devices that ran services which adapted according to the user’s context. As part of the project an ontology was developed for modelling context and the device properties. The ontology is highly application
specific and there is no connection with third-party ontologies or use of existing concepts within the ontology. Additionally, the ontology offers very little semantics for the concepts, for instance the class for *Time* and *Location* are effectively just place-holders and any value of any type can be associated with them.

- **CONON**: The CONON ontology \cite{Wang2004a} consists of an upper level ontology used for modelling generic context information, combined with application specific ontologies for each particular application. The ontology provides mechanisms for modelling users, devices, locations, activities and their associated properties. Unfortunately, the ontology is not publicly available and furthermore a namespace for the ontology is not provided. As a result reuse of the ontology is not possible.

- **GAS**: The GAS (Gadgetware Architectural Style) Ontology \cite{Christopoulou2005} provides the modelling capabilities required for the composition of ubicomp applications using GAS. The ontology is focused on the interoperation of objects within the ubicomp environment, such as lamps and alarm clocks, rather than modelling user context as is required within Cxaas. Additionally, there does not appear to be a publicly available version of the ontology and a namespace is not provided.

- **SOCAM**: The Service-oriented Context-Aware Middleware \cite{Gu2005} is designed for context-aware mobile services and the rapid prototyping of context-aware systems. The use of ontologies is central to the design of SOCAM and two types of ontologies are used, general ontologies and domain specific ontologies. People, Activities, Services, Devices and Locations are among the types of context modelled within the general SOCAM ontology and the domain-specific ontologies are used for domains such as homes, offices and vehicles. The SOCAM ontology is promising and appears to include many of the classes required by the Cxaas context model, however, the ontology is not publicly available for reuse.

- **SOUPA**: The Standard Ontology for Ubiquitous and Pervasive Applications \cite{Chen2005} was developed by Harry Chen and used within the CoBrA middleware \cite{Chen2004}. SOUPA defines entities such as people, places, devices, time, agents, policies and events. SOUPA is an example of a holistic approach to context modelling. The SOUPA ontologies are designed to be used in a relatively closed system, rather as part of an Internet-based web service. Rather than reuse existing popular ontologies, SOUPA has opted to use of a custom representations for concepts such as geographical locations, rather than utilising the established geographical ontologies.

### 6.2.2 Ancillary Ontologies

Outside of the context specific ontologies several models exist that are relevant to individual context categories identified in Section 6.1.1.
Chapter 6. Data Model

- **FOAF**: The Friend Of A Friend Ontology (FOAF) (Brickley and Miller, 2005) can be used to describe people, their friends, groups they are members of, their activities and generally a person’s relationships with people and objects. FOAF is one of the widest adopted ontologies, with mainstream websites such as LiveJournal, DeadJournal and MyOpera using it and, therefore, is a strong candidate for modelling users within CxaaS.

- **ACRONYM**: The Annotation CReatiON for Your Media (ACRONYM) Ontology (Monaghan and O’Sullivan, 2007) was developed to provide a mechanism for annotating the presence of a nearby wireless device upon creation of context data, for instance, upon creation of a photograph. The ontology contains constructs for modelling wireless devices and, therefore, is relevant to the CxaaS context model. Due to the vast array of available personal mobile devices it is difficult to model each individual mobile device, including their properties, and keep the model updated. The ACRONYM ontology takes the pragmatic approach of abstracting over the specific type/vendor and modelling mobile devices based on their connectivity options (Bluetooth, WiFi, RFID).

- **iCal**: The iCal ontology (W3C, 2002) is an RDF representation of the iCal format. It is used to model calendars in RDF and contains classes for events, alarms and todo items amongst others. There are two related iCal ontologies, the standard iCal ontology and the iCaltzd ontology, which uses timezones as datatypes.

- **GeoNames**: GeoNames (Vatant) is a geographical database created by integrating geographical data from various sources, including the U.S. Geological Survey Geographic Names Information System and the National Geospatial-Intelligence Agency. GeoNames provides a webservice for querying their data and an ontology for formally representing that data. Additionally, users can add to and correct information in the database if necessary. The GeoNames ontology is ideal for modelling locations at a level of a building or higher, however, the ontology does not model the internal structure of buildings.

- **Event Ontology**: The Event Ontology (Abdallah et al., 2006) was developed as a generic way of modelling events and its initial application was for describing events associated with music. The Event class has six primary properties connected to it: (i) ‘factor’ is any factor or influence on the event, (ii) ‘place’ links to a location associated with the event, (iii) ‘sub_event’ is another event that is part of the original event, (iv) ‘time’ which is the time the event occurred, (v) ‘agent’ relates the event to a person or actor of some description, and (vi) ‘product’ links the event to anything produced by the event.

---

1http://www.livejournal.com/
2http://www.deadjournal.com/
3http://my.opera.com/
4http://gnswww.nga.mil/geonames/GNS/index.jsp
5http://www.nga.mil/
6.3 Formalisation of Integrated Ontology

CxaaS provides a set of context ontologies which are used to model users, devices, locations, events, presence, activities and services. Using the definition that ontologies are shared models of a domain that encode a view which is common to a set of different parties (Bouquet et al., 2004) the CxaaS model uses existing popular ontologies where possible in order to ease the extension of the model and to leverage existing contextual data rather than create a completely new set of ontologies. Several types of context data that are of use to the majority of smart spaces have been identified. The ontology adopts a bottom-up approach to defining the model, and focuses on key context types commonly used in the development of context-aware applications. The key classes and their relationships are depicted in Figure 6.1.

Throughout the remainder of the thesis a number of namespace prefixes will be used. Table 6.4 lists the namespace to prefix mappings that are used. The following subsections detail the context types contained within the CxaaS ontology.

<table>
<thead>
<tr>
<th>Ontology Name</th>
<th>Prefix</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOAF Ontology</td>
<td>foaf</td>
<td><a href="http://xmlns.com/foaf/0.1/">http://xmlns.com/foaf/0.1/</a></td>
</tr>
<tr>
<td>GeoNames Ontology</td>
<td>geoname</td>
<td><a href="http://www.geonames.org/ontology#">http://www.geonames.org/ontology#</a></td>
</tr>
<tr>
<td>ICALTZD</td>
<td>icaltd</td>
<td><a href="http://www.w3.org/2002/12/cal/icaltd">http://www.w3.org/2002/12/cal/icaltd</a></td>
</tr>
<tr>
<td>ACRONYM</td>
<td>acronym</td>
<td><a href="http://acronym.deri.org/schema#">http://acronym.deri.org/schema#</a></td>
</tr>
<tr>
<td>SKOS</td>
<td>skos</td>
<td><a href="http://www.w3.org/2008/05/skos#">http://www.w3.org/2008/05/skos#</a></td>
</tr>
<tr>
<td>Event Ontology</td>
<td>event</td>
<td><a href="http://purl.org/NET/c4dm/event.owl#">http://purl.org/NET/c4dm/event.owl#</a></td>
</tr>
<tr>
<td>Online Presence Ontology</td>
<td>opo</td>
<td><a href="http://online-presence.net/opo/ns#">http://online-presence.net/opo/ns#</a></td>
</tr>
<tr>
<td>CxaaS Core Ontology</td>
<td>cxaas</td>
<td><a href="http://cxaas.org/ontologies/cxaas#">http://cxaas.org/ontologies/cxaas#</a></td>
</tr>
<tr>
<td>CxaaS Granularity Ontology</td>
<td>ontgran</td>
<td><a href="http://cxaas.org/ontologies/ontgran#">http://cxaas.org/ontologies/ontgran#</a></td>
</tr>
</tbody>
</table>

Table 6.4: Ontology namespaces and prefixes

6.3.1 User Details

As mentioned in Section 6.2.2 FOAF is used to represent user profiles and connections and as a result it is a good choice for modelling user’s profiles within the CxaaS model.
6.3.2 Devices

The simple and pragmatic approach adopted by the ACRONYM ontology for modelling devices is well suited to application within the CxaaS model. ACRONYM removes the overhead of constantly updating the device type list and the additional development complexity of integrating a large and frequently changing ontology. The device class has been extended with the \textit{cxaas:hasDetection} property in order to connect devices to detection events.

6.3.3 Location

The GeoNames location ontology and web service is used as the core of the CxaaS location model. Much in the same way that Fire Eagle uses GeoPlanet, CxaaS nodes can query the GeoNames web service to discover, amongst others, information about a particular latitude and longitude or to find the administrative divisions in which a particular location is situated. The CxaaS Ontology extends the GeoNames ontology to provide a method of modeling buildings, floors and rooms. This extension is based on the MIBO (Niu and Kay, 2008).
building ontology and it has been adapted to inter-operate with the GeoNames ontology which uses the Simple Knowledge Organization System (SKOS) \cite{Miles2005} ontology.

### 6.3.4 Calendar Events

The iCaltzd ontology is used within the CxaaS model for describing user’s calendar events. The two classes used from iCaltzd are the \texttt{icaltzd:vEvent} and \texttt{icaltzd:vFreebusy} classes which respectively are used to model full event details or simply when the user is free or busy. Existing sources of iCal data, such as Google Calendar\footnote{http://calendar.google.com} can be harnessed as sources of context data.

### 6.3.5 Location Updates

The ‘Event’ class of the Event Ontology is used within the CxaaS model in order to describe events occurring at a particular time and place. There are two types of location events within the model, detection events (known as a \texttt{cxaas:Detection}) and miscellaneous location updates (known as a \texttt{cxaas:LocationUpdate}).

**Detection Events**

The entry of a person in a building, which is detected by a CxaaS node, results in the creation of a detection event. A detection event is associated with a time, location and device; the device is then in turn associated with a person. This allows users to have many devices associated with them. Additionally, the association of a device with a location at a particular time is a better reflection of how detections occur in reality, rather than directly associating a person with the location. The time specified in a detection event can be instantaneous or enduring, for instance, if a person is sitting at their desk for 20 minutes, rather than create a new detection every time the person is detected a single detection event can be created with a 20 minute duration.

**Miscellaneous Updates**

\textit{LocationUpdates} are used to model location updates concerning a person that are independent of detection events. Examples of such events are a user manually updating their location or a location update originating from a service such as Fire Eagle.

### 6.3.6 Online Presence

Online presence is not a traditional type of context data used within UbiComp systems. However, the distributed and web-service oriented approach proposed by the CxaaS model makes the addition of context datatypes traditionally associated with web-oriented applications feasible. Online presence is a changeable, user-oriented context data that is traditionally
provided and consumed by instant messaging clients and other web-based communication applications, however, it also has relevance for traditional UbiComp applications. Context data concerning a user’s online presence, such as the most appropriate method to contact a person, whether a person can be disturbed or whether the person is discoverable online, could be used within both the decision making or actuation process of a ubicomp application. The Online Presence Ontology models all of the aspects of online presence and integrates with FOAF and the Event Ontology.

6.3.7 Activity

The Cxaas model provides a SKOS-based classification of common human activities that can be associated with events. This allows for the annotation of user activities to detection events, location updates or general events. The activity classification is hierarchical and is based on the activities used within XEP-0108 (Meijer and Saint-Andre, 2010). The hierarchy, which is shown in Figure 6.2, is extensible and provides two levels of detail for the activities. The activities are linked with events through the use of the ‘cxaas:associatedActivity’ property.

The existing formal models pertinent to the modelling and classification of activities are limited. The Suggested Upper Merged Ontology (SUMO) (Pease et al., 2002) is an extremely large upper level ontology that amongst many other things provides mechanisms for modelling activities. Unfortunately, it does not provide a classification for the activities and due to its large nature it is cumbersome to use. An alternative approach to activity modelling is adopted by Gu et al. (Gu et al., 2009). They base their classification on ‘The Activities of Daily Living’ (ADLs) (Katz et al., 1963). The ADLs are centred on the assisted living domain, in particular, the evaluation of the level of assistance required by elderly or disabled people in the execution of common daily activities. As a result, the ADLs are overly focused on self-care activities within the home and are not sufficiently broad for the Cxaas model.

Body Position

An additional piece of context data related to user activity is the user’s body position. The increasing inclusion of accelerometer functionality in mobile devices means the monitoring of a user’s body position, in particular when combined with other sources of context data, is increasingly possible. The ‘cxaas:BodyPosition’ class has four instances, representing the four body positions of cxaas:sitting, cxaas:standing, cxaas:prostrate and cxaas:kneeling. The body positions can be associated with an event, and hence associated with an activity, through the use of the ‘associatedBodyPosition’ property.
6.3. Formalisation of Integrated Ontology

Figure 6.2: Activities Classification
6.3.8 Services and Nodes

Traditional approaches to modelling web-services, which are relevant to the modelling of CxaaS services and nodes, originate in the web-services domain. There are three established approaches: OWL-S (Martin et al., 2004), WSMO (Roman et al., 2005) and SWSO (Battle et al., 2005). WSMO and SWSO are not suitable for use within CxaaS as they are not implemented in OWL and, therefore, introduce integration difficulties with the remainder of the CxaaS Ontology. OWL-S is a complex ontology and the small portion of OWL-S 1.2 that is of interest to CxaaS can easily be reproduced in a FOAF compatible manner. Finally, current REST oriented solutions, such as SA-REST (Sheth et al., 2007), are not suitable as they restrict the implementation options by requiring the use of REST and prohibiting SOAP-based services. Therefore, it was decided to integrate the modelling of CxaaS services and nodes into the context ontology.

Within the ontology the cxaas:Service class is a general class that can be used to model any service that interacts with a CxaaS-Node. It is a sub-class of foaf:Agent and, therefore, inherits all of its properties. Three additional properties are provided for describing the services: contactPerson, serviceHomepage and textDescription.

CxaaS:Node is a specialisation of the service class and it specifically deals with CxaaS-Nodes. Unlike cxaas:Service, cxaas:Node can be directly linked to cxaas:Detection through the cxaas:detectByNode property. Finally, cxaas:PeripheralNode is a sub-class of cxaas:Node and is used to model to peripheral nodes, which were described in the previous chapter. Figure 6.3 illustrates the node classification that has just been described.
6.3.9 Linking to Remote Context and Context Sources

Home nodes have full knowledge of the distribution of their user’s accessible context data. In order to link to remote context sources or remote context data some additional properties are required within the Cxaas model. The property ‘seeNode’ links between a context instance on the home node and a remote Cxaas node that contains additional information related to the linking context instance. Similarly, the property ‘seeContext’ links between a context instance on the home node and another remote context instance. The linking context instances on the home node will be of a similar type to the remote context(s) and typically the home node instance will only have a time or duration associated with it, therefore, additional information will only be available from the remote node.

Property seeNode is used where multiple context instances are being linked to or where direct linking to the remote context instance is not appropriate. Property seeContext is used where the home node can, and it is appropriate to, link to the specific remote context instance.

In the scenario outlined in Section 2.4, Donna is attending a sales exhibition. She may spend several hours at the exhibition, moving between stands and conversing with sales representatives. The exhibitions location tracking service will detect each time Donna moves between stands and can use this information to provide some context aware services. The exhibition has two choices when informing Donna’s home node that she is located in the exhibition center:

1. Inform the home node each time she moves between stands by creating a new ‘Detection’ on the home node each time and using a ‘seeContext’ property to link to the ‘Detection’ instance on the exhibition node.

2. Inform the home node that detection information is available on the exhibition node through the creation of a ‘Detection’ on the home node instance with a ‘seeNode’ property linking to the exhibition node. The ‘Detection’ instance will have an associated time. In the exhibition situation the exhibition can initially create a ‘Detection’ event on the user’s home node where the start time is provided by the end time is not. The end time can be updated once the user leaves the exhibition center.

6.3.10 Groups

FOAF provides mechanisms for modelling groups of foaf:Agents, this includes groups of services, nodes and users. The property foaf:member connects a group to its members and cxaas:memberOf connects the member to the group.
6.4 URI Structure

Dereferenceable Uniform Resource Identifiers (URIs) are preferable within the CxaaS model. This enables users and services, with appropriate permissions, to access the data represented by the URI. The URI format provides two main advantages, it is human understandable and it is REST compatible. The preferred structure for URIs is shown in the top row of Table 6.5, where ontprefix is a prefix commonly associated with the ontology, Class is the name of the class in question within the ontology and id is an identification that is unique for that ontology and class combination within the example.org domain. A service such as prefix.cc⁷ can be used to lookup or create common mappings between ontology prefixes and URIs. The bottom row of Table 6.5 provides an example of a URI for a FOAF Person that has an identification of 16 on the conserv.deri.ie CxaaS node.

<table>
<thead>
<tr>
<th>Class URI Structure</th>
<th><a href="http://example.org/ontprefix:Class/id">http://example.org/ontprefix:Class/id</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Class URI Example</td>
<td><a href="http://conserv.deri.ie/foaf:Person/16">http://conserv.deri.ie/foaf:Person/16</a></td>
</tr>
</tbody>
</table>

Table 6.5: Class URI Structure and Example

For properties associated with a particular class the format shown in the top row of Table 6.6 is preferred. The structure is fundamentally the same as the previous example with the addition of the property details after those of the class in question. The structure of the property details are provided by the ontprefix.b, which describes a prefix for the ontology in which the property is described, and property, which is the name of the property. The bottom row of Table 6.6 provides an example of a property URI where the acronym:carries property of the class from Table 6.5 is referenced. In this case a list of devices carried by the person with the identification of 16, on conserv.deri.ie, should be returned.

<table>
<thead>
<tr>
<th>Property URI Structure</th>
<th><a href="http://example.org/ontprefix:a:Class/id/ontprefix_b:property">http://example.org/ontprefix:a:Class/id/ontprefix_b:property</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Property URI Example</td>
<td><a href="http://conserv.deri.ie/foaf:Person/16/acronym:carries">http://conserv.deri.ie/foaf:Person/16/acronym:carries</a></td>
</tr>
</tbody>
</table>

Table 6.6: Property URI Structure and Example

Objects should not depend on other objects in order to be uniquely identifiable. In particular, the URIs of objects should not contain nested references to other objects within their URIs. Table 6.7 provides an incorrect and a correct example of the preferred URI structure. The example on the top row is referencing an object with an id of 765 that has a foaf:knows relationship with the foaf:Person with the id 16. In this case the person with the id 765 should be represented by the URI shown in the second row.

⁷http://prefix.cc
6.5 Granularity

Lederer (Lederer et al., 2002) identified granularity, or accuracy, as a key user-level abstraction necessary for the management of privacy in UbiComp systems. Granularity is related to both the privacy policies and the ontology. Users will use the granularity constructs within their privacy policies to specify access permissions at varying levels of accuracy, while the constructs are defined within the ontology.

Not every context type can have different levels of granularity. For instance, the fact that a user is carrying a wireless device is part of that user’s context data, however, it is difficult to provide access to the information about that wireless device at different levels of granularity as the possible levels are not obvious or intuitive. Conversely, there is a clear distinction between the levels of accuracy for location data, event data and activity data. Location granularity can begin with exact latitude and longitude details or details of what room the user is located in and then continue to building, city, county, country, continent levels as appropriate. Event granularity is defined by the RFC 5545 (iCalendar) (Desruisseaux, 2009) standard as implemented by most calendar applications. Calendar applications, such as Google Calendar, give the option of sharing all event details, free-busy event details or not sharing any details. Figure 6.2 showed the activity data hierarchy which can be used to provide activity data at varying levels of accuracy.

This section continues by discussing the types of granularity and describing a generic method for linking between context instance data and levels of granularity. Following this, the model for the granularity levels is described and the details of the three context types (location, calendar events and activities) within the CxxaS ontology that support granularity are provided.

6.5.1 Types of Granularity

Unfortunately there are very few similarities between the event and location instance data. Event granularity levels are modelled using the distinct OWL Classes, icaltzd:Vevent and icaltzd:Vfreebusy from the iCal ontology, and there is no link between Vevent and Vfreebusy instances. Activities and locations are both SKOS-based and have an explicit hierarchy and links between the levels.

All location data is modelled using the ‘Feature’ class from the Geonames ontology and each instance has a ‘parentFeature’ property, where a parent exists, which enables applications to traverse the location hierarchy. The location data resides on a third-party server and cannot be modified. As a result it is not feasible to create links from the instance data to the
granularity levels. In contrast to the large volume of location instance data that originates from a third-party service, the activity classification has a small number of instances that are defined within the Cxaas model. Therefore, linking the activity instances to granularity levels is straightforward.

As can be seen from the varying structures of context instance data the development of a single generic method for handling context-data granularity would be cumbersome and difficult to use. Therefore, two types of granularities have been defined: (i) ‘LinkingPropertyGranularityScheme’ and (ii) ‘CommonPropertyGranularityScheme’. LinkingPropertyGranularityScheme defines a granularity scheme in which the context instance links directly to its associated granularity level, using the ‘associatedGranularity’ property. The activity instance data is an example of a set of instance data that links to the appropriate granularity levels. Figure 6.4 illustrates the operation of the LinkingPropertyGranularityScheme.

![Figure 6.4: LinkingPropertyGranularityScheme Example](http://cxaas.org/ontologies/ontgran#)

CommonPropertyGranularityScheme defines a granularity scheme in which both the context instance data and the granularity levels have a common property value that indicates the granularity level of the instance data. Both the Location granularity and the Event granularity are examples of properties that use a CommonPropertyGranularityScheme. In the case of location granularity, the location instance data has a ‘locationCode’ property that can be compared with the ‘granularityCommonProperty’ property of the granularity scheme in order to establish the appropriate granularity level. While with event granularity, the
6.5. Granularity

The ‘rdf:type’ property of the context data is compared with the ‘granularityCommonProperty’ of the granularity scheme. Figure 6.5 illustrates the operation of the CommonPropertyGranularityScheme.

![Diagram of CommonPropertyGranularityScheme Example](http://cxaas.org/ontologies/ontgran#)

Figure 6.5: CommonPropertyGranularityScheme Example

6.5.2 Modelling Granularity

The Cxaas data model provides details of three types of granularity: (i) location, (ii) calendar events, and (iii) activities. A consistent model of the granularity levels for these granularity types is provided in order for third-party applications to understand the overall hierarchy and the relationship between the levels. The granularities are modelled in the Cxaas Granularity ontology, which is referred to in shorthand as ontgran. The SKOS Ontology is used to model the granularity order, with each granularity level being modelled as a skos:Concept and all the levels for that particular context type being contained within a skos:ConceptScheme. The skos:narrower and skos:broader object properties are used to move between granularities and the cxaas_ontgran:granularityLevel datatype property is used to indicate where in a scheme a particular level is located.
6.5.3 Location Granularity

The data available from Geonames dictated the granularity levels above the level of building. Geonames contains a vast amount of geographical information which is gathered from multiple third-party datasources and, as a result, the amount of data and the level of detail varies from country to country. The Geonames related granularity levels are based upon the maximum level of details available within the Geonames data hierarchy, as a result not all of the granularity levels will be applicable for all countries. Countries such as France have very fine levels of detail and each of the steps applies to them, other countries such as Ireland and The United Kingdom do not have data for all of the levels.

At granularity levels finer than that of a building the geonames ontology and the geonames dataset are no longer of benefit. The MIBO Ontology (written using OWL) was adapted in order to make it compatible with the Geonames Ontology (written using SKOS). By combining the MIBO and Geonames ontologies it is capable to model locations within a building. Table 6.8 provides details of the location granularity levels.

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Description</th>
<th>Geonames Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Earth</td>
<td>n/a</td>
</tr>
<tr>
<td>1</td>
<td>Continent</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Country</td>
<td>Code: #A.PCLI</td>
</tr>
<tr>
<td>3</td>
<td>Administration Level 1</td>
<td>Code: #A.ADM1</td>
</tr>
<tr>
<td></td>
<td>e.g. State (US) / County(UK)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Administration Level 2</td>
<td>Code: #A.ADM2</td>
</tr>
<tr>
<td></td>
<td>e.g. County(US)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Administration Level 3</td>
<td>Code: #A.ADM3</td>
</tr>
<tr>
<td>6</td>
<td>VicinityOf:</td>
<td>Code: #P</td>
</tr>
<tr>
<td>7</td>
<td>Building</td>
<td>Code: #S</td>
</tr>
<tr>
<td>8</td>
<td>Level In A Building</td>
<td>Code: #LevelInABuilding</td>
</tr>
<tr>
<td>9</td>
<td>Room</td>
<td>Code: #SingleRoom</td>
</tr>
<tr>
<td>10</td>
<td>Personal Office Space</td>
<td>Code: #PersonalOfficeSpace</td>
</tr>
<tr>
<td>11</td>
<td>Latitude/Longitude</td>
<td>Code: #LatLong</td>
</tr>
</tbody>
</table>

Table 6.8: Location Granularity Levels

Situations where people are commencing or finishing work provide good examples of where location granularity is of use. Employers may wish to know the exact location of their mobile sales representatives while they are working, however once the working day is finished it is unlikely that the employees will be prepared to share such location information. A possible solution would be for the employee to share their location data at the exact latitude and longitude from 9 a.m. to 5 p.m. and at administration level 2 at all other times.
6.5.4 Event Granularity

Event granularity, as outlined in Table 6.9, has three levels, these levels are directly related to the access levels given in online calendar applications such as Google Calendar. Level 0 does not allow any access to the calendar data, Level 1 allows services to see when the user is either free or busy but it does not provide any details about the events in questions, the final level provides all the available details about the events.

An employee can use the event granularity levels to share the full details of work related appointments with their employer but to only share their personal appointments at a free/busy level. By sharing at a free/busy level the employee can indicate when they are unavailable without providing any information. The scenario in Section 2.4 provides an example of where event granularity can be used. In the example, Walter, the sales representative, and Donna, the potential customer, wish to arrange a follow on meeting. Both Walter and Donna have set their privacy policies to allow their calendars to be shared at a free/busy level with exhibition attendees.

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Description</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Access</td>
<td>n/a</td>
</tr>
<tr>
<td>1</td>
<td>Users Free &amp; Busy Details</td>
<td>icaltzd:Vfreebusy</td>
</tr>
<tr>
<td>2</td>
<td>Users Event Details</td>
<td>icaltzd:Vevent</td>
</tr>
</tbody>
</table>

Table 6.9: Event Granularity Levels

6.5.5 Activity Granularity

Activity granularity, as outlined in Table 6.10, has three levels and they directly related to the levels shown in Figure 6.2. Level 0 does not provide any activity information, Level 1 provides the category of activity, while Level 3 provides the exact activity undertaken.

Activity granularity can be useful for situations where users wish to provide ambiguous results to activity queries. For instance, the coach of a running squad may forbid his athletes from cycling over the winter as the likelihood of them having an accident, due to slippery or dark roads, is increased. In order to prevent the coach from finding out about his cycling, the athlete may set his privacy policy to return a result of ‘exercising’ for all activity queries from the coach during the winter.

6.6 Service Descriptions

The Web Services Description Language (WSDL) (Moreau et al., 2006), which is an XML-based language for describing Web services, is used for describing the services offered by CxxaaS nodes. In particular, WSDL 2.0 is used for modelling the CxxaaS nodes as, unlike

---

http://calendar.google.com
Chapter 6. Data Model

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Description</th>
<th>Level Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Access</td>
<td>ontgran:NoActivityInfoGran</td>
</tr>
<tr>
<td>1</td>
<td>Activity Category</td>
<td>ontgran:ActivityCategoryGran</td>
</tr>
<tr>
<td>2</td>
<td>Full Activity Info</td>
<td>ontgran:FullActivityInfoGran</td>
</tr>
</tbody>
</table>

Table 6.10: Activity Granularity Levels

WSDL 1.1 (Christensen et al., 2001), it supports RESTful web services. Semantic Annotations for WSDL and XML Schema (SAWSDL) (Kopecky et al., 2007) is used in combination with WSDL to semantically annotate the service descriptions, thus ensuring a common understanding of the functionality provided by CxaaS nodes and the data inputs required by particular interfaces and the data that is outputted from the provided interfaces.

6.6.1 Overview

WSDL provides the necessary concepts for describing web services. In particular, WSDL can model the functionality offered by a web service (modelled using the ‘Service’ concept), the URL and port of the service (modelled using the ‘Endpoint’ concept), the communication mechanisms understood by the service (modelled using the ‘Binding’ concept), the methods or functions of the service (modelled using the ‘Operation’ concept) and the structure of the messages and the data types (modelled using the ‘Types’ concept). WSDL 2.0 allows developers flexibility when designing their context-aware service implementation that was not possible with WSDL 1.1. In particular, the standard WSDL 2.0 bindings allow developers to use SOAP and REST-based implementations, additionally, third-party bindings can be used for other service protocols if necessary.

SAWSDL defines how WSDL can be annotated using semantics, in particular the input and output structures, interfaces and operations defined in WSDL. The SAWSDL annotations are independent of the bindings used by the CxaaS node; therefore, they can be applied to any WSDL description irrespective of the underlying type of service implementation or binding. SAWSDL is based on WSDL-S, which was an earlier attempt at semantic annotations for WSDL.

6.6.2 Example

Both WSDL and SAWSDL are well documented and a comprehensive examples are available (Akkiraju and Sapkota, 2007). The documented examples combining WSDL and SAWSDL focus on SOAP-based binding; therefore, in the example provided below a RESTful service will be described and annotated. In this example provided below only the reading of a detection event will be dealt with, the remaining core functionality (Create, Update, Delete) will not be provided here due to space restrictions. However, the same techniques apply to the other RESTful functions, the full code for this example is available in the Appendix on..
Listing 6.1 shows a portion of a semantically annotated WSDL file. It describes the structure of a `cxaas:Detection` object, which is a Detection object as defined in the CxaaS ontology, that is returned by a RESTful CxaaS node located at `http://conserv.deri.ie`. An XML Schema element is defined, named `detection`, within a `wsdl:type` element, and is used to define the data types returned by the web service. The individual properties of the `detection xs:element` object are annotated with the appropriate URI from the `Detection` class in the CxaaS ontology. For instance on line 13 the `location` element is annotated with the URI `http://cxaas.org/ontology#eventLocation`. This informs third-party nodes of the structure of the detection event to be returned and the meaning of each of the elements. The same annotated detection element could be used within a SOAP implementation by changing the binding used.

```xml
<wSDL:types>
    <xs:element name="detection">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="service_uri" type="anyURI" sawsdl:modelReference="http://conserv.deri.ie/ontology#Service"/>
          <xs:element name="device_uri" type="anyURI" sawsdl:modelReference="http://acronym.deri.org/schema#WirelessDevice"/>
          <xs:element name="dtstart" type="dateTime" sawsdl:modelReference="http://conserv.deri.ie/ontology#dtstart"/>
          <xs:element name="dtend" type="dateTime" sawsdl:modelReference="http://conserv.deri.ie/ontology#dtend"/>
          <xs:element name="wireless_mac" type="string" sawsdl:modelReference="http://acronym.deri.org/schema#mac"/>
          <xs:element name="bluetooth_mac" type="string" sawsdl:modelReference="http://acronym.deri.org/schema#bluetooth"/>
          <xs:element name="rfid" type="string" sawsdl:modelReference="http://acronym.deri.org/schema#rfid"/>
          <xs:element name="location" type="anyURI" sawsdl:modelReference="http://conserv.deri.ie/ontology#eventLocation"/>
          <xs:element name="latitude" type="decimal" sawsdl:modelReference="http://conserv.deri.ie/ontology#eventLatitude"/>
          <xs:element name="longitude" type="decimal" sawsdl:modelReference="http://conserv.deri.ie/ontology#eventLongitude"/>
          <xs:element name="altitude" type="decimal" sawsdl:modelReference="http://conserv.deri.ie/ontology#eventAltitude"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
</wSDL:types>
```

Listing 6.1: Piece of Service Description Example
6.7 Requirements Mapping

The CxxaaS data model touches each of the core requirements outlined in Section 4.1 however, it is primarily focused on R.3 (A formal context model suitable for the sharing of context data between third-parties, appropriate for use within the discovery and privacy mechanisms). As is specified in the description of R.3, the data model is used in the discovery (R.1) and privacy mechanisms (R.2). The discovery and privacy mechanisms require the use of a formal context model as they both deal with heterogeneous third-parties.

Of the functional requirements, FR.1.2 (Formally described interfaces), FR.2.3 (Multiple identification mechanisms), FR.3.2 (Expressivity and Granularity) are directly related to the data model. FR.1.2 relates to the service descriptions of Section 6.6. FR.2.3 relates to the methods for modelling user devices as described in Section 6.3.2 and FR.3.2 relates to context granularity which was covered in Section 6.5.

The requirements coverage is summarised in Table 6.11. The table collates the requirements supported by the CxxaaS components presented thus far. The ‘Data Model’ column is separated between the two chapters that are focused on the data model: (i) this chapter, which includes the ontology and service descriptions, and (ii) the following chapter, which focuses on the privacy policies.

The CxxaaS ontology is deemed to have partial involvement in the discovery of context data and the controlled sharing of context data. While the ontology plays a vital role in both requirements, the importance of that role is less than some of the other components of the model, such as the network architecture. Contrastingly, the ontology and service descriptions are fundamental to functional requirements FR.1.2, FR.2.3 and FR.3.2.

6.8 Summary

The provision of a data model is crucial in order to ensure the understanding of context data exchanged between third-parties. This chapter discussed the core concepts required by a context ontology. These concepts were identified by analysing the most common concepts provided by other context vocabularies. An effort was made to re-use existing ontologies where possible in order to enable the reuse of existing context data, encourage adoption and enhance the semantics of the ontology. The CxxaaS Ontology models users, devices, locations, calendar events, location updates, online presence, activities, nodes and groups.

Some of the core ontology concepts represented in the CxxaaS ontology have an implicit hierarchy associated with them. In particular, location, activities and calendar events can be provided at varying levels of accuracy. The levels of accuracy, or granularity, are modelled for each of the concepts in order to provide a common understanding of the hierarchy.

The final portion of CxxaaS-Data covered in this chapter are the service descriptions. WSDL, annotated with SAWSDL, is used to describe the interfaces provided by the nodes and the input and output structures of the interface.
6.8. Summary

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Network</th>
<th>Node</th>
<th>Data Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ontology</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Privacy</td>
</tr>
<tr>
<td><strong>Core Requirements</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R.1 Discovery</td>
<td>•</td>
<td></td>
<td>o</td>
</tr>
<tr>
<td>R.2 Controlled Sharing of Context Data</td>
<td>•</td>
<td></td>
<td>o</td>
</tr>
<tr>
<td>R.3 Formal Context Model</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **Functional Requirements**                   |         |      |            |
| FR.1 Exchange of Context Data                 |         |      |            |
| FR.1.1 Interfaces for the provision of context data to third-parties | •       |      |            |
| FR.1.2 Formally described interfaces          |         |      | •          |
| FR.2 Discovery and Mobility                   |         |      |            |
| FR.2.1 Identification of new and returning users | •       |      |            |
| FR.2.2 Discovery of context data about identified users | •       |      |            |
| FR.2.3 Multiple identification mechanisms     | •       |      | •          |
| FR.3 Privacy Protection                      |         |      |            |
| FR.3.1 Distribution of Privacy Policies       | •       |      |            |
| FR.3.2 Expressivity and Granularity           |         |      | •          |
| FR.3.3 Multiple Policy Types                 |         |      |            |
| FR.4 Individual Nodes                         |         |      |            |
| FR.4.1 Implementation Agnostic               | •       |      |            |
| FR.4.2 Manage the entire lifecycle            | •       |      |            |
| FR.4.3 Policy Enforcement                    | •       |      |            |

* = central involvement, o = partial involvement

Table 6.11: CxxaS-Data Requirements Support
Chapter 7

Privacy Protection

Lack of privacy has been cited (Cardoso and Issarny 2007; Hong and Landay 2004) as one of the primary reasons for the low number of ubiquitous computing deployments. As a result, the importance of the protection of user’s context data cannot be over-emphasised. Iachello et. al. (Iachello et al., 2005) found that, from a user’s perspective, the utility of a context-aware application is tightly coupled with the privacy control functionality. Users find it difficult to trust the privacy protection mechanisms provided by ubicomp services. Therefore, regardless of an application’s utility, users will simply not use an application if they do not trust its privacy protection mechanisms.

In contrast to ubicomp deployments, the web-based location services, such as Google Latitude and Foursquare, do not seem to suffer from these issues. They have a large number of users and provide them with direct control over their location data. It was announced in May 2010 that Google Latitude has 3 million active users (Siegler 2010), while Foursquare has over 4 million users (AboutFoursquare 2010). The accessible, web-based, mechanisms provided for updating their privacy profiles contribute to the success of the web-based approaches.

The purpose of the CxxaaS privacy policy is to provide users with a mechanism to control third-party access to their context data. Simply providing users with a mechanism to permit or prohibit access is not sufficient as users may wish to restrict access according to other context data, such as location, ongoing events, current time, service classification etc. Where applicable it may also be necessary to provide limited access to certain types of context data, such as providing access to location data at different levels of granularity.

7.1 Facets of Privacy

In order to provide appropriate privacy protection mechanisms the facets of a context data query, and any other related data, that a user wishes to use to restrict access must be identified. There exists some related work in the area of privacy protection that can be built upon and specialised in order to provide appropriate privacy protection mechanisms for the CxxaaS model.
Lederer et al. have developed a conceptual model for everyday privacy in ubiquitous computing (Lederer et al., 2002). They identified three key user-level abstractions for a ubiquitous computing privacy management system: inquirer, situation and accuracy preferences. The inquirer represents the identity of the person or service wishing to access the context-data, the situation represents the current context of the user at the time of the request and accuracy represents the granularity at which the user is prepared to provide access to the context-data. Lederer’s privacy model is partially based on the model developed by Adams (Adams, 2000) for multimedia environments.

The inquirer identity and situational information are viewed as inputs to the privacy policy and the accuracy is part of the output. Inquirer identity is the most important of the two input abstractions. Users are more likely to apply the same set of privacy rules to an inquirer regardless of the situations rather than apply a separate set of privacy rules to the inquirer in different situations (Lederer et al., 2003). However, in certain circumstances, situational information was found to be of key importance, in particular when dealing with queries from a place of work; users were happy to provide information during working hours but not outside of those hours.

The scenario presented within Lederer’s survey (Lederer et al., 2003) contains users that have a single source of context data, a mobile phone, where they have direct control over the context data. Upon receipt of a context request the users apply different ‘faces’ and each face contains a list of context data and an associated granularity level. The decision on which face to apply, and hence how much context data to return, can be made on a request by request basis by the user or they can specify the periods of time for which particular faces are applied (Lederer et al., 2002). The ‘face’ model is not applicable to the CxaaS model as it is not feasible to have users evaluate each request for context data and the time-based approach does not take sufficient details of the query’s context into account. Therefore, a direct application of Lederer’s face-based approach is not appropriate. However, the identified facets are applicable to CxaaS.

In the remainder of this chapter the abstractions are applied to the distributed CxaaS model and its specific context types, levels of granularity and policy specification approach. The following sub-sections discuss the three facets as they can be adapted to the CxaaS model. Details of how the three facets are accessible within the privacy policies will be provided in Section 7.3.

7.1.1 Requester

The requester, analogous to inquirer, can be a person, CxaaS node or a context-aware service. Given the importance of the requester identity when defining the privacy policy it is necessary to incorporate as much expressivity as possible. Individual requesters or groups of requesters, such as friends or work colleagues, should be uniquely identifiable and the user should be able to apply access constraints to them.
7.2. Policy Specification Approaches

There is no single class used to model the requester, rather, the classes defined in the previous chapter are used to model the different types of requesters. A person is modelled using ‘foaf:Person’, a CxaaS node is modelled using ‘cxaas:Node’ and a context-aware service is modelled using ‘cxaas:Service’.

7.1.2 Context

There are many different factors that can influence a user’s decision to grant or deny access to a particular piece of context data such as time, day, location or ongoing events. Therefore, as much situational information as possible pertaining to the subject of the request should be usable within the privacy policy. For instance it should be possible to allow a meeting service access to a user’s contact information for the duration of a meeting they are attending, but not outside of that. There is no single class used to model context within the privacy policies, rather the ontological structures defined for modelling context data in the previous chapters are used. Further details on this are provided in Section 7.3.

7.1.3 Granularity

Granularity, equivalent to Lederer’s accuracy, enables users to specify different levels of accuracy as an output for different access conditions. Location is a suitable context type for providing different levels of granularity. A requester could be informed that the user is located in a particular building, city, county, country or continent depending on the granularity specified by the user. The granularity constructs defined in the previous chapter is used to model granularity in the privacy policies.

7.2 Policy Specification Approaches

The goal of the CxaaS privacy protection mechanisms is to provide users with mechanisms to control the sharing of their context data between CxaaS nodes. This directly relates to requirement R.2, ‘Mechanism available for sharing context data in a controlled, privacy sensitive, manner’, and the functional requirement FR.3 and its sub-requirements.

7.2.1 Representation Options

There are two primary approaches used when handling privacy in semantic web applications, description logic (DL) based and rule-based; additionally, there is a less common hybrid approach which combines both DLs and rules. In the following subsections the merits benefits of a DLs, rules and hybrid approaches are discussed.

Description Logics Approach: A DL-based privacy policy has several advantages over a rules-based approach. Using DLs, it is possible to assert the existence of anonymous individuals and DL policies are decidable. However, in general, rule skolemization makes
reasoning undecidable (Bonati, 2009). Additionally, DLs struggle with conditions involving three or more individuals and cannot include cyclic patterns.

**Rules Approach:** Rules-based approaches can operate well when conditions involve three or more individuals. It is possible to include cyclic patterns in a rules-based privacy policy. This is not possible using DLs as they are frequently fragments of 2-variable logic and frequently enjoy tree-model or forest-model properties.

**Hybrid Approach:** Recently, hybrid approaches to privacy policies have been deployed which consist of either DLs plus rules or DL queries. This approach provides for enhanced expressiveness but inherits problems from both approaches. Due to the immaturity of this approach the explanations, abductions and advanced implementations are lacking.

Using rules to represent the Cxaas privacy policy is the logical decision for a number of reasons. It is not appropriate to use a hybrid approach due to its lack of maturity. DLs are not suitable because of their difficulty creating conditions involving three or more conditions, this is a major drawback as context-aware privacy policies frequently combine three or more conditions. For instance, if a user wished to allow a smart meeting room service at their place of work access to their location while they are at work and while they have a scheduled meeting. This relatively simple rule contains three conditions: (i) the requester is the workplace smart meeting service, (ii) only while the user is at their place of work, (iii) only while they have a scheduled meeting.

Rules are not without their drawbacks, however, the points which are difficult for rules are avoidable within Cxaas. The skolemization issue with rules is avoidable by ensuring the absence of blank nodes and well structured rules. As blank nodes are being avoided the issue concerning the assertion of anonymous individuals becomes irrelevant.

### 7.2.2 Rules Languages

There are several options for the representation of rules within the Cxaas model, this section deals with three of these, SWRL (Horrocks et al., 2004), Jena Rules (Reynolds, 2004) and RIF (Kifer and Boley, 2010).

**Semantic Web Rule Language:** SWRL is a rules language for the semantic web which combines the OWL DL and OWL Lite sub-languages of the Web Ontology Language (OWL) with the Unary/Binary Datalog RuleML sub-languages of the Rule Markup Language. The SWRL specification includes a set of builtins which allow mathematical, time, comparison, string, boolean, URI and list functions to be included within the rule body. These builtins are of use to the Cxaas privacy policy as they facilitate the inclusion of much greater complexity within the rules.
7.2. Policy Specification Approaches

**Jena Rules:** Jena rules language is a popular rules language for semantic web applications. The expressiveness of Jena rules is similar to SWRL but it is specific to the Jena platform. It contains a set of builtins similar to those contained in the SWRL specification which can be extended if necessary. Additionally, unlike the full SWRL specification, the Jena rules are decidable.

**The Rule Interchange Format:** RIF is a language for expressing rules and has several versions, known as dialects, that provide varying levels of functionality depending on a developer's requirements. The three standard RIF dialects are Core, Basic Logic Dialect (BLD) and Production Rules Dialect (PRD). The Core dialect represents a common subset of what post rule engines support. BLD extends Core with features such as logic functions and named functions. PRD introduces forward chaining rules, negation and non-monotonicity. Additionally, RIF enables the interchange of rules between existing rule languages.

The rules language to be used within the CxaaS model must support builtins, be capable of executing rules in a predefined order (stratification) and support the retraction of information (non-monotonicity). Builtins allow for the creation of expressive privacy rules containing, for instance, mathematical, time-based conditions. The stratification of rules is required in order to be able to execute the rules in a predefined order. This functionality allows privacy policies to assign levels of precedence to rules.

The dynamic nature of context data, combined with the potential of privacy policy modifications by the user, make the direct annotation of access permissions to the context data infeasible. An alternative approach, and the one adopted in CxaaS, is to evaluate the access permissions upon receipt of a request. A requirement of this approach is the ability to retract information within the privacy policy rules, in order to change the type of the access request to a ‘GrantedAccessRequest’, or a ‘DeniedAccessRequest’, after it has been processed. The constructs provided to model requests will be described in Section [7.3](#).

SWRL does have a selection of builtins, however, it does not support the retraction of information or stratification. Jena rules and the RIF-PRD support each of the listed items. The main issue with Jena rules is the necessity to use the Jena semantic web framework to execute the rules, thus restricting the implementation options for developers. Additionally, the lack of an XML-based representation for the rules introduces complexities when sharing the rules with third-parties.

It was decided to use RIF, in particular the RIF Core and the RIF PRD dialects, to represent the rules within the CxaaS privacy policies. This allows developers wishing to use a different rules language internally within their nodes to do so provided an appropriate interchange is available. Additionally, it also leaves developers the option to internally use the other two rules languages previously discussed. With an appropriate mapping Jena could be used internally within a node.
7.3 Access Requests

The privacy policies in Cxaas allow restrictions based upon the three abstractions identified by Lederer (Lederer et al., 2002). These abstractions are incorporated into the ontology representing the access requests.

7.3.1 Policy Structure

The policy structure, shown in Figure 7.1, outlines the classes and properties that make up the policy ontology. Each person has one policy, modelled as ‘cxaas:Policy’, which is comprised of multiple access requests (‘cxaas:AccessRequest’). The access requests have five properties which are used to store information about the request’s inputs, outputs or status.

![Figure 7.1: Cxaas Policy Ontology](image)

The ‘cxaas:accessingAgent’ property links to the class representing the requester, which can be a person, Cxaas node, or a generic context-aware service. The ‘cxaas:contextInstance’ property links the the context instance being requested, while the ‘cxaas:contextType’ property links to the type of the requested context data. These two properties allow the creation of privacy rules for specific instances of context data or for classes of context data.
The ‘cxaas:accessGranularity’ has two subclasses, one to represent granted access requests, and one to represent denied access requests. Once an access request has been evaluated according to the privacy policy rules it is assigned a type of ‘cxaas:GrantedAccessRequest’ or ‘cxaas:DeniedAccessRequest’. If the privacy policy rule stipulates a granularity for that request then the ‘cxaas:accessGranularity’ property is used to link to the appropriate access level. Once the request has been fully serviced the ‘cxaas:expired’ boolean can be set to true in order to signify that the request has been processed.

This approach has several benefits. The use of an AccessRequest object allows the automatic recording of each request for context data that is made as each request will have a separate AccessRequest instance that contains the details of the request and the data that was returned. This results in the rules only being executed when there is a request for context data rather than when new context data is created. Additionally, requests for context data can be indexed more efficiently as each request is represented by an AccessRequest instance.

### 7.3.2 Application of the Privacy Policy

Figure 7.2 is an activity diagram of the steps involved in applying the user’s privacy policy. The depicted sequence of events occur on a single node, where the user’s privacy rules are applied by the node, based the context data stored on the node concerning that user. The privacy policies are stored in the ‘Policy Store’, as outlined in Chapter 5.

The diagram begins with the receipt of a request for context data. Firstly, a check is done to determine if the requester is authenticated. If the requester is authenticated, an AccessRequest object that contains details of the requested contextType, the accessingAgent, the expired boolean set to false and if applicable the URI of the context instance. These details allow users to set restrictions based upon the requesting agent, the type of context requested or some particular values of the context instance. Once the AccessRequest object is instantiated/retrieved the user’s privacy rules are executed to evaluate the request. If the request meets the conditions of the privacy policy then the AccessRequest is changed to type GrantedAccessRequest.

The evaluation of the access request, to determine whether it is granted or denied, occurs separately from the servicing of the request. In order to distinguish between GrantedAccessRequests that have been serviced and those that have not the expired boolean is used. Once the request has been processed then the expired boolean is changed to true in order to prevent the node from processing the same access request multiple times. If the AccessRequest fails then its type is changed to DeniedAccessRequest and the expired boolean is set to true within the rule.

If the AccessRequested is granted then the requested context data may be required at a specific granularity. There are several ways of doing this, the implementation described in Figure 7.2 begins by retrieving the context data at the finest level of granularity available.
Chapter 7. Privacy Protection

Receive request for context data

Request Authentication

Authenticated?

No

Yes

Create AccessRequest

Apply user's privacy policy

Request Granted?

No

Yes

Reject Request

Request

Current granularity level permitted?

No

Yes

Return results

Go to next coarsest granularity level

Retrieve finest granularity context data

Figure 7.2: Privacy Policy Activity Diagram
This granularity level is then compared with the granularity permitted by the privacy policy. If the granularity retrieved is finer than the permitted level, the parent instance is retrieved and its granularity level is checked. This cycle continues until the appropriate granularity level, or a coarser level, is returned. The results are then returned to the requester and the \textit{expired} boolean is set to \texttt{true}.

In situations where the \textit{AccessRequested} is denied, no results are returned. The returning of an empty instance of the requested context type is not appropriate, as it could indicate to the requester that the context data exists, but they do not have sufficient privileges to access it. The consistent returning of ‘404 Not Found’ errors for both denied requests and requests for non-existent data is required.

### 7.3.3 Reasoning Approach

The two primary inference methods supported by reasoning engines are ‘forward chaining’ and ‘backward chaining’. For the CxaaS application, if the access permissions were annotated directly on the context data, the forward chaining approach would result in the rules being fired upon creation of new context data. Backward chaining would result in rules being fired upon receipt of a request for context data. The forward chaining approach, used for the direct annotation of context data, is not suitable for the CxaaS privacy policies because of the changeable nature of context data and the potentially high number of users and nodes. The combination of these factors could result in an extremely large amount of permissions being annotated to the context data, many of which might never be used. The backward chaining approach does not suffer from the forward chaining issue, however, the direct annotation of context data with permissions is not an elegant solution as the separation of the context data from permissions and access requests is desirable in order to maintain a clean and consistent model (separation of concerns).

The adopted approach for CxaaS is applicable to both the forward and backward chaining techniques, thus providing developers with implementation flexibility. This is important as not all reasoners support both backward and forward chaining. Upon receipt of request for context data an instantiation of a \textit{AccessRequest} object, shown in Figure 7.1 is created. This object contains the details of the context request. Upon execution of the rules the request is evaluated and it is decided if the request is granted or denied. If the request is granted, then the level of access granularity is established.

### 7.4 Privacy Policy Rules

Rules within a user’s privacy policy follow the structure of RIF rules. The basic structure is as shown if Listing 7.1 where the ‘antecedent’ section contains the conditions for execution of the rule and the ‘consequent’ contains the actions. These conditions and actions are
also known as ‘atoms’. Atoms within RIF-PDR can be positive or negative conjunctions. However, the open world assumption concerning context data within CxaaS, and thus lack of negation-as-failure, must be considered when dealing with negative conjunctions. When CxaaS nodes enforce privacy policies they use the context data within their local context store. It cannot be assumed that they have full knowledge of a user’s context data.

Negative conjunctions can be used in relation to AccessRequests, if necessary, as nodes have complete knowledge of all access requests within their node. To aid in the understanding of the example rules this section uses human readable syntax, dubbed Convenience Syntax (CS), rather than XML or RDF syntax to represent the rules. Within the CS examples generic atoms in the antecedent will be referred to using \texttt{a_atoms} and \texttt{c_atoms} for generic atoms within the consequent.

\begin{Verbatim}
\textbf{If antecedent Then consequent}
\end{Verbatim}

Listing 7.1: Basic structure of a rule

The antecedent of every CxaaS policy rule begins by searching for all instances of type \texttt{AccessRequest} that have not expired, shown in Listing 7.2. Once the appropriate \texttt{AccessRequest} instance has been located then its properties (requester details, etc.) can be used within the remaining antecedent atoms to create the desired rule conditions.

\begin{Verbatim}
\texttt{Document (}
\texttt{Prefix (\texttt{rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#>}} )
\texttt{Prefix (\texttt{rdfs <http://www.w3.org/2000/01/rdf-schema#>}} )
\texttt{Prefix (\texttt{cxaas <http://cxaas.org/ontologies/cxaas#>}} )
\texttt{Prefix (\texttt{example <http://example.com/ontology#>}} )
\texttt{Group (}
\texttt{Forall ?AccessRequest ... ?Z (}
\texttt{\quad \texttt{If And (}}
\texttt{\quad \quad ?AccessRequest \# cxaas:AccessRequest}
\texttt{\quad \quad cxaas:expired(?AccessRequest, ‘false’) }
\texttt{\quad \quad ...}
\texttt{\quad \quad a_atom}
\texttt{\quad \quad a_atom}
\texttt{\quad \quad Then c_atom c_atom}
\texttt{\quad )}
\texttt{))}
\end{Verbatim}

Listing 7.2: Basic rule structure including required first terms

There are two possible structures for the consequent portion of the policy rules depending on whether successful matching of the conditions within the antecedent should result in the access request being granted or denied. For successful requests the type \texttt{GrantedAccessRequest} is added to the request instance and the granularity of the permitted access may optionally
7.4. Privacy Policy Rules

be specified. In the following example, Listing 7.3 a GrantedAccessRequest is created at the location granularity level of a building.

Listing 7.3: Basic rule structure for a granted access request

Unsuccessful requests are given they type DeniedAccessRequest and the expired boolean is set to to true, as demonstrated in Listing 7.4

Listing 7.4: Basic rule structure for a denied access request
7.4.1 Constructing Policy Rules

The AccessRequest instance created upon receipt of a request will have the accessingAgent URI annotated to it, thereby providing the user with easy access to the identity of the requester, which is the single most important piece of information in a privacy policy. Simple access rules, such as in Listing 7.5, could check the requester’s identity and grant access to the user’s detection data at a room level.

```
Document(
  Prefix(rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(cxaas <http://cxaas.org/ontologies/cxaas#>)
  Prefix(ontgran <http://cxaas.org/ontgran#>)
  Prefix(example <http://example.com/ontology#>)

Group(
  Forall ?AccessRequest ... ?Z ( 
    If And( 
      ?AccessRequest # cxaas:AccessRequest
      cxaas:expired(?AccessRequest, ’false’)
      cxaas:accessingAgent(?s, <http://conserv.deri.ie/cxaas:Service/1>),
      cxaas:contextType(?s, cxaas:Detection)
    )
    Then Do( 
      Assert(cxaas:GrantedAccessRequest(?AccessRequest))
      Assert( ?AccessRequest[cxaas:accessGranularity->ontgran:SingleRoom])
    )
  )
)
```

Listing 7.5: Grant Service #1 access to detections at room level

Complex Rule Conditions

The second most important piece of information for users when deciding whether to grant access to a requester or not is their context. The rules structure allows the creation of conditions involving any context data stored within the user’s inference model. The RIF builtins are of particular interest when creating context conditions as they enable the execution of conditions of greater complexity. Table 7.3 provides some details of RIF builtins that can be included within the body of the rule.
7.4. Privacy Policy Rules

<table>
<thead>
<tr>
<th>Builtin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numeric-add, numeric-integer-divide</td>
<td>Arithmetic operators</td>
</tr>
<tr>
<td>numeric-less-than, numeric-not-equal</td>
<td>Numeric predicates</td>
</tr>
<tr>
<td>lower-case, concat, replace</td>
<td>String functions</td>
</tr>
<tr>
<td>hours-from-dateTime, hours-from-time</td>
<td>date/time functions</td>
</tr>
<tr>
<td>dateTime-less-than, date-less-than, time-less-than</td>
<td>date/time predicates</td>
</tr>
<tr>
<td>pred:is-literal-anyURI, pred:is-literal-boolean</td>
<td>Guard Predicates for Datatypes</td>
</tr>
<tr>
<td>xs:base64Binary, xs:date</td>
<td>Casting to XML Schema Datatypes</td>
</tr>
<tr>
<td>pred:XMLLiteral-equal, pred:XMLLiteral-not-equal</td>
<td>Functions and Predicates on rdf:XMLLiterals</td>
</tr>
</tbody>
</table>

Table 7.1: Selected RIF Builtin

Rule Precedence

Within RIF the order in which the rules are executed is defined by the order in which they appear in the policy document. However, if a rule is shared without the rest of the user’s policy being included the order of execution relative to other rules is required; therefore, it is important that users have a mechanism to control this order. CximaS uses RIF’s metadata support convention to specify the precedence of a rule. If two or more rules have the same precedence their order of execution is assumed to be inconsequential and those rules can be loaded in a random order. Listing 7.6 shows an example of a rule with a precedence of ‘100’.

Users may wish to operate their privacy policies in either optimistic or pessimistic modes. Optimistic mode grants access to context requests by default and the rules within the policy generally set the conditions for which they wish to deny access. Pessimistic mode is the inverse of this, the policy denies access by default and then rules are generally specified for the conditions when access is granted. In CximaS policies optimistic and pessimistic modes are achieved through the use of rules. Through the use of a general rule with the lowest precedence in the policy, thus being executed last, users can set their policies to optimistic or pessimistic mode. An example of a pessimistic policy is provided in Listing 7.6. If no operation mode is provided then pessimistic mode is used by default.

```
Document(
  Prefix(rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(cxaas <http://cxaas.org/ontologies/cxaas#>)
  Prefix(xs <http://www.w3.org/2001/XMLSchema#>)

    Forall ?AccessRequest(
      If And(?
        ?AccessRequest # cxaas:AccessRequest
cxaas:expired(?AccessRequest,'false')
    )
  )
)
```
7.4.2 Policy Execution

No single CxaaS node will have all of the context data concerning a particular user. Therefore, in order to fully enforce the user’s privacy policy rules, which may depend on other context types, they would have to request context data from other nodes. This solution is not scalable as a single query could result in several remote queries, which in turn could result in more remote queries. Therefore, nodes are restricted to enforcing privacy policies based on the context data they have stored locally.

This restriction is not a major issue as most context rules will depend on the user’s context at the time of the creation of the context data rather than their context data at the time of the request. Therefore, at the time the requested context data was created several other types of context data concerning the user could also have been created and stored by the same node.

The policy structure from the user policies can be used by nodes to represent their context access preferences and to represent the legal requirements for sharing context data. The order of execution can be controlled within the Meta-Policy module through the use of the precedence mechanism previously outlined.

Privacy policies can be distributed between the user’s Home Node and other nodes as a whole policy or as individual rules. It is the responsibility of the Home Node to redistribute rules once they have been modified by the user. The circulation of an entire privacy policy may not be appropriate as the rules themselves may contain sensitive information. The accessing and updating of rules is handled much in the same way context is accessed and updated. Interfaces are provided by nodes, which are described through the use of WSDL and SAWSDL.

7.5 Access Request and Privacy Policy Examples

This section provides some examples of a granted access request and a denied access request, including details of the ‘AccessRequest’ objects and the related policy rules. The privacy policy in the example, shown in Listing 7.7, belongs to the White House Chief of Staff, Bill Daley. The White House has recently installed a ubicomp service so that President...
Barack Obama can locate key members of his staff when necessary. The requests in the example are being made to Bill’s home node, which has the URI of ‘http://conserv.deri.ie’.

### 7.5.1 The Privacy Policy

Bill’s privacy policy contains three rules. Each of the rules has an associated precedence. The first rule has a precedence of 1, the second rule has a precedence of 50, while the third rule has a precedence of 100. This means the rules will be executed in the order which they appear in Listing 7.7.

The first rule in Bill’s privacy policy restricts access to location events based on three conditions. As no granularity restriction is provided, then the finest granularity available will be returned, upon successful execution of the rule. The three conditions are as follows:

1. The requester must be a person (have type foaf:Person)
2. The requester must have the firstname ‘Barack’
3. ‘Barack’ can only access the user’s location information during his term as president, i.e. between 08:00:00 on the 20th of January 2009 and 07:59:59 on the 20th of January 2013

The second rule presented is a relatively straightforward rule which restricts Service #1, from domain ‘whitehouse.gov’, to accessing the user’s calendar event data. Service #1 is granted access to the calendar information but only at the free-busy granularity level.

The final rule is the same as the rule shown in Listing 7.6. This is the default rule for the policy, which denies any access requests that have not been granted in the previous two rules.
Then Do(
    Assert(cxaas:GrantedAccessRequest(?AccessRequest))
)
)

xs:integer] *)
Forall ?AccessRequest (  
    If And(
        ?AccessRequest # cxaas:AccessRequest
        cxaas:expired(?AccessRequest,'false')
        cxaas:contextType(?AccessRequest, <http://cxaas.org/ontologies/cxaas#
        CalendarEvent>)
        cxaas:Service/1>)
    Then Do(
        Assert(cxaas:GrantedAccessRequest(?AccessRequest))
        Assert(?AccessRequest[cxaas:accessGranularity->ontgran:FreeBusyInfo ]
    )
)
)

Group( (* 'http://conserv.deri.ie/cxaas:Policy/9'[cxaas:precedence->'100'^^
xs:integer] *)
Forall ?AccessRequest(  
    If And(
        ?AccessRequest # cxaas:AccessRequest
        cxaas:expired(?AccessRequest,'false')
    Then Do(
        Assert(cxaas:DeniedAccessRequest(?AccessRequest))
        Modify(?AccessRequest[cxaas:expired->'true'])
    )
)
)

Listing 7.7: Example User Privacy Policy

Overall, this policy allows 'Barack' to access his exact location at any time during his term
as president. Calendar event requests from the White House CxaaS node, whose purpose is
unknown, are permitted at a ‘free/busy’ level. All other requests will be denied.
7.5. Access Request and Privacy Policy Examples

7.5.2 Access Requests

Barack Requests Bill’s Latest Location

Figure 7.3 shows the structure of the ‘AccessRequest’ object received by Bill’s home node. In this situation an individual context instance is requested, which is of type ‘event:Event’. The URI or the ‘cxaas:accessingAgent’ is provided and the ‘cxaas:expired’ boolean is set to false.

Assuming that Bill’s home node is using forward chaining reasoning, the ‘AccessRequest’ object, shown in Figure 7.3, will be evaluated by the node’s inference model upon its creation. Only the first rule will be executed because the request has been made by Barack Obama, who is of type ‘foaf:Person’, during his presidential term.

As all the conditions of the first rule are satisfied, the consequent of the rule is executed. The consequent gives the ‘cxaas:AccessRequest’ instance the type ‘cxaas:GrantedAccessRequest’.

In order to fulfil the request, Bill’s home node checks to see whether the ‘cxaas:AccessRequest’ instance it created has been assigned type ‘cxaas:GrantedAccessRequest’. As it has, the requested context data is returned by Bill’s home node, and the ‘cxaas:expired’ boolean is set to ‘true’.

The White House’s Cxaas Node Requests Bill’s Calendar Information

Figure 7.4 shows the structure of a ‘cxaas:AccessRequest’ instance created upon receipt of a request from a Cxaas node operating from the URI of ‘http://whitehouse.org/cxaas:Service/1’. The request is for a calendar event, which has the URI of ‘http://conserv.deri.ie/cxaas:CalendarEvent/654wv'.
Chapter 7. Privacy Protection

Upon creation of this ‘cxaas:AccessRequest’, the privacy policy rules begin executing. The first rule fails when the type of the requested context data is checked. The second rule then begins execution. As the requested context is of type ‘cxaas:CalendarEvent’, and the cxaas:accessingAgent has the URI ‘http://whitehouse.gov/cxaas:Service/1’, the rule proceeded to its consequent portion.

In the consequent, the ‘cxaas:AccessRequest’ is assigned the type ‘cxaas:GrantedAccessRequest’ and a granularity is specified for the output. When Bill’s home node reads the details of the processed request it will return the requested calendar event data at a ‘free/busy’ level of granularity.

The White House’s CxaaS Node Requests Bill’s Latest Location

The final example request for the privacy policy involves the White House’s ubicomp service requesting Bill’s latest location. Figure 7.5 is similar to the ‘cxaas:AccessRequest’ from the previous example. The two differences are in the URI and the type of the requested context data; they are ‘http://conserv.deri.ie/cxaas:CalendarEvent/564opq’ and ‘http://purl.org/NET/c4dm/event.owl#Event’ respectively.

When executing the first rule for this ‘cxaas:AccessRequest’, the first condition gets successfully passed as the requested context is of type ‘event:Event’. The second condition, however, fails, as the requester is not of type ‘foaf:Person’.

The inferencing then continues with the second rule. This rule fails when the type of the requested context data is checked. Execution of the rule requires a ‘cxaas:CalendarEvent’.

The final rule is then invoked. This is a default rule, and as such, does not have any context-based conditions in the antecedent. In the execution of the consequent, the rule
sets the type of the ‘cxaas:AccessRequest’ to ‘cxaas:DeniedAccessRequest’ and sets the ‘cxaas:expired’ boolean to ‘true’. Failed access requests do not require any further processing by the node and therefore the boolean is set to ‘false’. The node then returns a ‘404 Not Found’ error to the requester.

7.6 Requirements Mapping

Table 7.2 shows the completed table, detailing the CxaaS model’s components support for the core and functional requirements. The contributions described in this chapter, relative to the requirements, are summarised in the final column of the table.

The controlled sharing of context data (R.2) contains two primary parts: (i) the network architecture for the discovery, and sharing, of context data (CxaaS-Net), and (ii) the privacy protection mechanisms enabling the ‘controlled’ sharing of the context data. This chapter described the details of the latter of the two parts of R.2.

In terms of functional requirements, the privacy protection portion of CxaaS-Data is central to the privacy protection functional requirements (FR.3). FR.3.1 (Distribution of Privacy Policies) also has two primary parts: (i) the definition of the privacy policies, and (ii) mechanisms for the distribution of the privacy policies. The former is described within this chapter.

The privacy policy structure outlined in this chapter not only applies to user policies, but also to node and legal policies. This policy definition flexibility covers functional requirement FR.3.3 (Multiple policy types). The expressivity portion of FR.3.2 is also encompassed within the privacy policy chapter, while the granularity structures are covered by the previous
the CxaaS privacy policies enable the controlling of context access based on the identity of the requester, the type of the context requested, the details of the context instance requested, or any relevant context data within the context store. Access requests can be granted, denied or granted at a particular level of granularity.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Network</th>
<th>Node</th>
<th>Data Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ontology</td>
</tr>
<tr>
<td><strong>Core Requirements</strong></td>
<td></td>
<td></td>
<td>Privacy</td>
</tr>
<tr>
<td>R.1 Discovery</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R.2 Controlled Sharing of Context Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R.3 Formal Context Model</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Functional Requirements</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.1 Exchange of Context Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.1.1 Interfaces for the provision of context data to third-parties</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.1.2 Formally described interfaces</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.2 Discovery and Mobility</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.2.1 Identification of new and returning users</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.2.2 Discovery of context data about identified users</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.2.3 Multiple identification mechanisms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.3 Privacy Protection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.3.1 Distribution of Privacy Policies</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.3.2 Expressivity and Granularity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.3.3 Multiple Policy Types</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.4 Individual Nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.4.1 Implementation Agnostic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.4.2 Manage the entire lifecycle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR.4.3 Policy Enforcement</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* = central involvement, o = partial involvement

Table 7.2: Completed CxaaS Requirements Support Table

7.7 Summary

The privacy protection mechanisms are the final part of CxaaS-Data. In order to provide comprehensive mechanisms for the protection of context data the key facets of privacy were initially identified: (i) requester identity, and (ii) context being requested, (iii) the granularity
of the context being returned. Subsequently, the possible mechanisms for providing privacy policies, incorporating the identified privacy facets, were analysed. Three representation options were identified: (i) description logics, (ii) rules, and (iii) hybrid. A rules-based approach was decided upon due to its support for conditions involving three or more conditions.

Following this the potential rules approaches were analysed. A RIF rules-based approach is decided upon to represent the privacy policies because of its support for non-monotonicity and stratification. The ontological structures supporting the privacy rules are described and some sample rules provided. Support for complex rules is provided through the use of RIF build-ins and rule precedence mechanisms are also provided, thus allowing the specification of ‘deny by default’ or ‘grant by default’ policies.
Chapter 8

Evaluation

8.1 Introduction

To reach this point in the thesis, the fundamental ubicomp and AmI scenarios were analysed and the core challenges related to the sharing of context data were identified. From these challenges a new, updated, scenario was developed and a set of comparison criteria defined. This comparison criteria was then applied to the related work to identify their deficiencies, and based on these deficiencies, a set of requirements were defined. The coverage of these requirements by the CxaaS model, developed subsequent to the requirements, will be evaluated in this chapter.

In order to evaluate the applicability of the CxaaS model, to the controlled sharing of context data, a node and lookup service implementation has been developed and applied to the scenario specified in Section 2.4. The goal of this chapter is to determine whether the CxaaS model can fulfil the requirements in a real-world deployment, in addition to their previously established theoretical fulfilment.

Currently, there is no common understanding on how ubicomp research should be evaluated. Albrecht Schmidt (Schmidt, 2002) states that this is as a result of ubicomp researchers originating from different research domains, each of which has a established evaluation techniques. Schmidt provides details of evaluation techniques suitable for the ubicomp domain, they are as follows:

- Pre-implementation Evaluation: Used in situations where the design of a complete system can cost substantial effort. ‘Instead of implementing a system one or more humans are used to mimic the system intelligence and interacting through interfaces which leave it open to the user whether there is a computer or human behind’.

- Sub-system Evaluation: Evaluating parts of the ubicomp system, based on established evaluation techniques from the field where these sub-systems belong.
• Overall System Evaluation: Evaluation of the system as a whole. This category of evaluation has the following sub-categories:
  
  – Single domain focus: Similar to sub-system evaluation, where the focus of the evaluation is on a single, non-ubicomp, research domain.
  
  – System Feasibility: Provide a proof of concept for a particular system, through demonstrating that an implementation can realise the desired functionality.
  
  – Prototyping: The development of a prototype that provides a similar usage experience to the final envisaged system.
  
  – Living Lab: An extension of prototype-based evaluation, where the prototype is deployed, for an extended period of time, in a real-world environment.
  
  – Deployment and Studies: This type of evaluation involves the deployment of the system and creating studies based around the deployment. This type of evaluation is suitable for systems built upon mature, or off-the-shelf, technologies.

Of the evaluation options, the ‘Overall System Evaluation’ is the most suited to CxaaaS. Pre-implementation evaluation is not necessary as the implementation overhead is not unduly large. Sub-system evaluation, while technically possible, would be of limited benefit, as each of CxaaaS’s sub-systems are required in order for the controlled sharing of context data between heterogeneous third parties to be possible.

The single domain approach is not suitable as CxaaaS is wholly focused on ubicomp environments. The ‘Living Lab’ and ‘Deployment and Studies’ options are not appropriate, as the the feasibility of the CxaaaS model has not yet been established, and therefore, the adoption of either of these approaches would be premature.

A combination of the ‘System Feasibility’ and ‘Prototyping’ approaches is adopted. The adopted approach uses prototype implementations of the two components within the CxaaaS model: CxaaaS nodes and a lookup service. These prototypes are applied to the scenario, developed in Chapter 2, to establish the system feasibility.

The evaluation centres on the scenario developed in Section 2.4. That scenario is an updated and refined version of the fundamental scenarios of Weiser [Weiser, 1991] and Ducatel [Ducatel et al., 2001]. The original scenarios contained the goals, expectations, challenges and motivations of the ubicomp and AmI visions. These scenarios were dissected and the key components related to the topic of this thesis were incorporated into an updated scenario. This new scenario contains the goals, expectations, challenges and motivations for the controlled sharing of context data between heterogeneous third-parties. It is made up of three sub-scenarios, each of which will be analysed separately: (i) UbiCafé sub-scenario, (ii) Exhibition sub-scenario, and (iii) Meeting sub-scenario.

This chapter begins by analysing the requirements and identifying which of them can be demonstrated through the implementation of the scenario from Chapter 2. Following this,
the experimental setup used in the implementation of the scenario is described, particular
emphasis is placed on the instantiations of the CxaaS lookup service, and CxaaS node, used
in the implementation. Based on this, the sub-scenario implementations are described, and
their relationship with the requirements is highlighted. The final section describes in detail
how each of the core and functional requirements have been validated.

8.2 Requirements Analysis

Chapter 4 began by analysing the deficiencies of the related work, and based on this,
three core requirements and a larger set of functional requirements were provided. The core
requirements represent the deficiencies of the related work that, thus far, have prevented the
vision of the controlled sharing of context data between heterogeneous third-parties, while
the functional requirements focus the on system capabilities. There is a strong relationship
between particular core requirements and particular functional requirements and, as a result,
the satisfaction of those functional requirements will also result in the satisfaction of the core
requirement. The following paragraphs expand on this relationship.

R.1 Mechanisms for the discovery of new context data concerning new or returning users:
The discovery of context data concerning new and returning users has two primary facets: (i)
the identification of users and (ii) the discovery of context data. These two facets map directly
onto functional requirements FR.2.1 (Identification of new and returning users) and FR.2.2
(Discovery of context data about identified users). Therefore, through the satisfaction of the
two functional requirements the core requirement, R.1, will also be satisfied. The satisfaction
of these two functional requirements is outlined in Section 8.5.1.

R.2 Mechanism available for sharing context data in a controlled, privacy sensitive, man-
ner: The privacy sensitive sharing of context data also has two facets: (i) context privacy
protection, and (ii) exchange of context data. These two facets map onto the two high-level
functional requirements FR.1 (Exchange of Context Data) and FR.3 (Privacy Protection).
Through the satisfaction of the two functional requirements, the core requirement, R.2, will
also be satisfied. The satisfaction of FR.1 and FR.3 is outlined in Section 8.5.1.

R.3 A formal context model suitable for the sharing of context data between third-parties,
appropriate for use within the discovery and privacy mechanisms: Unlike R.1 and R.2,
R.3 does not have a clean mapping onto a set of functional requirements, however, an
implementation-based verification is still possible. The applicability of the context ontol-
ogy, described in Chapter 6, to the discovery and privacy mechanisms can be established in
the scenario-based implementation. The validation of this requirement will be discussed in
Section 8.5.1.
Chapter 8. Evaluation

8.2.1 Scenario Specific Requirements

The majority of the functional requirements can be evaluated through the use of an implementation to establish their validity. The following paragraphs elaborate on this and Table 8.1 summarises the findings.

**FR.1 Exchange of Context Data:** Both of the sub-requirements, FR.1.1 (Interfaces Provision) and FR.1.2 (Formal Description of Interfaces) are candidates for validation by implementation. The provision of interfaces and the use of a formal description can be demonstrated by the processing of a third-party interface description, and subsequent exchange of context data. Implementation examples concerning FR.1 are contained in each of the sub-scenarios, Sections 8.4.1, 8.4.2 and 8.4.3.

**FR.2 Discovery and Mobility:** The scenario developed in Section 2.4 contains several examples of user mobility between ubicomp environments. This movement requires the identification of the user and consequent discovery of context. The realisation of this functionality will demonstrate the applicability of the CxaaS model to the identification of new and returning users (FR.2.1) and the discovery of context data about identified users (FR.2.2). Sections 8.4.1 and 8.4.2 contain examples of these two functional requirements. Additionally, an example of FR.2.3 (Multiple Identification Mechanisms) is provided in the follow-on meeting part of the scenario, Section 8.4.3 known as the ‘meeting sub-scenario’.

**FR.3 Privacy Protection:** The expressivity and granularity of privacy policies (FR.3.2) can be demonstrated through the use of an implementation, as can the use of multiple policy types (FR.3.3). The scenario contains three examples, in Sections 8.4.2 and 8.4.3 of the use of granularity and expressive context controls in user privacy policies in addition to a single example multiple policy types, which is in Section 8.4.3. There is also an example of FR.3.1 (Distribution of Privacy Policies) across multiple environments, which are described in Sections 8.4.2 and 8.4.3.

**FR.4 Individual Nodes:** Of the three sub-requirements only FR.4.3 (Policy Enforcement) is wholly suited to validation through implementation. Each request for context data within the scenario has an associated privacy policy enforcement on the side of the context owner. Therefore, there are multiple examples of policy enforcement (FR.4.3) within the scenario, such as in Sections 8.4.2 and 8.4.3.

8.2.2 Non-Scenario Specific Requirements

There are two functional requirements that cannot be entirely validated through scenario implementation alone. These requirements will be dealt with separately from the scenario.
8.2. Requirements Analysis

The functional requirements are FR.4.1 (Implementation Agnostic) and FR.4.2 (Manage the entire lifecycle).

8.2.3 Scenario Analysis

The remainder of this section will analyse each of the sub-scenarios and identify the relationship between requirements and sub-scenarios. The scenario focuses on Donna as she interacts with three ubicomp environments over a period of two days. The scenario is business-oriented, where the primary actor wishes to source a timber frame manufacturer for a new home she is building. The first environment in the scenario is a café that Donna has previously visited, and therefore, she is a returning user in this particular ubicomp environment. It is Donna’s first time interacting with the two remaining environments, the exhibition and the office for the follow-up meeting. The following sub-sections outline the primary interactions within the sub-scenarios. Within the analysis, a single example for each of the implementation-based validation functional requirements is highlighted.

UbiCafé Sub-scenario

At the café, Donna has two interactions that involve ubicomp functionality: (i) upon entering the café Donna is greeted by name and she is told that a table similar to the one she had on her previous visit is free, and (ii) a waitress makes purchase suggestions based on the previous purchases of one of Donna’s friends. The first interaction involves Donna’s identification and the discovery of her first name. The second interaction requires the discovery of Donna’s friends and from that the café can check its records to find a recent purchase of one of Donna’s friends.

This sub-scenario provides an example of how CxaaS can be used to provide ubicomp functionality in an environment that ordinarily might not have sufficient context data. The interactions correspond with the following functional requirements:

**FR.1.1 (Interface provision):** In order to discover who Donna’s friends are, the café must request context data from interfaces provided by Donna’s home node.

**FR.1.2 (Formal description of interfaces):** The interfaces provided by Donna’s home node must be formally described in order for the café to be able to invoke them.

**FR.2.1 (Identification of new and returning users):** In order to discover context data about Donna, the café must be able to identify her.

**FR.2.2 (Discovery of context data about identified users):** The retrieval of context data concerning Donna’s friends necessitates the discovery of the location of that context data before it can be retrieved.
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
<th>Validation Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR.1.1</td>
<td>Interfaces for the provision of context data to third-parties: Interfaces must be provided through which the context data can be accessed.</td>
<td>Demonstrated by the exchange of context data with third-parties.</td>
</tr>
<tr>
<td>FR.1.2</td>
<td>Formally described interfaces: Remote nodes must be capable of understanding the interface’s functionality.</td>
<td>Demonstrated by the processing of third-party interface descriptions.</td>
</tr>
<tr>
<td>FR.2.1</td>
<td>Identification of new and returning users: All users entering a ubicomp space must be uniquely identifiable.</td>
<td>Demonstrated through the unique identification of previously unknown users in the scenario environments.</td>
</tr>
<tr>
<td>FR.2.2</td>
<td>Discovery of context data about identified users: Once a ubicomp environment has identified a user they must be able to locate context data concerning that user.</td>
<td>Demonstrated through the discovery of previously unknown context data, concerning the identified users, from remote nodes.</td>
</tr>
<tr>
<td>FR.2.3</td>
<td>Multiple identification mechanisms: Existing research often ties implementations to one particular user identification mechanism, e.g. RFID, such restrictions are not practical.</td>
<td>Demonstrated through the support for, and use of, multiple identification mechanisms within the scenario.</td>
</tr>
<tr>
<td>FR.3.1</td>
<td>Distribution of Privacy Policies: It must be possible to enforce a user’s privacy policy across all relevant nodes.</td>
<td>Demonstrated through the distribution of a user’s privacy policy to a third-party node and subsequent enforcement of that policy.</td>
</tr>
<tr>
<td>FR.3.2</td>
<td>Expressivity and Granularity: The ability to specify access permissions at varying levels of granularity based on context values is necessary.</td>
<td>Expressivity and granularity can be demonstrated through the specification of privacy policy conditions within the scenario that contain context-based conditions and granularity-based outputs from the conditions.</td>
</tr>
<tr>
<td>FR.3.3</td>
<td>Multiple Policy Types: The ubicomp service providing the data will also have privacy requirements that must be enforced.</td>
<td>Support for multiple policy types can be demonstrated through the enforcement of more than one policy type, for instance support for user and node privacy policies.</td>
</tr>
<tr>
<td>FR.4.3</td>
<td>Policy Enforcement: Nodes must enforce the policies, from both local and distributed sources.</td>
<td>Each time context data is shared with a third-party it results in a policy enforcement. FR.4.3 can be demonstrated by show examples of context data being requested and subsequent privacy policy enforcement, irrespective of whether the policies are of remote or local origin.</td>
</tr>
</tbody>
</table>
Exhibition Sub-scenario

The exhibition environment sub-scenario contains three interactions with Donna: (i) identifying the position of the timber frame manufacturers, (ii) sharing contact details for the duration of the exhibition, and (iii) sharing calendar details at a ‘free/busy’ level of granularity. Locating known users within the exhibition center does not require CxxaS-Net functionality. However, the creation of new context data requires the informing of the user’s home node and the enforcement of the user’s privacy policy, if that context data is going to be shared. The second and third interactions require the discovery of both the attendee’s and the exhibitor’s contact and calendar details.

In addition to the four functional requirements from the previous sub-scenario (FR.1.1, FR.1.2, FR.2.1 and FR.2.2), this sub-scenario also involves FR.3.2 (Expressivity and granularity), FR.3.3 (Multiple policy types) and FR.4.3 (Policy enforcement). Finally, there is a partial example of FR.3.1 (Distribution of Privacy Policies). The following list outlines the portions of the scenario related to the four previously unverified functional requirements:

**FR.3.2 (Expressivity and granularity):** Expressivity and granularity is demonstrated in the exhibition sub-scenario through the exhibition attendee’s ability to share contact details for the duration of the exhibition and share calendar details at a ‘free/busy’ level of granularity.

**FR.3.3 (Multiple policy types):** Multiple policy types are used in the sub-scenario. Donna and Walter’s policies are used in the sharing of their contact and calendar data, while the exhibitions’s node policy is used to forbid the sharing of context data concerning attendees who cannot be identified.

**FR.3.1 (Distribution of Privacy Policies):** FR.3.1 requires the distribution of privacy policies and the subsequent enforcement of those policies. This sub-scenario provides an example of the distribution of privacy policies, the enforcement of the distributed privacy policy will be dealt with in the following sub-scenario.

**FR.4.3 (Policy enforcement):** The sub-scenario contains multiple examples of the enforcement of privacy policies, such as Donna and Walter’s contact and calendar sharing restrictions.

Meeting Sub-scenario

In the final sub-scenario Walter, the timberframe manufacturer, takes a more active role. There are two main ubicomp system interactions: (i) identify that Donna is running late and adjust the calendar accordingly, and (ii) customise the portfolio based on Donna’s interests and location history. Donna restricts access to her location data to a ‘Building’ level of granularity as any information finer than this is of no use to the portfolio customisation.
This sub-scenario is the first of the three to use location data originating from outside of the ubicomp service’s domain of operation.

This sub-scenario is covers the functional requirements related to the exchange of context data (FR.1.1 and FR.1.2), the discovery and mobility functional requirements (FR.2.1 and FR.2.2) the privacy protection functional requirement (FR.3.2). The sub-scenario also contains an example of multiple identification mechanisms (FR.2.3), policy enforcement (FR.4.3) and a partial example of the distributed enforcement of privacy policies (FR.3.1).

The two remaining functional requirements that are suitable for implementation-based evaluation, but as yet haven’t been entirely covered, are FR.2.3 and FR.3.1. The following aspects of the meeting sub-scenario enable their validation:

**FR.2.3 (Multiple identification mechanisms):** The calendar-based smart meeting system uses email identifiers for meeting attendees. The combination of these email identifiers and the standard Bluetooth identifiers provide an example of multiple identification mechanisms.

**FR.3.1 (Distribution of Privacy Policies):** The previous sub-scenario provided an example of the retrieval of remote privacy policies, this sub-scenario provides an example of those privacy policies being enforced. When Donna shares her location history with Walter, in order for him to customise the portfolio, this includes Donna’s location data from the exhibition. The exhibition node invokes her privacy policies when a request is made for her location data.

### 8.3 Experimental Setup

The scenario implementation is made up of three environment specific services, one for each sub-scenario (Barista Helper, EcoHome Assistant, Smart Meetings), combined with a CxaaS Lookup Service (CLS) and an implementation of a CxaaS node, known as ConServ. The evaluation scenario uses ConServ nodes as the primary storage points for context data within each of the environments and a single CLS instance is used for the lookup functionality across all of the environments.

Figure 8.1 provides an overview of the structure of the deployment. The deployment consists of three environments, one lookup service and one remote ConServ node. The exhibition and the meeting environment contain a ubicomp service combined with a ConServ node. Within those environments, the ConServ node is tasked with the management of the privacy-sensitive context data, while the ubicomp services provide the environment-specific functionality. UbiCafé does not have the resources to deploy and manage a ConServ node, therefore, it is deployed as a peripheral node, as described in Section 5.2.6. The ubicomp services (Barista Helper, EcoHome Assistant and Smart Meetings) provide HTML interfaces for users to interact with, while the ConServ nodes provide web-service interfaces for other
8.3. Experimental Setup

ConServ nodes or the ubicomp services to interact with. Within the scenario, the remote ConServ node is used as Donna’s home node.

Figure 8.1: Evaluation Scenario Deployment

This section begins by expanding on the existing high-level scenario description, by providing instantiation specific details. Following this, CLS and ConServ are discussed in detail, including their architecture, resources and common operations. Sequence diagrams and example interactions are not included in this section as they will be provided in later sections.

8.3.1 Scenario Setup

The scenario described in Chapter 2 provides a general, high-level description. This section expands on that description, and provides additional setup details of for this particular scenario instantiation. The precise implementation details will be provided in later sections.

Café Sub-scenario

The ubicomp functionality in UbiCafé is provided by Barista Helper. Barista Helper, which will be described in detail in Section 8.4.1 is a web-based application designed to help the café’s staff provide a personalised experience for their customers. Staff interact with
Barista helper via a web-browser. They can use it to provide seating and purchase suggestions to customers based on previous visits to the café by a customer or their friends.

Barista Helper’s web interface provides staff with a list of customers detected in the café. Staff members can, via the CxaaS network, find out the identity of the customer based on their device identifier, and from this, they can discover the customer’s profile information and their friend information. This allows staff members to greet customers by name, make purchase suggestions based on what their friends purchased, and inform the customer of when their friends are next expected.

**Exhibition Sub-scenario**

The ubicomp functionality within the exhibition is provided by the combination of a web-application, called EcoHome Assistant, and a dedicated ConServ node to manage its context data. The EcoHome Assistant is targeted at the attendees of the exhibition. Attendees can use the web-interface to locate exhibitors and other attendees in real-time, exchange contact details and arrange follow-on meetings.

Prior to the exhibition commencing, all of the exhibitors register with EcoHome Assistant. As part of their registration the exhibitors provided information on their product and they are assigned a stand location. As a result the EcoHome Assistant assumes that any unregistered devices that are detected during the event are attendees of the exhibition rather than exhibitors.

EcoHome Assistant has a location service, which is constantly monitoring the location of attendees and exhibitors. When a user or attendee is detected for the first time their home node is informed. Subsequent detections will either result in the updating of the original detection entry or the creation of a new detection entry. If the person has changed location or they have not been detected in the previous five minutes then a new detection event is created in the local ConServ node and, if applicable, the person’s home node. If the person’s location is unchanged and they were recently detected in that location, then the existing detection event is updated by extending its duration.

**Follow-on Meeting Sub-scenario**

Smart Meetings, which is deployed in the conference room in Walter’s office, provides context-aware meeting room functionality for meetings with clients. The Smart Meetings web-application combines information from the company’s room booking system with information from the CxaaS network to enable the locating of meeting attendees.

The timber frame manufacturer uses a calendar-based system for booking rooms. When booking a room, the organiser specifies the meeting attendee’s email address and they are sent a confirmation email. Upon accessing Smart Meetings, a user is presented with a list of meetings they are scheduled to attend and a list of each meeting’s attendees. These lists are automatically created from the ical file associated with the conference room.
The Smart Meeting users can then request to see the location of an attendee. Additionally, Walter can use Smart Meetings to request Donna’s location history in order for him to customise the portfolio he will show Donna.

### 8.3.2 Lookup Service

CxaaS Lookup Service (CLS) is an instantiation of the lookup service, described as part of CxaaS-Net, in Section 4.3.2. Its function is to provide CxaaS nodes with the ability to map user identifiers, such as email addresses or RFIDs, to URIs representing the users. The lookup service also provides mappings between the user’s URI and the details of the home node. The remainder of this section provides the details on how this lookup service instantiation is structured and how it operates.

#### Resources

The lookup service contains details of three types of resources: (i) identifiers, (ii) users and (iii) services. The following paragraphs provide the details of the resources, and a related class diagram is contained within Figure 8.2.

![Figure 8.2: CLS Resource Class Diagram](image)

**Service:** Services equate to objects of type ‘Service’, as defined by the CxaaS ontology in Chapter 6. As such, the service can be a CxaaS node or a peripheral node. Services resources include its URI, name, location of its WSDL file, and a username and password.

The URI of the service is the one which is used within the CxaaS network. The ‘name’ property equates to the same property from the FOAF ontology, and can be applied to services of type cxaas:Service. The WSDL location property is used provide the location of the service’s WSDL file, so that other nodes can interact with it. This property equates to the ‘wsdlLocation’ property from the CxaaS Ontology. Usernames and passwords are used during the authentication process. Neither of these properties are shared with other nodes or services.

**User:** The user resource on CLS equates to the ‘Person’ class from the FOAF ontology, which is also used in the CxaaS ontology. The amount of information stored in CLS concerning users is kept to a minimum for privacy reasons. The user resource within CLS has two properties: (i) URI and (ii) usersHomeNode. The URI is that of the user within the CxaaS
network. This is not be the same as the URL that identifies the user resource on CLS. The usersHomeNode property contains the URI of the user’s home node. The details of the home node can be discovered by searching for service resources on CLS with ‘usersHomeNode’ as its URI.

**Identifier:** The identifier resources contain mappings between user identifiers and users. The identifier resource equates to the ‘UserIdentifier’ class within the CxaaS ontology. The resource has four properties: (i) URI, (ii) Identifier Type, (iii) Identifier, and (iv) User URI.

The provision of a URI, from the CxaaS Network, for the identifier resource is optional. Certain identifier types, such as RFID tags, may have URIs within the CxaaS network, while others, such as email addresses, may not. The identifier type property provides the type of the identifier represented by the resource. Examples of common identifier types are: the ‘bluetooth’ and ‘rfid’ properties from the ACRONYM ontology and the ‘mbox’ property from the FOAF ontology. The identifier property provides the value for the actual identifier, i.e. the email address, the Bluetooth MAC address or the RFID value. The user URI property relates the identifier resource back to a user resource. By knowing the user’s URI, the user resources can be searched for an entry with that value as a URI.

**Architecture**

The architecture of the lookup service developed for the scenario implementation is shown in Figure 8.3. This instantiation adopts a RESTful approach and uses the Model View Controller (MVC) architecture pattern (Burbeck, 1987).

A Representational State Transfer-based (REST) (Fielding and Taylor, 2002) architecture uses a client server approach, focused on the request and transfer of resource representations. Resources can be any addressable concept. HTTP is a popular protocol for use with RESTful application development, and is used in this evaluation, however, it is not limited to this protocol.

A RESTful approach is suited to instantiations of the CxaaS model. The CxaaS network is a client/server network, and the core functionality does not require stateful client interactions. Additionally, context data can be provided as addressable resources in a RESTful network. Through the use of URLs as the context identifiers, as shown in Table 8.3, and by making the context data accessible at those URLs, context data can be transferable and addressable resources within the RESTful CxaaS network. The benefits of adopting a RESTful approach apply equally to CxaaS nodes and to resources on the lookup service.

The MVC pattern was chosen because of its separation of the domain logic from the presentation (separation of concerns), thus enabling the attachment of different types of presentations to the same domain logic, thereby easily allowing multiple output formats, e.g. XML, JSON etc. Additionally, MVC is suited for use with RESTful web applications and its structure promotes code reuse and strong code organisation.
8.3. Experimental Setup

Figure 8.3: Lookup Service Architecture

Components

Within MVC-based architectures, requests/inputs are routed from the application interfaces to the appropriate controller. The controller instructs the model to perform specific operations based on the requests/inputs. The controller provides the outputs of the operation to the appropriate ‘view’, which renders the data from the model in a suitable form, and it is returned to the user.

The CLS MVC architecture has five sets of models, views and controllers. There is one set for each of the three resource types (service, user, identifier), one (auth) manages the registration of new service accounts and the logging in and out of services; while the final set provides the WSDL descriptions for the service. The WSDL descriptions do not have a model as the descriptions are static and stored within the views.

The security component intercepts all service requests to the interfaces. HTTP Authentication is used as the authentication mechanism. If the interacting service provides a username and password in request’s header, and the authentication is successful, then the security component forwards the request to the appropriate controller.

All of the resource and account information is stored within the data store, with the exception of the service description information, as it is static it is contained within the view.

Interfaces

The CLS interfaces are based around the Create, Read, Update and Delete (CRUD) operations. For security and privacy reasons, some of the operations have restrictions. Table 8.2 outlines the interfaces available within CLS for the three resource types.
Chapter 8. Evaluation

Due to space limitations, the details of the ‘auth’ and ‘service description’ operations have not been included in the table. The ‘auth’ set of MVC components are focused on browser-based interactions with humans and do not provide any web-service interfaces. It provides the mechanisms for users to login, logout and handle browser-based service registrations.

The ‘service description’ controller only provides methods for reading service descriptions, through the use of HTTP GET requests.

Common CLS Operations

Direct access to the resources is possible, for registered services, by sending a request to the resource’s URL. The URL structure used within CLS is outlined in the top five rows of Table 8.3 and it follows the guidelines outlined in Section 6.4.

Service Registration  To begin interacting with other CxaaS nodes, a service must register an account with the CLS. It provides some minimal profile information (username, password and name) in addition to the details of the service’s CxaaS URI and the location of the service’s WSDL file. The location of the service’s WSDL file is required as there is no default location for WSDL files to be stored within web-service deployments.

User Registration  Services are charged with handling the registration of their users and, in order to do so, they provide CLS with the URI of the user. A user can be associated with many identifiers, and the service also creates these mappings between users and identifiers. The reason the service handles these operations is to keep the level of user inconvenience to a minimum upon, initial use of CxaaS-based services.

Identifier Association  When creating an identifier entry, services provide CLS with the identifier, the type of that identifier, the URI of the person who is associated with the identifier and, if applicable, a URI that is currently used to represent that identifier. A single physical device may require multiple entries, for example, a NFC enabled mobile phone may require the registration of its Bluetooth MAC address and the NFC details.

Identifier Lookup  The ability to search for the properties of a particular identifier is crucial for the identification of new users in a smart space. Identifier entries are assumed to be globally unique, based on the combination of identifier and identifier type properties. The user and service resources also allow searching for entries based on their respective URIs. The lower three rows of Table 8.3 show the structure of these queries.

User Lookup  An individual service can only list the users and identifiers that they are associated with. However, this restriction does not apply to searching for resources. Once an identifier has been searched for, and located, it is possible to find out the user and the service associated with that identifier.
### Services

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>/cxaas:Service/</td>
<td>HTTP POST, containing: CxaaS URI, Name, WSDL Location, Username and Password</td>
</tr>
<tr>
<td>Read</td>
<td>/cxaas:Service/{SERVICE_ID}</td>
<td>HTTP GET, the ID of the service is required for this request</td>
</tr>
<tr>
<td>Update</td>
<td>/cxaas:Service/{SERVICE_ID}</td>
<td>HTTP PUT, can contain: CxaaS URI, Name, WSDL Location, Username and Password. Unprovided properties are assumed to be unchanged. Only the service can update itself.</td>
</tr>
<tr>
<td>Delete</td>
<td>/cxaas:Service/{SERVICE_ID}</td>
<td>HTTP DELETE, the ID of the service is required for this request. Only the service can delete itself.</td>
</tr>
<tr>
<td>List</td>
<td>/cxaas:Service</td>
<td>HTTP GET, no service ID is provided for this request. A list of service names and URIs is returned.</td>
</tr>
<tr>
<td>Search</td>
<td>/cxaas:Service/search?uri={SERVICE_URI}</td>
<td>HTTP GET, Search for a particular service based on its CxaaS URI. Successful searches are redirected to the service resource, unsuccessful searches are sent a '404 Not Found' error</td>
</tr>
<tr>
<td>Associated Users</td>
<td>/cxaas:Service/{SERVICE_ID}/cxaas:registeredUser</td>
<td>List all users that have this service as their home node. Access restricted to the home node, third party services cannot access this list.</td>
</tr>
</tbody>
</table>

### Users

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>/foaf:Person/</td>
<td>HTTP POST, containing: CxaaS URI of the User, home node URI. This operation can only be performed by the home node.</td>
</tr>
<tr>
<td>Read</td>
<td>/foaf:Person/{USER_ID}</td>
<td>HTTP GET, the ID of the user is required for this request</td>
</tr>
<tr>
<td>Update</td>
<td>/foaf:Person/{USER_ID}</td>
<td>HTTP PUT, used to change the home node of the user. The user’s URI cannot be changed. The operation is performed by the user’s old home node upon the user’s request.</td>
</tr>
<tr>
<td>Delete</td>
<td>/foaf:Person/{USER_ID}</td>
<td>HTTP DELETE, used when the user does not wish to be part of the CxaaS network any longer.</td>
</tr>
<tr>
<td>Search</td>
<td>/foaf:Person/search?uri={USER_URI}</td>
<td>HTTP GET, Search for a particular user based on their CxaaS URI. Successful searches are redirected to the user resource, unsuccessful searches are sent a ‘404 Not Found’ error</td>
</tr>
<tr>
<td>Associated Identifiers</td>
<td>/foaf:Person/{USER_ID}/cxaas:hasIdentifier</td>
<td>HTTP GET, List all identifiers that are associated with this user. This operation is restricted to the user's home node for privacy reasons.</td>
</tr>
</tbody>
</table>

### Identifiers

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>/cxaas:UserIdentifier/</td>
<td>HTTP POST, containing: CxaaS Identifier URI (optional), Identifier, Identifier Type, User URI. This operation can only be performed by the home node.</td>
</tr>
<tr>
<td>Read</td>
<td>/cxaas:UserIdentifier/{IDENT_ID}</td>
<td>HTTP GET, the ID of the identifier is required for this request</td>
</tr>
<tr>
<td>Update</td>
<td>/cxaas:UserIdentifier/{IDENT_ID}</td>
<td>HTTP PUT, used to change the user associated with the identifier. No other properties can be changed. The operation can only be preformed by the home node.</td>
</tr>
<tr>
<td>Delete</td>
<td>/cxaas:UserIdentifier/{IDENT_ID}</td>
<td>HTTP DELETE, used when identifier is no longer in use. The operation can only be preformed by the home node.</td>
</tr>
<tr>
<td>Search</td>
<td>/cxaas:UserIdentifier/search?identifier={IDENT} &amp;identifierType={IDENT_TYPE}</td>
<td>HTTP GET, search for an identifier resource based on the type and value of the identifier. Redirected to the resource if search is successful, otherwise a ‘404 Not Found’ error is sent.</td>
</tr>
</tbody>
</table>

Table 8.2: CxaaS Lookup Service Interfaces
8.3.3 ConServ: A CxaaS Node

ConServ is also a RESTful web-application, employing the Model-View-Controller (MVC) architectural pattern. ConServ is an implementation of a CxaaS node, and is focused on the management of context data across its entire lifecycle. ConServ is designed to be a generic CxaaS node, not focusing on an individual application, which can be used across multiple deployments. ConServ provides the context management functionality, while the application specific functionality must be provided by a local interacting service. Two examples of ConServ being used with application specific local ubicomp services will be provided later in this chapter.

This section describes ConServ, its architecture, interfaces and typical operations.

Architecture

The ConServ architecture is shown in Figure 8.4 and, as previously mentioned, it adopts an MVC-based design. The reasoning behind the adoption of a RESTful and MVC-based design is the same as for the lookup service. In addition, the loosely coupled and stateless nature of RESTful web-services, coupled with RESTful frameworks suitable for rapid prototyping (Hansson 2004; Rocher and Brown 2009), allows for the rapid development of interacting ubicomp services.

The architecture depicted in Figure 8.4 differs from the node architecture in Section 5.2 because it is structured in an MVC format. The primary differences between the two diagrams is due to the ‘Node Components’ from the original CxaaS node architecture being separated into model and controller layers in ConServ’s architecture diagram. The combination of the model and controller layers into a single layer would result in an architectural diagram which is very similar to the CxaaS node architecture.

<table>
<thead>
<tr>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service profile</td>
<td><a href="http://cxaas.org/cxaas:Service/%7BSERVICE_ID%7D">http://cxaas.org/cxaas:Service/{SERVICE_ID}</a></td>
</tr>
<tr>
<td>Service’s user list</td>
<td><a href="http://cxaas.org/cxaas:Service/%7BSERVICE_ID%7D/cxaas:registeredUser">http://cxaas.org/cxaas:Service/{SERVICE_ID}/cxaas:registeredUser</a></td>
</tr>
<tr>
<td>User profile</td>
<td><a href="http://cxaas.org/foaf:Person/%7BUSER_ID%7D">http://cxaas.org/foaf:Person/{USER_ID}</a></td>
</tr>
<tr>
<td>User’s identifier list</td>
<td><a href="http://cxaas.org/foaf:Person/%7BUSER_ID%7D/cxaas:hasIdentifier">http://cxaas.org/foaf:Person/{USER_ID}/cxaas:hasIdentifier</a></td>
</tr>
<tr>
<td>Identifier profile</td>
<td><a href="http://cxaas.org/cxaas:UserIdentifier/%7BIDENTIFIER_ID%7D">http://cxaas.org/cxaas:UserIdentifier/{IDENTIFIER_ID}</a></td>
</tr>
<tr>
<td>Identifier search</td>
<td><a href="http://cxaas.org/cxaas:UserIdentifier/search?identifier=%7BIDENTIFIER_ID%7D&amp;identifierType=%7BIDENTIFIER_TYPE%7D">http://cxaas.org/cxaas:UserIdentifier/search?identifier={IDENTIFIER_ID}&amp;identifierType={IDENTIFIER_TYPE}</a></td>
</tr>
<tr>
<td>User search</td>
<td><a href="http://cxaas.org/foaf:Person/search?uri=%7BUSER_URI%7D">http://cxaas.org/foaf:Person/search?uri={USER_URI}</a></td>
</tr>
<tr>
<td>Service search</td>
<td><a href="http://cxaas.org/cxaas:Service/search?uri=%7BSERVICE_URI%7D">http://cxaas.org/cxaas:Service/search?uri={SERVICE_URI}</a></td>
</tr>
</tbody>
</table>

Table 8.3: CxaaS Lookup Service URL Structure
8.3. Experimental Setup

![ConServ Architecture Diagram]

**Figure 8.4: ConServ Architecture**

**URL Structure:** ConServ follows a URL structure that is similar to that of CLS, as described in Section 6.4. The example URIs used throughout this section adhere to this structure.

**Component Categories** There are four main component categories within the ConServ architecture: (i) Auth Components, (ii) Policy Components, (iii) Service Description Components, and (iv) Context Components.

Auth components provide similar functionality to the auth components in CLS. They are concerned with the registration of users and services, and the logging in and out of users and services. The auth components consist of the auth controller, auth model, auth data store and auth views. There are four views: (i) user registration, (ii) user login, (iii) service registration, and (iv) service login.

The policy components manage the creation, reading, updating and deleting of privacy policies. The policy components are made up of: the policy controller, the policy views, the policy mode and the policy store. RESTful interfaces are provided for web-services to interact with the policy controller. There are four views: (i) create, (ii) edit, (iii) list, and (iv) show.

The service description components consist of the service description controller and views. They provide WSDL service descriptions for the services offered by the ConServ node. The WSDL descriptions are static and as a result do not have an associated model. The WSDL files can be accessed by sending a HTTP GET request to the appropriate URL.
Chapter 8. Evaluation

The context components are a set of models, views and controllers associated with a range of context resources. Each type of context resource has an associated model, controller and set of views. The exact context types will be described later in this section. These sets of components are represented in the architecture diagram as the ‘Context Controllers’, ‘Context Views’, ‘Context Models’ and ‘Context Store’. Each context type has a set of four views, they are: (i) create, (ii) edit, (iii) list, and (iv) show.

Miscellaneous Components The primary components of the ConServ architecture are outlined above. The remaining components are Security, Policy Execution and Inference Graph. The security component performs the same functions as the security component in CLS. It intercepts all requests from the interfaces and checks whether the requester is registered. If they authenticate successfully, then the request proceeds to the appropriate controller, otherwise an error is returned.

After authentication the CxaaS node knows the identity of the requester, however, the requester’s permissions with respect to the requested context is as yet unknown. The Policy Execution component is charged with calculating the appropriate permissions for the requester. The Policy Execution component is invoked by the context model, when it receives the request, in order to calculate the permissions. The Policy Execution component creates an Inference Graph, if one does not already exist for the user. The Inference Graph combines the node and legal privacy policies, along with user’s privacy policies and context data, and calculates the permissions for the request.

Context Components and Supported Context Types

The controllers and views of the context resources are similar to most controllers and views. The controller receives the requests from the client and sends the appropriate commands or retrieves the appropriate context resource information. The view presents the resources information prior to it being returned to the client.

However, the model of the context resources is different to typical models. The the ‘Context Handling’ stage and a significant portion of the ‘Controlled Dissemination’ stage are handled by the context models. Upon receipt of a command to create a new context resource, the context model must also materialise the necessary additional granularity information. This context data is subsequently stored.

Upon receipt of a request for context information, the context model gets the ‘Policy Execution’ component to establish the requester’s access permissions. Subsequently, the context model must enforce those access permission.

Context Types The context types supported by ConServ are based on the context types within the CxaaS ontology. The ConServ instances used within the scenario focus on a subset of classes from the CxaaS ontology, a class diagram of those supported context types is provided in Figure 8.5. The class diagram illustrates the classes and properties as they are
used within ConServ, and in certain places the ranges of object properties are narrower than those defined in their associated ontologies.

Figure 8.5: ConServ’s Resources Class Diagram

The key datatype properties for the user resource, which is modelled as a foaf:Person, are defined in the FOAF spec, they are: name, firstname, surname and mbox. Additionally, there are multiple object properties that link the user to the other resources. These object properties include ‘foaf:knows’, which is used to link users to their friends, and ‘acronym:carries’, which is used to link users to devices.

The service context type, modelled as a cxaas:Service, is used to represent services, both local and remote, that interact with the ConServ instance. The properties supported by ConServ are ‘cxaas:contactPerson’ for linking the service to a person, ‘cxaas:textDescription’ for describing the functionality of the service, ‘cxaas:madeDetection’ for linking to detections associated with the service, and ‘cxaas:serviceHomepage’ for linking to the homepage of the service.

The device class, modelled as acronym:WirelessDevice, can have three datatype properties, which are defined in the ACRONYM specification, and are used to uniquely identify devices. They are: (i) ‘acronym:bluetooth’ for recording Bluetooth mac addresses, (ii) ‘acronym:rfid’ for recording a tag’s RFID details, and (iii) ‘acronym:mac’ for recording a wireless device’s WiFi mac address. Object properties are provided for linking the device to its carrier and its associated detections.

The calendar event class, modelled as cxaas:CalendarEvent, is used to store calendar event information, which can be returned in either ‘free/busy’ representation or with full event details. The datatype properties associated with this class, which are defined in the iCalztd spec, include the events start time, end time, location, description and summary.

The ‘foaf:Group’ class is used to link groups of ‘foaf:Agents’ together, which includes both services and users. For instance, these groups can be people working in the same place, or a category of services offering similar functionality. The users can restrict the access of agents, to their context data, based on their group membership.
Detection events, modelled as cxaas:Detection, are used to store information on users who are detected in a particular location by a ubicomp service. The properties associated with a detection event in ConServ include the time and location of the detection, and the device, service and/or user associated with the detection. The ‘cxaas:associatedAgent’ object property, used within ConServ for linking detections to users, is an example of a property that is used in a narrower fashion than is defined in the CxaaS ontology.

Each of the context types outlined above has an associated controller, model and set of views. The instance data for these context types is stored, as RDF, in the context store.

Interfaces

There are four types of interfaces provided by ConServ, and they directly map onto the component categories previously outlined.

The ‘Auth’ interfaces are designed for browser-based interactions and provide the functionality for registering, logging in and logging out services and users. Web service-based logging in and logging out is handled by the security component.

The ‘Web Service Description’ controller provides a single interface, from which interface descriptions can be retrieved. This interface is invoked with a HTTP GET request to the appropriate URL for that interface. Typically that url takes a form similar to: ‘http://example.org/wsdl/conserv.wsdl’.

The functionality of the remaining two types of interfaces, for context and policies, are outlined in Table 8.4. The table groups the context interfaces together due to space restriction, however, similar interface functionality is provided by each context type. The operations for both of the operation categories are based on CRUD operations. The domain portion of the URLs is not provided in the table, an entry with a URL of ‘rule/{ID}’ may take the form ‘http://example.org/rule/ID’, where ‘ID’ is the internal identifier used for that rule.

Two entries in the table are provided for the creation of context data. The first entry is for context types that are root nodes within the graph of context data. The second URL structure for the creation of context data is for context types that have parent nodes, and therefore are not root nodes, within the context graph. The exact properties provided as part of the creation or updating of context data will differ between context types, and therefore has not been specified in the table. The ‘Component Categories’ portion of this section outlined the possible properties for each context type.
### Table 8.4: ConServ Interfaces

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Policies</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create</td>
<td>/foaf:Person/{USER_ID}/rule</td>
<td>HTTP POST, containing rule name, precedence, description and code. This operation creates a new rule in the privacy policy of the user with the ID of ‘USER_ID’.</td>
</tr>
<tr>
<td>Read</td>
<td>/rule/{ID}</td>
<td>HTTP Get, the ID of the rule is required for this request.</td>
</tr>
<tr>
<td>Update</td>
<td>/rule/{ID}</td>
<td>HTTP PUT, can contain any, or all, of rule name, precedence, description and code.</td>
</tr>
<tr>
<td>Delete</td>
<td>/rule/{ID}</td>
<td>HTTP DELETE, the ID of the rule is required for this request.</td>
</tr>
<tr>
<td>List</td>
<td>/foaf:Person/{USER_ID}/rule</td>
<td>HTTP GET, returns all of the policy rules for a particular user. No rule ID is provided for this request. A list of rule names, precedences, descriptions and codes is returned.</td>
</tr>
<tr>
<td><strong>Context</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create (No parent class)</td>
<td>/ont_prefix:Class/</td>
<td>HTTP POST, containing the appropriate properties for the particular context type. This URL structure is used for context types that do not have a parent class. Examples of such context types are users and services.</td>
</tr>
<tr>
<td>Create (Has parent class)</td>
<td>/parent_ont_prefix:Parent_Class/link_prop_ont_prefix:link_prop/</td>
<td>HTTP POST, containing the appropriate properties for that particular context type. This URL structure is used for context types that do have a parent class. For instance, a wireless device must be associated with a user. The URL structure for creating the device, to which the HTTP POST request would be sent, would be ‘/foaf:Person/{USER_ID}/acronym:carries/’.</td>
</tr>
<tr>
<td>Read</td>
<td>/ont_prefix:Class/{CONTEXT_ID}</td>
<td>HTTP GET, the ID of the context instance is required for this request. ‘ont_prefix’ is the ontology prefix associated with the context type and ‘Class’ is the class of the context type.</td>
</tr>
<tr>
<td>Update</td>
<td>/ont_prefix:Class/{CONTEXT_ID}</td>
<td>HTTP PUT, used to update details of the context instance.</td>
</tr>
<tr>
<td>Delete</td>
<td>/ont_prefix:Class/{CONTEXT_ID}</td>
<td>HTTP DELETE, delete the context instance.</td>
</tr>
<tr>
<td>Search</td>
<td>/ont_prefix:Class/{CONTEXT_ID}/search?prop_prefix:prop_name={VALUE}</td>
<td>HTTP GET, Search for a particular context instance based on one of its properties. Successful searches are redirected to the context instance, unsuccessful searches are sent a ‘404 Not Found’ error.</td>
</tr>
</tbody>
</table>
Chapter 8. Evaluation

ConServ Request Processing Sequence

Figure 8.6 is a state diagram that depicts the internal states related to the receipt of a request for context data. The diagram begins with the receipt of a request for context data. The requester must be authenticated prior to any processing of the request being done. If the authentication fails then a ‘404 Not Found’ error is returned, otherwise the request is provided to the appropriate controller.

Each of these controllers maps onto associated models, which manage the data for that particular resource type. The controller begins by requesting the context data from the model. Initially, the model gets the Policy Execution component to establish the appropriate permissions for the request, and then attempts to retrieve the context data at the appropriate level. If no context data is retrieved then a ‘404 Not Found’ error is returned, otherwise the results are returned to the controller.

Upon receipt of the result, the controller calls the appropriate view and gets it to format the results as required. The correctly formatted result is then returned to the requester.

8.4 Scenario Instantiation

The following section describes the instantiations of the three sub-scenarios and outlines the requirements that they meet. Detailed analysis of the requirements will be provided in the subsequent section. Implementation details and screenshots for each of the sub-scenario implementations are provided in the Appendix.

8.4.1 UbiCafé Sub-scenario

Barista Helper is a web-application that provides ubicomp functionality in the café sub-scenario. Barista Helper is used to manage information on the café’s menu, seating, customers and customer purchases (known as visits). By constantly monitoring the presence of Bluetooth devices nearby, Barista Helper is automatically and unobtrusively aware of the ubicafé’s current customers. As Barista Helper is a peripheral node it does not implement all of the functionality that a regular CxaaS-Node does. It does not provide interfaces for third-parties as it does not allow third-parties to query it. As it does not share context data, it does not need to download the privacy policies of the users it interacts with.

Architecture

Barista Helper adopts an MVC architecture, as shown in Figure 8.7. It contains five sets of controls, views and models; one set per resource type managed by the web application. Those resource types are: menus, visits, tables, devices and customers.

The purpose of the ‘Location Monitor’ is to constantly monitor the café for the presence of new devices, and retrieve the details of the user associated with the device. This is initially done by checking existing records, using the ‘Device Model’, to establish if that device was
8.4. Scenario Instantiation

previously in the environment and, if it was, what are the associated customer details. If the device was not previously in the environment, the ‘Location Monitor’ uses CLS to retrieve the new customer’s URI and home node details. The device and customer resources are then created in their respective models.

The five sets of MVC components provide CRUD functionality for their respective resource types and the views provide HTML-based outputs which café staff view on a browser. The customer model provides two additional operations: ‘get_profile’ and ‘get_friends’.

‘CLS Helper’ is a component that contains helper functions for retrieving information from CLS’s interfaces. ‘Location Monitor’ uses this component to retrieve information on the user associated with a device detected in the café. Similarly, ‘CxaaS Operations’ is a component that contains helper functions for retrieving context data from remote CxaaS nodes.
Scenario Interaction Sequence

Figure 8.8 outlines an interaction, between Barista Helper and the CxaaS network, based on the café section of the scenario. The sequence begins with the detection of a customer’s Bluetooth device in the café. Upon detection of the new device, Barista Helper checks its records for information concerning that device. In this situation, no information is found and Barista Helper continues by searching CLS to try and discover the device carrier’s URI, their home node and the location of the home node’s WSDL file. This information is provided to the UbiCafé employees via their web-browser.

Based on the newly discovered CxaaS URI for Donna, a waiter can use Barista Helper to recheck the café’s customer records. Donna has previously been to the café, but for that last visit she was using a different device than she currently uses, thus the initial check of café records based on her current device did not return any results. Information concerning Donna’s previous visit to the café are found including details of the table Donna was last seated at. Using this information, and if the table is currently free, the waiter can suggest this table to Donna.

The second portion of the sequence commences when the waiter wishes to retrieve additional information about Donna, in particular her profile and friend information. The WSDL file of Donna’s home node is retrieved and parsed and the appropriate interfaces are identified. Following this Donna’s profile and friend list are requested. Provided the she has granted appropriate permissions, the requested context information is returned.

If Donna’s friends are discovered then Barista Helper’s historical customer records can
be used to check if any of them are also customers of the café. From this information, the employee can make menu suggestions to Donna based on their friend’s purchases, suggest a table that their friends frequent, or, if applicable, inform them that one of their friends is expected for their regular coffee in 20 minutes time.

In order to identify Donna’s friends, Barista Helper begins by retrieving the WSDL file of the customer’s home node. The WSDL file is parsed and the operation of category ‘http://cxaas.org/ontologies/service_categories#retrieveAcquaintances’ is identified. The inputs, outputs, location and binding of this operation are established and a request is sent to the appropriate location. Finally, the request result is parsed, and the URI of Donna’s friends are extracted. A similar process is used to discover Donna’s profile information, including her name.

Provided the customer is already registered with CLS and has their Bluetooth is set to discoverable then all of the previously mentioned functionality occurs unobtrusively from a customers perspective. The user is provided with a seamless and personal service, thus making their visit to the café efficient and enjoyable.

Requirements Identification

Section [8.2.3] identified that functional requirements FR.1.1 (Interface provision), FR.1.2 (Formal description of interfaces), FR.2.1 (Identification of new and returning users) and FR.2.2 (Discovery of context data about identified users) are related to this sub-scenario.
Chapter 8. Evaluation

Within the sub-scenario, the Barista Helper retrieved the profile and friend information from remote ConServ nodes. The location of these interfaces are not known in advance and, therefore, the WSDL-based formal description of the interfaces must be processed. The action of retrieving context data from the customer’s home node, and the related preceding steps, relates to requirements FR.1.1, FR.1.2, FR.2.1 and FR.2.2.

According to Table 8.1 the validation of FR.2.1 requires the demonstration of the unique identification of previously unknown users. This was shown in the café sub-scenario through the identification of Donna based on her Bluetooth device.

From Donna’s CxaaS URI, context data concerning her friends and profile was discovered from her home node. This is an example of FR.2.2, as Donna’s friend and profile data was previously not known to Barista Helper and it was located on a remote CxaaS node.

Donna’s home node, from which the context data was retrieved, was previously unknown. In order to retrieve the context data the appropriate interface had to be identified and in order to do so the remote node’s WSDL file had to be processed. This retrieval and successful processing of the remote node’s WSDL file relates to FR.1.2. The subsequent retrieval of the profile and friend information is an example of FR.1.1.

8.4.2 EcoHome Exhibition Sub-scenario

Unlike Barista Helper, the EcoHome Assistant uses a dedicated ConServ instance for the management of their context data. EcoHome Assistant is focused on enhancing the attendee’s experience while at the exhibition, by facilitating the efficient locating of exhibitors and attendees, and enabling the easy sharing of contact details and arrangement of meetings.

Architecture

Figure 8.9 shows the architecture of EcoHome Assistant. Its structure is similar to Barista Helper and it reuses many of its components, such as ‘CLS Helper’ and ‘CxaaS Operations’. EcoHome Assistant performs two functions: (i) monitoring the exhibition environment and identifying the detected attendees, and (ii) providing the attendees with the ability to locate exhibitors and other attendees, and retrieve their contact details and arrange appointments.

The Location Monitor component is tasked with monitoring the exhibition environment for attendees and the identification of those users. It makes use of the ‘CLS Helper’ component to identify the unknown attendees using the CLS. Once the users are identified Location Monitor attempts to inform their home node of the location of a new Detection event on the exhibition’s ConServ node. Location Monitor uses ‘CxaaS Operation’ to inform the remote home nodes of this.

‘ConServ Helper’ controls the interactions between EcoHome Assistant and its local ConServ node. This includes the reading, creation, updating and deletion of context data from the exhibition.
### Resource Types:
Four types of resources are controlled within EcoHome Assistant web-application: (i) Attendees, (ii) Exhibitors, (iii) Categories, and (iv) Calendars. Attendee resources contain information on attendee: name, URI, detected device identifier, their home node, whether their home node as been notified and their most recent detection location in the exhibition. The exhibitor resource contains the same information as the attendee, but in addition it also includes the sales category they are associated with and the location of their stand.

The categories resources contain information on the categories of exhibitors that are present at the exhibition. Categories simply consist of a name and a description. Each of the previous three resource types had a models, views and controllers associated with them. The calendar resources does not have an associated model, as the calendar information is retrieved from remote CxaaS nodes and not stored locally.

![EcoHome Assistant Architecture](image)

**Figure 8.9: EcoHome Assistant Architecture**

### Scenario Interaction Sequence

Figure 8.10 outlines the sequence of interactions that occur in the exhibition sub-scenario. The sequence begins with the Donna entering the exhibition space. Upon detection of the unidentified attendee, EcoHome Assistant searches the local ConServ node for existing records for the attendee. As no records are found, the lookup service is searched for an entry corresponding with the detected device. Upon successful identification of the attendee, an entry is created in the local ConServ deployment.
Following Donna’s identification, her home node must be informed of the existence of new context data concerning her. The lookup service is queried in order to find her home node and the location of its WSDL file. The WSDL file is then downloaded, parsed and the appropriate interface for the creation of a ‘cxaas:Detection’ event is identified. A new ‘cxaas:Detection’ is created on Donna’s home node and it is linked with the exhibition’s ‘cxaas:Detection’, using the ‘seeContext’ property of the Cxaas ontology. Additionally, the attendee’s rules are copied from their home node to the exhibition’s ConServ instance, via the EcoHome Assistant.

If the attendee does not have an entry in CLS then the home node cannot be informed. In such cases the attendee’s location information will not be shared with third-party applications. This default privacy setting is implemented through the use of a node policy that sets the default privacy settings to forbidding the sharing of context data with third-parties.

Subsequent detections will either result in the updating of the original detection entry, or the creation of a new detection entry. If the person has changed location or they have not been detected in the previous five minutes then a new detection event is created in the local ConServ node and, if applicable, the person’s home node. If the person’s location is unchanged, and they were recently detected in that location, then the existing detection event is updated by extending its duration.

The aforementioned operations occur unobtrusively in the background, and the web-application provides additional functionality based on the gathered information. It allows users to search for exhibitors based on their category. In the case of Donna’s search a list of timber frame manufacturers will be returned from which she can select particular individuals.

The second part of the sequence depicted in Figure 8.10 commences once Donna locates a suitable exhibitor. Through the use of EcoHome Assistant, Donna requests the contact details of the exhibitor. In this sequence, it is assumed that EcoHome Assistant is aware already aware of Walter’s home node details and, based on those details, requests a copy of the node’s WSDL file. The interface from which to request the contact information are identified from the WSDL file. The contact information is then requested and, provided sufficient permissions are provided, returned to Donna via her browser.

The final portion of the sub-scenario and sequence concerns the arrangement of a follow-on meetings between Donna and Walter. At this point the EcoHome Assistant has the home node interface details for both the exhibitor and the attendee, from which it requests information concerning their calendar events for a certain period of time. This information is then combined and displayed on a single calendar. The exhibitor and attendee can use this to easily arrange a follow-on meeting that suits both parties.

Requirements Identification

The exhibition sub-scenario expands upon the requirements covered by the first sub-scenario. In addition to those first four functional requirements, it also provides implementation-based examples for requirements FR.3.2 (Expressivity and granularity), FR.3.3 (Multiple
Figure 8.10: Exhibition Sequence Diagram
policy types) and FR.4.3 (Policy enforcement).

The examples of requirements FR.1.1, FR.1.2, FR.2.1 and FR.2.2 in this sub-scenario are very similar to the previous sub-scenario. The exchange of user’s contact details and calendar details requires the identification of users (FR.2.1) and the discovery of the location of their contact and calendar details (FR.2.2). Subsequently, formally described interfaces must be invoked in order to retrieve the context data (FR.1.1 and FR.1.2).

The exchange of contact and calendar details also includes privacy policy elements. Donna restricts the sharing of her contact details to people attending the exhibition and this policy is executed when EcoHome Assistant retrieves her contact. Additionally, Donna and Walter’s privacy policies restrict their calendar details to being shared at a free/busy level. Both of these interactions are examples of FR.3.2.

It cannot be assumed that every attendee of the exhibition will have an account with CLS. The privacy wishes of users without accounts cannot be enforced as their rules do not exist or are not accessible. In order to deal with this, the exhibition makes use of a node policy that forbids the exchange of this context data with third-parties. This is an example of FR.3.3.

Users that have accounts with CLS, and have accessible rules, have their privacy wishes enforced by the EcoHome Assistant. Upon detection of a user, EcoHome Assistant attempts to notify their home node of the user’s location and download the user’s privacy policy rules. This implementation, in addition to the previously discussed policy enforcement concerning calendar and contact details, is an example of requirement FR.4.3.

The sub-scenario also provides a partial example of the distribution of privacy policies (FR.3.1). When possible the privacy policies of the users of EcoHome Assistant are downloaded to the exhibition’s ConServ node. This is a partial example of FR.3.1 as it shows privacy policies being distributed to third-parties. However, this sub-scenario does not provide an example of these third-part policies being enforced. The meeting sub-scenario, described in the following section, contains the example of the third-party policies being enforced by the exhibition ConServ node.

8.4.3 Meeting Sub-scenario

In the scenario, Walter and Donna have arranged a follow-on meeting for the day after the exhibition. However, Donna is running late for the meeting as a previous engagement ran overtime. The conference room in Walter’s office provides some smart space functionality to enable the efficient running of meetings with clients. The Smart Meetings web-application combines information from the company’s room booking system with information from the CxaaS network to enable the locating of meeting attendees.
8.4. Scenario Instantiation

SmartMeetings reuses some of the components from EcoHome Assistant: Security, Cxaas Operations, CLS Helper, Location Monitor and ConServ Helper. These components operated in a similar fashion in the SmartMeetings deployment. In this sub-scenario the Location Monitor component monitors the timber frame manufacturer’s office rather than an exhibition space.

Three types of resources are managed by SmartMeetings: Users, Meetings and Attendees. The users resource contains basic account and login details for all users of the SmartMeetings service. The meetings resource does not have an associated model as it retrieves its information from the company’s calendaring system. This information includes details of the meeting organiser, the attendees, the meeting room location, the time and duration of the meeting. The attendee model contains information on meeting attendees. It does not have an associated controller or set of views. The attendee information is shown via the meeting controller and views.

The location monitor component monitors the local office environment and upon detection
Chapter 8. Evaluation

of a new device the local ConServ instance is updated. When the location of a meeting attendee is requested via Smart Meetings, the web-app first checks the local ConServ instance and subsequently, if necessary, the remote CxaaS nodes.

Scenario Interaction Sequence

Figure 8.12 outlines the sequence of interactions associated with the follow-on meeting portion of the scenario. There are three sets of interactions in the diagram. In the first set of interactions Donna turns on the PoFriend mobile application, which results in PoFriend updating Donna’s home node with her current location. PoFriend is a basic mobile application that periodically retrieves the mobile phone’s location and updates Donna’s home node with this information.

The second set of interactions begins with Walter using the Smart Meetings ubicomp service to locate a particular meeting attendee. Smart Meetings initially searches the local ConServ node for information regarding a person with a particular email identifier. The ConServ node does not have any record of that person and, therefore, Smart Meetings must use the lookup service to identify the meeting attendee. The lookup service provides both Donna’s URI and her home node details.

Donna’s home node’s WSDL file is downloaded and parsed, and the location interface is identified. The interfaces is then sent a query for Donna’s last known location, which was provided by PoFriend, and the information is returned to Walter’s browser.

The final set of interactions begins when Donna arrives at the meeting and Walter wishes to show her a subset of the company’s portfolio, customised based on buildings Donna may have previously seen. In order to do this, Smart Meetings requests Donna’s location history from her home node. The list returned will contain links to third-party nodes or context instances stored on third-party nodes. In this situation, one of those links is to location data from Donna’s visit to the exhibition the previous day. Smart Meetings sends a request to the exhibition’s ConServ node, which then invokes Donna’s privacy policies, and returns Donna’s location details from the exhibition.

Requirements Identification

Similar to the two previous sub-scenarios, the follow-on meeting sub-scenario also contains examples of functionality related to FR.1.1 (Interface provision), FR.1.2 (Formal description of interfaces), FR.2.1 (Identification of new and returning users) and FR.2.2 (Discovery of context data about identified users). The discovery of Donna’s current position when she is running late for the meeting, and the discovery of her location history while in the meeting, require the identification of Donna based on her Bluetooth MAC address and email address (FR.2.1), and the discovery of location data (FR.2.2) in addition to the invoking of formally described interfaces to retrieve the location data (FR.1.1 and FR.1.2). This is also an example of multiple identification mechanisms (FR.2.3), as Donna is identified by both her email
address and her Bluetooth MAC address.

FR.3.2 (Expressivity and granularity) is evident within the scenario when Donna shares her location history with Smart Meetings for the duration of the meeting. This restriction is an example of expressivity within Donna’s privacy policy.

The sharing of Donna’s most recent location and her location history requires the enforcement of privacy policies (FR.4.3). The location data returned as part of Donna’s location history includes location data originating from the exhibition on the previous day. The requesting of location data from the exhibition results in the enforcement of Donna’s privacy rules, which were downloaded on the previous day. Therefore, this is a partial example of FR.3.1. When combined with the partial example of FR.3.1 from the previous sub-scenario an complete example is formed.
8.5 Validating Requirements

This section discusses each of the core requirements and functional requirements, established in Section 4.1 for the controlled sharing of context data between heterogeneous third parties. Each of the requirements are dealt with individually, and CxaaS’s support for the requirements is outlined. To begin with, the functional requirements are discussed and subsequently the core requirements are discussed.

8.5.1 Functional Requirements

The sub-scenario interactions in Section 8.4 provide multiple, implementation-based, examples of the functional requirements. However, for convenience sake, a single set of interactions and associated sequence diagram, shown in Figure 8.13 has been developed for this section. The interaction sequence contains the key elements of the scenario interactions from the sub-scenarios. This set of interactions will primarily be referred to during the discussion of the functional requirements, rather than the three separate sequence diagrams earlier in the chapter.

The sequence contains two local services, made up of a ConServ node and a local ubicomp service, a lookup service and two remote CxaaS nodes. In the sequence, the local ubicomp service wishes to discover the last known detection event related to a user that has just entered its environment. In this situation, the detection information is not available on the user’s home node, but is available on a third party node.

FR.1.1, Interface for the Provision of Context Data

Two examples are shown in Figure 8.13 of the provision of interfaces for the retrieval of context data. In the first example, the local ubicomp service requests the context data but is redirected to a third party node that has the required context data. The first line of Listing 8.1 shows the request that is sent to the user’s home node, while Figure 8.14 describes the detection event that is returned. The recentRelatedLocation property within the CxaaS ontology connects a user with the most recent location associated with them, in this case the location is a detection event that has the URI of ‘http://conserv.deri.ie/cxaas:Detection/abc123’. The data returned by the home node contains the time associated with the detection event, and a link to the remote location of the actual detection event.

| Request #1: HTTP GET, URL=http://conserv.deri.ie/foaf:Person/abc123/cxaas:recentRelatedLocation |
| Request #2: HTTP GET, URL=http://example.org/cxaas:Detection/k1762 |

Listing 8.1: FR.1.1 Context Requests

The second example in the sequence diagram shows, the previously linked, context data being requested. The second request in Listing 8.1 shows the request sent to the remote node, and Figure 8.15 outlines the data returned for the request. The similarities between
the data in Figure 8.14 and Figure 8.15 are clear, the sole difference being that Figure 8.15 provides the ‘eventLocation’ rather than a ‘seeContext’ link.

Each of the three sub-scenarios discussed in the previous section, contained examples of interfaces for the provision of context data, and these examples are highlighted in their respective sequence diagrams. In both the sub-scenarios and the integrated sequence diagram example of this section, the interfaces described are RESTful interfaces. In total the sequences from scenario contains seven individual implemented examples of context being requested from third parties. These examples, which are similar to those described in this section, are sufficient to demonstrate the validity of FR.1.1.

**FR.1.2, Formally Described Interfaces**

ConServ provides WSDL descriptions, annotated with SA-WSDL, of the service interfaces it makes available. The WSDL files can be located by checking the node’s entry in CLS. Listing 8.2 provides a snippet of a SA-WSDL description of the interface for locating a user’s
The service portion of the WSDL file provides the service name and the location of the service. The binding in this example refers to the ‘getFriendsList’ operation within the ConServFriends’ interface. The ‘getFriendsList’ operation is located at ‘http://conserv.deri.ie/foaf:Person/\{PARENT\_ID\}/foaf:knows’ and is accessed with a HTTP Get request, where ‘\{PARENT\_ID\}’ is the local identifier of the parent user. The operation also has an annotated category which is specified with the ‘http://cxaas.org/ontologies/service\_categories#retrieveAcquaintances’ URI. This annotation enables interacting services to quickly locate the operation category they require.

The interface provides additional details regarding the ‘GetFriendsList’ operation. Elements from the conserv.xsd file are specified as the operation’s inputs and outputs, these elements are annotated with the appropriate properties from the CxaaS ontology in order provide some semantics to the inputs and outputs of the operation. Additionally, the ‘wsdl:output’ property of the operation is annotated with ‘http://xmlns.com/foaf/0.1/knows’, which indicates that the output of the operation is related to the object of the foaf:knows property.

Figure 8.14: FR.1.1: Interfaces Sends Redirect to the Requester

Figure 8.15: FR.1.1: Interfaces Sends Detection Event to Requester

friends\(^1\)

\(^1\)The full description file, including .xsd file, of this example can be found in the Appendix
Table 8.1 states that FR.1.2 can be validated by demonstrating ‘the processing of third-party interface descriptions’. The previous example demonstrated how an interfaces description is processed within the scenario and is sufficient to validate the functional requirement FR.1.2.

**FR.2.1, Identify New and Returning Users**

Within CxaaS, the lookup service provides functionality for identifying new and returning users. Listing 8.3 provides the request that is sent to the lookup service in order to identify the user. The URL for the HTTP GET request contains two parameters, the first is the identifier, which is a Bluetooth mac address, and the second is the type of the identifier, which is the URL-encoded URI ‘http://acronym.deri.org/schema#bluetooth’.

```
HTTP GET, URL= http://cxaas.org/cxaas:UserIdentifier/search?identifier=C0E4229DF2F8&identifierType=http\%3A\%2F\%2Facronym.deri.org\%2Fschema\%23bluetooth
```

Listing 8.3: FR.2.1 Lookup Service Request

If the lookup service identifier search successfully executes, the appropriate identifier resource is returned. Figure 8.16 provides an illustration of the identifier resource that is returned for this sequence. In addition to the ‘identifier’ and ‘identifierType’ properties, the resource includes the details of the associated user, which is connected via the ‘cxaas:identifiedUser’ property.

Each of the three sub-scenarios provide implementation-based examples of the identification of previously unknown users. These users may be new to the environment, or they may
be returning users who are using a new identifier, for instance, they may have purchased a new mobile phone. CxaaS operates in the same manner, as is outlined in this example, for both of these situations. Therefore, the CxaaS model is suitable for the identification of both new and returning users. CLS, and its use within the scenario, provides implementation-based validity to this claim.

FR.2.2, Discovery of Context Data

The integrated sequence diagram, Figure 8.13, provides an example of the discovery of context data. As can be seen from the figure, the discovery of context data covers multiple interactions, and it commences once the user has been identified. After user identification, the user’s home node must be identified. ‘Request #1’, of Listing 8.4, provides the request sent to CLS in the sequence. The request takes a single parameter, which is the URI of the person being searched for. The user resource returned by CLS is shown in Figure 8.17. The resource contains the URI of the user and the URI of the home node.


Listing 8.4: FR.2.2 Requests for the Discovery of Context Data

If the requesting ubicomp service does not already have the home node’s WSDL information, from a previous interaction, they must request this information from CLS. The request sent to CLS is given as ‘Request #2’ in Listing 8.4. It follows a similar pattern to the first request, where the search is done based on the service’s URI. The returned result is shown in Figure 8.18. The service property contains two properties; the name of the service and the location of the node’s WSDL file.
At this point in the sequence the user’s home node has been identified and location of the node’s WSDL file. The subsequent interactions relate to functional requirements FR.1.1 and FR.2.2, as is illustrated in the sequence diagram. The user’s home node’s WSDL file is requested and parsed by the requesting ubicomp service, and following this, the user’s last known location is requested. The request and the data returned were previously discussed in this section, and are shown in Listing 8.1 and Figure 8.14 respectively.

The home node redirects the requester to a remote third party node that has the required context data, by using the ‘cxaas:seeContext’ property. In order to request the context data from the remote node, the requester must parse its WSDL service description file. In the sequence illustrated in Figure 8.13, it is assumed that the requester is already aware of the location of the node’s WSDL file, based on a previous interaction, otherwise the requester would have to use CLS to discover the file’s location.

The final two interactions in this sequence concern the requesting of the context data from the third party node. Initially the WSDL file is requested and parsed, and subsequently the user’s ‘recentRelatedLocation’ is requested. The context request and details of the information returned were previously discussed in this section and are shown in Listing 8.1 and Figure 8.15 respectively.

The discovery of context data detailed in this section, involves interactions between four parties; the requesting node, CLS, the home node and a third party node. Four parties is the maximum number of parties that will be involved in the location of a piece of context.
Chapter 8. Evaluation

data. The context discovery example in the UbiCafé sub-scenario involves three parties; the requesting node, CLS and Donna’s home node. In situations where the home node is already known, and therefore CLS is not needed, then the discovery of context data can potentially involve only the requesting node and the user’s home node.

In total there are four examples of FR.2.2 across all of the sub-scenarios. These implementation sequences are similar to the sequence outlined in this section, however, the number of interacting parties varies across the sub-scenario examples. These implementation-based examples, in conjunction with the detailed example provided in this section, are sufficient to illustrate Cxaas’s support for discovery of context data (FR.2.2).

FR.2.3, Multiple Identification Mechanisms

There is only one example of the use of multiple identification mechanisms across all the sub-scenarios. However, this single example is sufficient to validate the applicability of the Cxaas model to the identification of users based on multiple identifiers. The reason for the low number of examples is due to hardware deployment limitations rather than any inherent issues with the Cxaas model.

The majority of the identification examples in the scenario are based on Bluetooth. Listing 8.3 provided the request to retrieve the Bluetooth-based identifier resource, while Figure 8.16 illustrated the data returned to the requester. The second type of identifiers used in the scenario are email addresses. Within the meeting sub-scenario, Smart Meetings uses email addresses to uniquely identify people attending meetings. In order to retrieve additional information concerning those users, Smart Meetings sends CLS a request for their Cxaas URI, that request is shown in Listing 8.5 and the response is shown in Figure 8.19.


Listing 8.5: FR.2.3 Lookup Service Email-based Identifier Request

These two identifier example show that the lookup service is capable of handling multiple identification mechanisms. The lookup service is not limited to supporting a particular set of identifiers, the only condition is that the combination of identifier and identifier type is unique.

FR.3.1, Distribution of Privacy Policies

An example of the distribution of privacy policies (FR.3.1) is provided across both the second and the third sub-scenarios. Within the exhibition sub-scenario, Donna’s privacy policy is downloaded to the exhibition’s ConServ node. When Smart Meetings requests Donna’s location history from the exhibition, it invokes the previously downloaded privacy policy.
8.5. Validating Requirements

The integrated interaction sequence from this section contains a similar policy distribution sequence. In order for the local ConServ node to be able to share the user’s context data it must enforce their privacy policies. The local ubicomp service downloads the user’s privacy policies from their home node, and then provides them to the local ConServ node.

At the beginning of this chapter, Table 8.1 described how implementations could be used to validate the functional requirements. It was stated that FR.3.2 could be validated through the distribution of a users privacy policy to a third-party node and subsequent enforcement of that policy. The combination of the two partial validations across the two sub-scenarios combines to a single complete validation of functional requirement FR.3.1, as was described in Table 8.1.

FR.3.2, Expressivity and Granularity

Table 8.1 outlined how FR.3.2 could be ‘demonstrated through the specification of privacy policy conditions within the scenario that contain context-based conditions and granularity-based outputs from the conditions’. The sub-scenarios contain multiple examples of expressivity and granularity of privacy policies, in particular the exhibition and follow-on meeting sub-scenarios.

Within the exhibition sub-scenario, the exhibition attendee, Donna, and the exhibitor, Walter, share contact details with others at the exhibition for the duration of the exhibition and share calendar details at a free/busy level of granularity. This privacy rule is contains two condition: (i) sharing with exhibition attendees and (ii) for the duration of the exhibition. The output of the privacy rule makes use of the granularity levels for calendar events by restricting access to a ‘free/busy’ level.

In the follow-on meeting, Donna shares her location history with Smart Meetings, for the duration of the meeting and at a ‘Building’ level of granularity, in order for them to customise the portfolio they show to Donna, according to her location history. This privacy rule also
contains two conditions: (i) it is for the duration of the meeting, and (ii) it is only with Smart Meetings. The output of the rule is restricted to locations at a level of granularity at, or coarser than, a ‘Building’ level.

These two implemented examples contain context-based conditions and granularity-based outputs. Thus, they demonstrate CxaaS’s support for expressivity and granularity within the privacy policies.

**FR.3.3, Multiple Policy Types**

It was stated in Table 8.1 that support for multiple policy types can be demonstrated through the enforcement of more than one policy type, for instance support for user and node privacy policies. Within the scenario there is one explicit use of multiple types of policies, this occurs in the exhibition sub-scenario.

Node policies are used within the exhibition to ensure the protection of the context data belonging to attendees whose privacy policies cannot be located. As their privacy rules do not exist or are not accessible, their privacy wishes cannot be enforced. The exhibition’s node policy forbids the exchange of this context data with third-parties.

Upon receipt of a request for context data, the exhibition’s ConServ loads the user’s policy, if one is available, and the node policy into the inference model simultaneously. The rules within the node policy are given a lower precedence than the user’s privacy policy rules, and therefore, the node rules are executed after the user’s rules.

The same approach could be adopted for enforcing legal policies. The use of precedence levels allows for easy execution of multiple policy types in a predefined order. This feature, and the implementation-based example of its use, illustrates CxaaS’s support for multiple policy types.

**FR.4.1: Implementation Agnostic**

The implementation agnosticism (FR.4.1) of the CxaaS model is clear from the minimal technological requirements specified. A context ontology, RIF-based privacy policy and WSDL-based interface descriptions are the sole technological requirements made within the CxaaS model. The removal of any of these technological requirements would result in the invalidation of other core and functional requirements. This demonstrates that the selection of technologies has been kept to a minimum and thus the implementation agnosticism requirement has been fulfilled to the maximum extent possible.

The MVC-based, RESTful CxaaS node implementation used in the scenario realisation could equally be replaced with a significantly different implementation, while adhering to the minimal technical requirements established earlier in this thesis. For instance, a node implementation, using SOAP web-services for inter-node communication, would still be able to use the service descriptions. Internally, the node could use an agent-based architecture,
such as was use in CoBrA (Chen et al., 2004a), for the provision of ubicomp services and the privacy-sensitive management of context data across its entire lifecycle.

**FR.4.2: Manage the Entire Lifecycle**

Chapter 5 established a lifecycle suitable for use in distributed, privacy-sensitive, context sharing deployments. ConServ’s support for this lifecycle was not explicitly highlighted in the scenario presented in this chapter. The following list, arranged according to the lifecycle stages, highlights ConServ’s support for the CxaaS lifecycle, and thus, provides an implementation example of how the CxaaS lifecycle can be integrated into CxaaS nodes:

- **Acquisition**: ConServ can receive context data by pulling it from context sources, or by context sources pushing the data to ConServ. Pushed data can either be manually entered, via the web-interface, or pushed to the RESTful interface. Context data can also be acquired by pulling data from a context source. Acquiring calendar data is a good example of this; the user provides the URL of their iCalendar file, this XML file is then imported by ConServ and the data in the iCal file is then provided to the classification stage of the lifecycle.

- **Classification**: Context classification occurs in two different ways depending on whether the context data is pulled or pushed. If the context data is pushed, then the URL which it was pushed to is decoded into a class and a property for a particular ontology, and then the data is converted into RDF, before being passed to the Context Handling stage of the lifecycle. If the context data is pulled, it is converted to RDF by the Context Model, then the individual pieces of context data are separated and forwarded to the Context Handling stage.

- **Handling**: ConServ materializes all appropriate location and event data in order to provide access to those concepts at different levels of granularity. The materialization of context data and its storage is handled by the ‘Context Model’ portion of ConServ’s architecture.

- **Dissemination**: The dissemination of context data begins with a request for context being sent by a service to the RESTful interface or from a user via their browser. Dissemination involves a significant number of components in the ConServ architecture, in particular: context interfaces, authentication, admin model, admin data store, context controller, context model policy execution and the context store. Privacy protection is a critical aspect of the dissemination stage. The policy execution component establishes the appropriate access rights for the requester.

- **Maintenance**: Context maintenance is not integrated into the current ConServ version as the duration of the scenario is too short (two days) to warrant archival or retention of context data.
Chapter 8. Evaluation

• **Disposition:** ConServ supports the disposition of context data via both the html and context interfaces and is handled by the deletion portion of ConServ’s CRUD functionality. The deletion of context data is restricted to the owner of the context data and services belonging to the ConServ node.

**FR.4.3, Policy Enforcement**

Table 8.1 states that: ‘Each time context data is shared with a third-party it results in a policy enforcement. FR.4.3 can be demonstrated by show examples of context data being requested and subsequent privacy policy enforcement, irrespective of whether the policies are of remote or local origin.’ The scenario contains multiple examples of the enforcement of privacy policies.

The discussion of FR.3.2, Expressivity and Granularity, earlier in this section, provided two examples of privacy policies being enforced. These two examples, the sharing of contact details during the exhibition and Donna sharing her location history, are implementation-based examples of privacy policies being enforced by Cxaas nodes. These implementation examples provide validation for the functional requirements FR.4.3.

The requesting of calendar details in the exhibition sub-scenario is another example of policy enforcement. Listing 8.6 is an example of a request for one of Donna’s calendar events. The rule in Donna’s privacy policy related to this example is shown in Listing 8.7. The rule limits access to Donna’s calendar events to a ‘free/busy’ level for the EcoHome Assistant service for the duration of the exhibition.

### Listing 8.6: FR.4.3 Request Location Information From Exhibition

```xml
Document(
  Prefix(rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(cxaas <http://cxaas.org/ontologies/cxaas#>)
  Prefix(event <http://purl.org/NET/c4dm/event.owl#>)
  Prefix(xs <http://www.w3.org/2001/XMLSchema#>)

Group( (* 'http://donnahomenode.com/cxaas:Policy/1'[cxaas:precedence->'10'
  ^ xs:integer]) *)

Forall ?AccessRequest ( 
  If And(
    ?AccessRequest # cxaas:AccessRequest 
    cxaas:expired(?AccessRequest,'false')
    cxaas:contextType(?AccessRequest, <http://cxaas.org/ontologies/cxaas#
      CalendarEvent>)
      cxaas:Service/1>)
    CurrentTime(?x),
```
8.5. Validating Requirements

| 19 | pred:dateTime - greater - than (?x, '2010-10-31T08:00:00'^^xsd:dateTime), |
| 20 | pred:dateTime - less - than (?x, '2010-10-31T18:00:00'^^xsd:dateTime), |
| 21 | Then Do( |
| 22 |   Assert(cxaas:GrantedAccessRequest(?AccessRequest)) |
| 24 | ) |

Listing 8.7: Donna’s Exhibition Calendar Access Rule

Functional Requirements Within the Scenario

As can be seen in Table 8.5, the scenario as a whole provides implementation-based validations of each of the functional requirements, with the exception of FR.4.1 and FR.4.2. Of the validated requirements, particular emphasis is placed on discovery and mobility (FR.2) and the exchange of context data (FR.1). Multiple implementation examples, and thus validations, have been provided for functional requirements FR.1.1, FR.1.2, FR.2.1 and FR.2.2.

The scenario contains a single example of one of the environments making their context data available to one of the other environments. For this reason, there is only one example of the use of multiple types of privacy policies. The single example of multiple policy types is not due to an issue with the CxaaS model, rather it is due to the structure of the scenario.

8.5.2 Validating Core Requirements

Section 8.2 outlined the overlap between the functional requirements and the core requirements. In particular, the overlap between (i) R.1 (Context Discovery) and the functional requirements FR.2.1 (Identification of new and returning users) and FR.2.2 (Discovery of context data about identified users), and (ii) R.2 (Context Sharing) and the functional requirements FR.1 (Exchange of Context Data) and FR.3 (Privacy Protection), were identified. A single core requirement, R.3 (Formal Context Model) was identified that could not be completely validated through the scenario implementation. This section discusses each of the core requirements, with particular emphasis placed on R.3 because it does not have a direct mapping to functional requirements.

R.1: Mechanisms for the discovery of new context data concerning new or returning users

It was identified in Section 8.2 that the discovery of context data concerning new and returning users is comprised of two primary parts: (i) the identification of users and (ii) the discovery of context data. These two facets map directly onto functional requirements FR.2.1
Chapter 8. Evaluation

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Sub-scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR.1</td>
<td>Exchange of Context Data</td>
<td></td>
</tr>
<tr>
<td>FR.1.1</td>
<td>Interfaces for the provision of context data to third-parties</td>
<td>● ● ●</td>
</tr>
<tr>
<td>FR.1.2</td>
<td>Formally described interfaces</td>
<td>● ● ●</td>
</tr>
<tr>
<td>FR.2</td>
<td>Discovery and Mobility</td>
<td></td>
</tr>
<tr>
<td>FR.2.1</td>
<td>Identification of new and returning users</td>
<td>● ● ●</td>
</tr>
<tr>
<td>FR.2.2</td>
<td>Discovery of context data about identified users</td>
<td>● ● ●</td>
</tr>
<tr>
<td>FR.2.3</td>
<td>Multiple identification mechanisms</td>
<td>●</td>
</tr>
<tr>
<td>FR.3</td>
<td>Privacy Protection</td>
<td></td>
</tr>
<tr>
<td>FR.3.1</td>
<td>Distribution of Privacy Policies</td>
<td>circ circ</td>
</tr>
<tr>
<td>FR.3.2</td>
<td>Expressivity and Granularity</td>
<td>● ●</td>
</tr>
<tr>
<td>FR.3.3</td>
<td>Multiple Policy Types</td>
<td>●</td>
</tr>
<tr>
<td>FR.4</td>
<td>Individual Nodes</td>
<td></td>
</tr>
<tr>
<td>FR.4.3</td>
<td>Policy Enforcement</td>
<td>● ●</td>
</tr>
</tbody>
</table>

Table 8.5: Requirements Validated by Entire Scenario

(Identification of new and returning users) and FR.2.2 (Discovery of context data about identified users). Through the previously demonstrated validation of functional requirements FR.2.1 and FR.2.2, the core requirement of context discovery (R.1) can also be considered validated.

R.2: Mechanism available for sharing context data in a controlled, privacy sensitive manner

Section 8.2 described how R.2 (mechanism available for sharing context data in a controlled, privacy sensitive, manner) comprises two facets: (i) context privacy protection, and (ii) exchange of context data. These two facets map onto the two high-level functional requirements FR.1 (Exchange of Context Data) and FR.3 (Privacy Protection). The previous section validated the functional requirements that comprise these high-level functional requirements. They are: FR.1.1 (Interface for the Provision of Context Data), FR.1.2 (Formally Described Interfaces), FR.3.1 (Distribution of Privacy Policies), FR.3.2 (Expressivity and Granularity) and FR.3.3 (Multiple Policy Types). The validation of functional requirements FR.1 and FR.3 also results in the validation of the core requirement R.2.

196
R.3: Formal context model suitable for the sharing of context data between third-parties, appropriate for use within the discovery and privacy mechanisms

The third core requirement necessitates ‘a formal context model suitable for the sharing of context data between third-parties, appropriate for use within the discovery and privacy mechanisms’. Chapter 6 delivered this required context model. The context types modelled within it were subsequently used within the privacy policies, Chapter 7, and the node interface descriptions, Section 6.6. Therefore, a formal context model has been developed which is theoretically suitable for the exchange of context data and for use in the discover and privacy mechanisms.

The scenario implementation provides a practical validation of R.3, in addition to the theoretical validation. The ontology was heavily used throughout the scenario implementation. The context data stored by ConServ uses the CxaaS ontology and the data exchanged between the nodes is annotated using the CxaaS ontology.

The previously validated functional requirements FR.1.2 (Formally described interfaces) and FR.3.2 (Expressivity and Granularity) also provide implementation-based credence to the validity R.3 within CxaaS. Each of the sub-scenarios provide examples of the use of formally described interfaces, whose descriptions make use of the CxaaS ontology. The exhibition and meeting sub-scenarios provide examples of the use of expressive privacy policies. This expressivity is enabled through the use of the CxaaS ontology within the policies. The combination of the theoretical and implementation demonstrations are sufficient to assert the validity of the third core requirement of the CxaaS model.

8.6 Summary

The purpose of this chapter was the validation of the CxaaS model. The validation process was separated into four parts. To begin with, the requirements established in Section 4.1 were analysed in order to identify which requirements could be evaluated through the implementation of the scenario and which could not. The setup of the experiment was then described and the common components of for the scenario instantiation were detailed. Each of the three sub-scenarios were then discussed, the component interactions shown and their associated requirements highlighted. Finally, each of the core and functional requirements were then individually discussed and their validations detailed.

A scenario-based implementation was chosen as the primary validation method as the scenario embodies the goals, expectations, challenges and motivations of the context sharing aspects of the original AmI and ubicomp scenarios. The updated, and refined, scenario is focused on a distributed setting where context data is exchanged between third-parties in a privacy-sensitive manner.

Included in this section was a description of two generic pieces of software (ConServ and CLS) and three scenario specific pieces of software (Barista Helper, EcoHome Assistant and
Smart Meetings). CLS is a RESTful CxaaS Lookup Service, while ConServ is a RESTful CxaaS node. CLS and ConServ were applied, in conjunction with scenario specific software, to each of the sub-scenarios.

The validation of the core and functional requirements provides credence to the applicability of the CxaaS model to the controlled sharing of context data between third-parties. Additionally, when compared with the core related work, as shown in Table 8.6, the advantages of the CxaaS model over the existing options become clear.

Section 3.4.2 identified three key criteria combinations: (i) C1.1 and C2.1, (ii) C3 and C5.1, and (iii) C2 and C5.1. Fire Eagle, UIF and CoBrA, respectively, supported individual sets of combined criteria. It can be seen from that updated table that CxaaS supports all of the combined criteria. This is significant as support for individual criteria, or individual combined criteria, is not sufficient to provide the controlled sharing of context data with heterogeneous third-parties. For instance, the ability to exchange a user’s context data with third-parties (C1.1) and enforce their privacy preferences (C2.1) is of limited benefit to ubicomp services if they cannot identify the user who has entered the environment, or find the location of their remote context data (C3). As the only approach to support all of the identified criteria, CxaaS is the most suited to the provision of the controlled sharing of context data with heterogeneous third-parties.
### 8.6 Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>C1: Exchange</th>
<th>C2: Privacy</th>
<th>C3: Discovery</th>
<th>C4: Manage</th>
<th>C5: Context Types</th>
<th>C6: Device Agnostic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1.1: 3rd Party</td>
<td>C1.2: Interfaces</td>
<td>C1.3: Location independent</td>
<td>C2.1: User Controlled</td>
<td>C2.2: Granularity</td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>C2</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>C3</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>C4</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>C5</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

**Table 8.6:** CxaaS's fulfilment of the comparison criteria

<table>
<thead>
<tr>
<th>Name</th>
<th>C1: Exchange</th>
<th>C2: Privacy</th>
<th>C3: Discovery</th>
<th>C4: Manage</th>
<th>C5: Context Types</th>
<th>C6: Device Agnostic</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoBrA</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Context Fabric</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Nexus (extended)</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Context Bridge</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>UIF</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Hydrogen</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Pervasive Web</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Fire Eagle</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Google Latitude</td>
<td>P</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IYOUIT</td>
<td>P</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

*N=no support, P=partial support, Y=full support, n/a=not applicable*
Chapter 9

Summary and Future Work

In traditional ubiquitous computing deployments, the environment that produces the context data is the sole consumer of that data. The environments are like stovepipes, having little integration beyond their isolated application scenarios. These isolated environments contrast significantly with the mobile nature of their users, where users freely move between environments. Users are continuously creating context data, regardless of the environment they are located in, and this context data can be distributed over a wide number of isolated ubicomp environments. Unfortunately, traditional deployments do not have the ability to retrieve this distributed context data.

This thesis addresses the issue of the sharing of context data between Context Management Services (CxMSs). A holistic approach to the issue was taken by considering the implications for individual Context Management Service (CxMS) nodes, the network of CxMS nodes and the users of those nodes. This comprehensive view on the issue resulted in the introduction of several challenges, in particular:

- **Mobility and Discovery**: How to identify new and returning users, and discover remote context data concerning those users?

- **Exchange of Context Between Third-Parties**: How to enable the sharing of context data between third-party CxMSs?

- **Context Privacy Protection**: The protection of the privacy of users context data is of the utmost importance. How could user’s privacy wishes be enforced in a distributed situation?

- **Formal Context Model**: How to provide a formal context model suitable for use with the discovery and privacy mechanisms?

A core set of related work was identified and analysed in Chapter 3. From this set, it was clear that none of the systems fully address the identified challenges. Two of the systems that provide functionality for sharing context data with third-parties do not provide privacy protection features. This discrepancy is a significant issue when dealing with highly sensitive
context data. The final system that supports third-party context sharing does not provide mobility functionality, or formal representations for the shared context data.

The approach described in this thesis adopts a web service-based approach to the controlled sharing of context data. Context as a Service (CxaaS) is comprised of three high-level segments: (i) the network architecture (CxaaS-Net), (ii) the node architecture (CxaaS-Node) and (iii) the data model (CxaaS-Data). These segments, and their relation to the identified challenges, are outlined in the following section.

9.1 Contributions

This thesis presents several contributions to the state of the art in the area of ubiquitous and context-aware computing. The overall contribution of this thesis is the definition of a web service-based model for the controlled sharing of context data between heterogeneous third-parties. The adoption of a web service-based approach maximizes the variety of potential interacting applications, due to the highly accessible nature of web-based services and applications. This benefits service providers by enabling access to context-data from domains beyond their usual operational boundaries. It is also of benefit to users, as it allows them to seamlessly move between ubiquitous environments, while being provided with increased levels of personalisation thanks to the environments access to additional information.

The overall contribution, CxaaS, contains several distinct sub-contributions, which are outlined in the following subsections:

9.1.1 CxaaS-Net

CxaaS-Net provides a model for transparent identification, discovery and acquisition of context data in a privacy sensitive manner from heterogeneous third-parties. The architectural model defines both the structure and the technological details for the discovery and sharing of context data between third-parties. CxaaS-Net moves ubiquitous computing beyond stovepipe-like deployments, to a collaborative distributed model.

CxaaS-Net is focused on the external interoperation of CxaaS nodes, in particular the architectural requirements for supporting the discovery and exchange of context data. It is comprised of a set of interconnected CxaaS nodes, sharing context data, and a lookup service to facilitate the mobility and discovery functionality that is central to CxaaS. The lookup service provides the methods for mapping identifiers to a URI representing that user across all CxaaS nodes.

9.1.2 CxaaS Lifecycle and CxaaS-Node

A context data lifecycle, based upon related work from the enterprise data lifecycles and ubiquitous computing lifecycles, was developed as part of this thesis. The existing offerings for the modelling of context data were overly simplistic and not of use beyond their own individual
applications. The new context lifecycle, which is focused on distributed deployments, is an integral part of the CxaaS-Node design. Through the support of such a lifecycle a consistent handling of context data, across its entire lifecycle, can be provided across nodes.

CxaaS-Node provides a node architecture that is focused on: (i) supporting the local ubi-comp environment, (ii) providing the required functionality behind the CxaaS-Net interfaces, and (iii) incorporating the CxaaS context lifecycle. The exact type of architecture (e.g. SOA, Agent-based, P2P) to be used internally within the environment is not constrained, however, externally the CxaaS node must adhere to the web service-based approach specified in this thesis.

9.1.3 The CxaaS Ontology and Granularity Model

The CxaaS ontology provides a commonly understood representation for information exchanged between the nodes. It defines a core set of commonly used context types and re-uses existing commonly used ontologies where possible. While the ontology has been designed with the CxaaS application in mind, it is sufficiently versatile to be used in other context aware applications.

The use of existing ontologies where possible, provides several benefits compared with traditional context ontologies: (i) existing sources of context data can be harnessed; (ii) the levels of semantics and understanding of the modelled concepts is increased through the connection with third-party ontologies; and (iii) increased levels of compatibility with existing ontologies. Additionally, the online publishing of the ontology, rather than just the textual description of the ontology in publications, makes integration with, and extension of, the ontology easier.

The CxaaS ontology also provides developers with a novel method for modelling granularity. Examples of its use are provided through the provision of granularity levels for location, calendar events and activities. The modelling of granularity has applications beyond the context awareness domain. Context types such as location, events and activities are used in many other domains, and the granularity model is appropriate for use should there be a need for providing different levels of accuracy.

9.1.4 CxaaS Privacy Policies

A flexible privacy policy structure was defined to model user’s privacy preferences. The privacy policy combines RIF rules with an OWL-based model for context access requests. This combination allows the specification of optimistic and pessimistic rules, provides rule precedence, and the granting of access at different levels of accuracy. By using RIF it allows the individual nodes to internally implement the rules in any rules language where an appropriate mapping exists. The privacy policies are sufficiently flexible that, in conjunction with the CxaaS Ontology, they could be used in non-CxaaS applications.
9.2 Future Work

I believe that the ubiquitous computing domain is approaching a critical point where applications and services must expand beyond the single-environment-scoped deployments. The lack of commercial success within the domain indicates that there is no appetite, within the general population, for ubicomp products and services in their current form. Ubicomp applications must increase their visibility and expand beyond the current limited operational boundaries. This thesis detailed an approach that complimented existing ubicomp deployments, however, recent developments from the social networking and smart phone domains could push the domain in an alternative direction.

9.2.1 Social Network Integration

Social networks complement the research work being undertaken in the ubicomp domain. They contain a large amount of context data, such as friend information, virtual interaction data and, increasingly, data on physical interactions. Through the use of smart phone applications, users can update their social networks with their physical location and those of their nearby friends.

The exact position of ubicomp applications within this picture is as yet unclear. Context data originating form social networks can be harnessed by traditional ubicomp applications, however, there is undoubtedly scope for more than this. One potential option would be the conversion of the social network account into a user’s individual CxMS, where existing context data from the social network can be combined with ubicomp environmental context data. Integration with a successful and established social network platform could address the adoption issues currently facing the ubicomp domain.

Unlike the approach proposed in this thesis, the social network-based CxMS disrupts the current context-ownership and context-control structure within the ubicomp domain, whereby the environment that creates the context has control and ownership over it. A solution to this issue would be the integration of a user’s CxaaS home node with their social network account, thereby maintaining the current context ownership and control structures, while providing significant social network integration.

9.2.2 Smart Phone Integration

The increasing power, flexibility, connectivity and capabilities of smart phones has enormous potential for ubicomp applications and services. The smart phones can provide a vast amount of context data, such as friend, location, calendar and movement data. They also support multiple communication channels, including WiFi, Bluetooth and 3G. The user friendly form of the smart phone, combined with its high levels of connectivity and access to significant amounts of context data, could replace much of the functionality currently offered by ubicomp services.
The challenge for the ubiquitous computing domain is how to integrate with this potentially disruptive technology, while building on existing ubicomp research. Successful integration with the smart phone could lead to increased visibility, awareness and trust of ubicomp applications and services by the general public. One potential solution is the deployment of user’s home nodes onto their smart phones. This would enable the privacy-sensitive provision of context data, originating from the smart phone, to ubicomp environments, while maintaining the current ownership and control structures. There may also be some integration opportunities between the smart phone-based home node and the social network-based home node.

9.2.3 Web Application Personalisation

While some ubicomp services may make use of data, such as weather data, originating from third-party web applications, context data originating from ubicomp environments does not get used by third-party web applications. Web applications could be personalised based on context data from ubicomp environments, such as information on the building they are located in, the people they are with, the place they have just come from. Such functionality would increase the range, impact and operational scope of ubicomp services. It would also provide an additional incentive for people to use ubicomp services.

A CxaaS-based approach could be used to provide such functionality. The primary challenge in this situation is the deployment of a critical mass of CxaaS nodes from which sufficient context information could be gathered.

9.3 Summary

The controlled sharing of context data between heterogeneous third-parties, through the use of the CxaaS model, can provide significant benefits to both ubicomp environments and the users of those environments. Access to new and diverse types of context data allows environments to offer enhanced services, with increased personalisation, to users and react to events that occurred outside of their domain of operation.

From a user’s perspective, the adoption of the CxaaS model by ubicomp services enables increased personalisation and lower barriers for interacting with the services. The holistic approach taken within this thesis combats the typical user concerns regarding the sharing of their data, with particular emphasis on the privacy of their context data.
Bibliography

Brightkite. URL http://brightkite.com/
Foursquare. URL http://foursquare.com/
Friends on fire. URL http://apps.facebook.com/on-fire/
Google latitude. URL http://www.google.com/latitude/
Gowalla. URL http://gowalla.com/
The hot list. URL http://www.thehotlist.com/
Iyouit. URL http://www.iyouit.eu/
Locle. URL http://www.locle.com/
Loopt. URL http://www.loopt.com/
Mapme. URL http://mapme.at/
Mit project oxygen. URL http://oxygen.csail.mit.edu/Overview.html
Nike + ipod. URL http://www.apple.com/ipod/nike/
Rummble. URL http://www.rummble.com/
Skyhook wireless. URL www.skyhookwireless.com
Smheart link. URL http://www.smheartlink.com/products.html
Spurty chest strap. URL http://www.mobimotion.com/index.php?id=brustgurt&L=1
Whrrl. URL http://whrrl.com/
Yahoo! fire eagle. URL http://fireeagle.yahoo.net/
Yahoo! geoplanet. URL http://developer.yahoo.com/geo

Bibliography


Bibliography


210


A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to context than location, 1999. ISSN 0097-8493. URL http://www.sciencedirect.com/science/article/B6TYG-3YTBV04-M/2/e3187a15abb0c1a22b257ddd73c3fd3b.


W3C. Delivery context: Client interfaces (dcci), July 2007. URL http://www.w3.org/TR/DPF/.


220


Appendix

Fundamental Scenarios

The Fundamental UbiComp Scenario Developed by Mark Weiser

Sal awakens; she smells coffee. A few minutes ago her alarm clock, alerted by her restless rolling before waking, had quietly asked, "Coffee?" and she had mumbled, "Yes." "Yes" and "no" are the only words it knows.

Sal looks out her windows at her neighborhood. Sunlight and a fence are visible through one, and through others she sees electronic trails that have been kept for her of neighbors coming and going during the early morning. Privacy conventions and practical data rates prevent displaying video footage, but time markers and electronic tracks on the neighborhood map let Sal feel cozy in her street.

Glancing at the windows to her kids’ rooms, she can see that they got up 15 and 20 minutes ago and are already in the kitchen. Noticing that she is up, they start making more noise.

At breakfast Sal reads the news. She still prefers the paper form, as do most people. She spots an interesting quote from a columnist in the business section. She wipes her pen over the newspaper’s name, date, section and page number and then circles the quote. The pen sends a message to the paper, which transmits the quote to her office.

Electronic mail arrives from the company that made her garage door opener. She had lost the instruction manual and asked them for help. They have sent her a new manual and also something unexpected: a way to find the old one. According to the note, she can press a code into the opener and the missing manual will find itself. In the garage, she tracks a beeping noise to where the oil-stained manual had fallen behind some boxes. Sure enough, there is the tiny tab the manufacturer had affixed in the cover to try to avoid E-mail requests like her own.

On the way to work Sal glances in the foreview mirror to check the traffic. She spots a slowdown ahead and also notices on a side street the telltale green in the foreview of a food shop, and a new one at that. She decides to take the next exit and get a cup of coffee while avoiding the jam.

Once Sal arrives at work, the foreview helps her find a parking spot quickly. As she walks into the building, the machines in her office prepare to log her in but do not complete the
sequence until she actually enters her office. On her way, she stops by the offices of four or five colleagues to exchange greetings and news.

Sal glances out her windows: a grayday in Silicon Valley, 75 percent humidity and 40 percent chance of afternoon showers; meanwhile it has been a quiet morning at the East Coast office. Usually the activity indicator shows at least one spontaneous, urgent meeting by now. She chooses not to shift the window on the home office back three hours – too much chance of being caught by surprise. But she knows others who do, usually people who never get a call from the East but just want to feel involved.

The telltale by the door that Sal programmed her first day on the job is blinking: fresh coffee. She heads for the coffee machine.

Coming back to her office, Sal picks up a tab and ”waves” it to her friend Joe in the design group, with whom she has a joint assignment. They are sharing a virtual office for a few weeks. The sharing can take many forms – in this case, the two have given each other access to their location detectors and to each other’s screen contents and location. Sal chooses to keep miniature versions of all Joe’s tabs and pads in view and three-dimensionally correct in a little suite of tabs in the back corner of her desk. She can’t see what anything says, but she feels more in touch with his work when noticing the displays change out of the corner of her eye, and she can easily enlarge anything if necessary.

A blank tab on Sal’s desk beeps and displays the word ”Joe” on it. She picks it up and gestures with it toward her live board. Joe wants to discuss a document with her, and now it shows up on the wall as she hears Joe’s voice: ”I’ve been wrestling with this third paragraph all morning, and it still has the wrong tone. Would you mind reading it?” Sitting back and reading the paragraph, Sal wants to point to a word. She gestures again with the ”Joe” tab onto a nearby pad and then uses the stylus to circle the word she wants: ”I think it’s this term ’ubiquitous.’ It’s just not in common enough use and makes the whole passage sound a little formal. Can we rephrase the sentence to get rid of it?” ”I’ll try that. Say, by the way, Sal, did you ever hear from Mary Hausdorf?” ”No. Who’s that?” ”You remember. She was at the meeting last week. She told me she was going to get in touch with you.” Sal doesn’t remember Mary, but she does vaguely remember the meeting. She quickly starts a search for meetings held during the past two weeks with more than six people not previously in meetings with her and finds the one. The attendees’ names pop up, and she sees Mary.

As is common in meetings, Mary made some biographical information about herself available to the other attendees, and Sal sees some common background. She’ll just send Mary a note and see what’s up. Sal is glad Mary did not make the biography available only during the time of the meeting, as many people do...

A Fundamental Ambient Intelligence Scenario

After a tiring long haul flight Maria passes through the arrivals hall of an airport in a Far Eastern country. She is travelling light, hand baggage only. When she comes to this particular
country she knows that she can travel much lighter than less than a decade ago, when she had to carry a collection of different so-called personal computing devices (laptop PC, mobile phone, electronic organisers and sometimes beamers and printers). Her computing system for this trip is reduced to one highly personalised communications device, her PCom that she wears on her wrist. A particular feature of this trip is that the country that Maria is visiting has since the previous year embarked on an ambitious ambient intelligence infrastructure programme. Thus her visa for the trip was self-arranged and she is able to stroll through immigration without stopping because her P-Comm is dealing with the ID checks as she walks.

A rented car has been reserved for her and is waiting in an earmarked bay. The car opens as she approaches. It starts at the press of a button: she doesn't need a key. She still has to drive the car but she is supported in her journey downtown to the conference centre-hotel by the traffic guidance system that had been launched by the city government as part of the AmI-Nation initiative two years earlier. Downtown traffic has been a legendary nightmare in this city for many years, and draconian steps were taken to limit access to the city centre. But Maria has priority access rights into the central cordon because she has a reservation in the car park of the hotel. Central access however comes at a premium price, in Maria's case it is embedded in a deal negotiated between her personal agent and the transaction agents of the car-rental and hotel chains. Her firm operates centralised billing for these expenses and uses its purchasing power to gain access at attractive rates. Such preferential treatment for affluent foreigners was highly contentious at the time of the introduction of the route pricing system and the government was forced to hypothecate funds from the tolling system to the public transport infrastructure in return. In the car Maria's teenage daughter comes through on the audio system. Amanda has detected from En Casa system at home that her mother is in a place that supports direct voice contact. However, even with all the route guidance support Maria wants to concentrate on her driving and says that she will call back from the hotel.

Maria is directed to a parking slot in the underground garage of the newly constructed building of the Smar-tel Chain. She is met in the garage by the porter - the first contact with a real human in our story so far! He helps her with her luggage to her room. Her room adopts her personality as she enters. The room temperature, default lighting and a range of video and music choices are displayed on the video wall. She needs to make some changes to her presentation - a sales pitch that will be used as the basis for a negotiation later in the day. Using voice commands she adjusts the light levels and commands a bath. Then she calls up her daughter on the video wall, while talking she uses a traditional remote control system to browse through a set of webcast local news bulletins from back home that her daughter tells her about. They watch them together. Later on she localises her presentation with the help of an agent that is specialised in advising on local preferences (colour schemes, the use of language). She stores the presentation on the secure server at headquarters back in Europe. In the hotel's seminar room where the sales pitch is take place, she will be able to call down
an encrypted version of the presentation and give it a post presentation decrypt life of 1.5 minutes. She goes downstairs to make her presentation this for her is a high stress event. Not only is she performing alone for the first time, the clients concerned are well known to be tough players. Still, she doesn't actually have to close the deal this time. As she enters the meeting she raises communications access thresholds to block out anything but red-level emergency messages. The meeting is rough, but she feels it was a success. Coming out of the meeting she lowers the communication barriers again and picks up a number of amber level communications including one from her cardio-monitor warning her to take some rest now. The day has been long and stressing. She needs to chill out with a little meditation and medication. For Maria the meditation is a concert on the video wall and the medication a large gin and tonic from her room's minibar.

SAWSDL Annotated RESTful WSDL Service Description Examples

General Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl"
    targetNamespace="http://conserv.deri.ie/conserv:Detection/wsdl"
    xmlns:tns="http://conserv.deri.ie/conserv:Detection/wsdl"
    xmlns:whttp="http://www.w3.org/ns/wsdl/http"
    xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    <wsdl:types>
            <xs:element name="detection">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="service_uri" type="anyURI" sawsdl:modelReference="http://conserv.deri.ie/ontology#Service"/>
                        <xs:element name="device_uri" type="anyURI" sawsdl:modelReference="http://acronym.deri.org/schema#WirelessDevice"/>
                        <xs:element name="dtstart" type="dateTime" sawsdl:modelReference="http://conserv.deri.ie/ontology#dtstart"/>
                        <xs:element name="dtend" type="dateTime" sawsdl:modelReference="http://conserv.deri.ie/ontology#dtend"/>
                        <xs:element name="wireless_mac" type="string" sawsdl:modelReference="http://acronym.deri.org/schema#mac"/>
                        <xs:element name="bluetooth_mac" type="string" sawsdl:modelReference="http://acronym.deri.org/schema#bluetooth"/>
                        <xs:element name="rfid" type="string" sawsdl:modelReference="http://acronym.deri.org/schema#rfid"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
    </wsdl:types>
</wsdl:description>
```
Listing: Service Description Example

Evaluation Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl"


targetNamespace="http://conserv.deri.ie/wsdl/conserv.wsdl"

xmlns:tns="http://conserv.deri.ie/wsdl/conserv.wsdl"

xmlns:http="http://www.w3.org/ns/wsdl/http"

xmlns:xs="http://www.w3.org/2001/XMLSchema"

xmlns:sawsdl="http://www.w3.org/ns/sawsdl"

xmlns:msg="http://conserv.deri.ie/wsdl/conserv.xsd">

<wsdl:documentation>

This is a WSDL 2.0 description of a the ConServ context management service

</wsdl:documentation>
```
Listing: A Section of WSDL for ConServ

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://conserv.deri.ie/wsdl/conserv.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://conserv.deri.ie/wsdl/conserv.xsd"
    xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
    elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:element name="User" sawsdl:modelReference="http://xmlns.com/foaf/0.1/Person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="uri" type="xs:anyURI"/>
        <xs:element name="username" type="xs:string"/>
        <xs:element name="email" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="userList" type="tns:userListType"/>
  <xs:complexType name="userListType">
    <xs:sequence>
      <xs:element name="user" type="tns:User" minOccurs="0" maxOccurs="unbounded" sawsdl:modelReference="http://xmlns.com/foaf/0.1/Person"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
Scenario Implementation Details

Chapter 8 provided the majority of the details of the scenario instantiations. However, some of the implementation details were omitted from that chapter in order to focus on the architectural and interactional aspects of the instantiations. The following section provides some specific implementation details and screenshots from the implementations.

Each of the ubicomp services (Barista Helper, EcoHome Assistant and SmartMeetings) are developed using Ruby on Rails (Hansson, 2004). The CxaaS Lookup Service (CLS) and ConServ are both developed using the Grails (Rocher and Brown, 2009). Both Ruby on Rails and Grails are open source web application development frameworks. Ruby on Rails is a Ruby-based (Flanagan and Matsumoto, 2008) framework, while Grails is based on Groovy (Koenig et al., 2007), an object-oriented and dynamic language for the java platform.

Common Implementation Details

The positioning mechanisms used in the sub-scenarios, and the structure of the data exchanged between ConServ nodes, is common across all of the sub-scenarios.

Positioning: Bluetooth is used as the primary device-based identification mechanism within the implementations as the majority of mobile phones currently provide Bluetooth functionality. Therefore, users are not required to carry any additional devices in order to interact with the services and the hardware requirements for a deployment are minimal.

For the follow-on meeting, a mobile phone-based location updater is used to update Donna’s location while she is not in any smart space, and a Bluetooth-based indoor localisation for location tracking within the timber-frame manufacturer’s office.
Communication and Data Format: The scenario implementation uses XML as the format of the context exchanged between nodes. Semantics are provided for the exchanged data through the use of SA-WSDL annotations for the WSDL files describing the exchanged data.

HTML representations of the data, for browser-based interactions, are available for all of the components. Sending a HTTP GET request to a particular resource’s URL will provide the HTML representation by default. Access to XML representation of the resource data is possible through the specification of the content-type as text/xml in the HTTP request header, or by appending ‘.xml’ to the request URL.

HTTP-based RESTful communication is used throughout the scenario instantiation. This includes communication with remote nodes and the ubicomp services communicating with their local ConServ node. A RESTful, HTTP-based, approach is adopted in order to maximize the variety of interacting applications.

Authentication: HTTP basic access authentication is used, where necessary, in the ConServ nodes, CLS and ubicomp services. The authentication is handled by the security components of the requested service.

ConServ

Formats: The Grails framework allows the outputting of data as HTML, JSON and XML, while the Jena framework supports the outputting of RDF data in the form of RDF/XML, N-Triple, Turtle and N3. By providing multiple output formats ConServ ensures that it is able to inter-operate with as many applications and services as possible, regardless of their use of semantics. For the purpose of the evaluation the focus was on the HTML and XML outputs.

Privacy: Currently, the ConServ rules implementation uses Jena rules to represent the user’s privacy policies. This differs from the policy representation specified in the CxaaS model which requires the use of RIF-based policies. Currently, there is no mapping between Jena rules and RIF, however, this issue will be addressed, as this mapping is planned for a future Jena release (Reynolds, 2010, 2009).

Context Storage: The Jena framework (McBride, 2002) is used to manage the RDF graphs. The RDF data and basic website interaction data are separated in order for ConServ to function as efficiently as possible. After registration, users can provide a variety of personal information such as their FOAF profile, iCal calendar location and details of any portable devices they have (e.g. phone bluetooth mac-address, RFID).
Implementation Screenshots

This section provides eight screenshots of the components used in the scenario instantiation. The first two screenshots, Figure 1 and Figure 2, show user’s web interface on CLS containing identifier details and service details respectively.

Figure 1: CxaaS Identifier Profile Screenshot

Figure 2: CxaaS Service Profile Screenshot

Figure 3 shows a screenshot of a page providing the details available on user ‘gearoid’ on ConServ node http://conseve.deri.ie. Figure 4 provides a screenshot of a user’s privacy policy containing three rules, one rule restricting location requests, one rule restricting event requests and a final rule which denies any unhandled requests by default.
**Appendix**

---

**Figure 3:** Details of user ‘gearoid’ on ConServ node http://conserv.deri.ie

**Figure 4:** Listing of a privacy policy’s rules, listed according to precedence.

---

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Precedence</th>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LocationGranularityTest</td>
<td>1</td>
<td>Restricts service to accessing locations at a granularity</td>
<td>@prefix conserv: <a href="http://conserv.deri.ie/ontologies#">http://conserv.deri.ie/ontologies#</a> .</td>
</tr>
</tbody>
</table>
Figures 5 and 6 provide screenshots of the Barista Helper web-application. The first screenshot is of the application’s homepage, from where staff can access information on devices nearby, a list of customers, the café’s menu and a list of historical purchases (visits) of customers. The second screenshot shows the ‘Devices Nearby’ page, which has three devices in the café at that time. Two of the devices do not have associated entries in CLS while the third one does.

![UbiCafé’s Barista Helper Homepage](image1.png)

Figure 5: UbiCafé’s Barista Helper Homepage

![Listing devices](image2.png)

Figure 6: UbiCafé’s Barista Helper Nearby Device List

Figures 7 and 9 show screenshots from the EcoHome Assistant and Smart Meetings respectively. The EcoHome Assistant screenshot shows exhibitor information from the web-application. This includes the category of the exhibitor, their stand location, their last
detected location, and time of that location, within the exhibition and links that can be followed in order to get the exhibitor’s contact details and schedule a follow-on meeting with them.

The smart meetings screenshot shows a webpage providing the most recent location of an attendee, Donna, of a meeting scheduled for one of the meeting rooms of the timber frame manufacturer. In the scenario instantiation, this information is provided by a mobile phone application that constantly updates Donna’s location. A screenshot for this mobile phone application, called PoFriend, is provided in Figure 8.

Figure 7: EcoHome Assistant Exhibitor Information

Figure 8: PoFriend Screenshot
Figure 9: Walter searches for meeting attendee Donna using “Smart Meetings”