A Context-Aware Access Control Framework for Software Services

by

A. S. M. Kayes

A thesis submitted to

School of Software and Electrical Engineering
Faculty of Science, Engineering and Technology
Swinburne University of Technology
Melbourne, Australia

for the degree of

Doctor of Philosophy

2014
Abstract

Context-awareness is an important aspect of the dynamically changing environments, where users typically access resources (information, services, etc.) in an anywhere-anytime fashion. Access control in such environments needs to be dynamic and context-aware, taking into account the relevant contextual information. By leveraging the dynamically changing contextual information, we can achieve context-specific control over access to information resources and services, better satisfying the security and privacy requirements of the stakeholders. Therefore, there is the increasing need for new access control approaches to link their decision making abilities with context-awareness in recent years.

The basic role-based access control (RBAC) approach has become the most widely used access control model. It typically evaluates access permission through roles assigned to users. On the other hand, the attribute-based access control (ABAC) approach grants access permissions to resources based on the attributes of relevant entities. More recently, some context-aware access control approaches have been introduced and they extend the basic RBAC and ABAC approaches with specific types of dynamic contextual information. However, these approaches and their associated context models are not adequate to capture and reason about all the different types of contextual information such as the information characterizing the relationships between entities. Existing situation-aware access control approaches grant access to resources depending on the specific combinations of certain contextual information. However, they do not consider the purpose or user’s intention in accessing the resources, which is another important aspect. Furthermore, the existing access control policy models do not have adequate functionalities to incorporate diverse context and situation information into user-role and role-permission assignments for dynamic access control decision making.

In order to address the above mentioned issues, this thesis introduces a novel framework for context-aware access control, CAAC. It makes several important research contributions, including a new context-aware access control approach and the associated enabling techniques. Firstly, our new access control approach adopts and extends the basic
RBAC approach with dynamically changing contextual information. It introduces the dynamic context-awareness capabilities into both user-role and role-permission assignments. **Secondly**, the framework provides an ontology-based context model, including an upper ontology to represent the access control-specific core concepts and the domain ontologies specializing the core concepts for the representation of domain-specific concepts. The context model captures the different types of context entities and context information relevant to access control. **Thirdly**, the framework provides an ontology-based situation model to represent the purpose-oriented atomic situations and reason about composite situations. The situation model captures the states of the context entities, the states of the relationship between entities and the purpose or user intention, by using the context information from the context model. **Fourthly**, the framework provides an ontology-based model for specifying context-aware access control policies in terms of dynamic context-aware user-role assignment and role-permission assignment. The policy model uses the relevant contextual information from the context and situation models that reflects the dynamically changing conditions of the environments.

A prototype implementation has been provided for the CAAC framework to support the development of context-aware access control applications. In addition, an example case study has been carried out to demonstrate the applicability of the framework in a healthcare environment. Furthermore, a range of experiments have been performed to evaluate the feasibility of the CAAC framework. Overall, this research contributes to improving the security and privacy of access control systems in dynamic context-aware environments. It allows software engineers to model context and purpose-oriented situation information specific to access control, specify context-aware user-role and role-permission assignment policies and thus provides effective support for building context-aware access control applications.
Acknowledgements

I am deeply grateful for the extensive support and inspiration I have had the privilege of enjoying, from a whole host of people. I would like to express my earnest gratitude to my principal supervisor Professor Jun Han. It is he who led me to the entrance to scientific research of Context-Aware Software Systems. His insightful suggestions, generous encouragements and considerate instructions accompanied me along the way of my PhD study. I have learned many things from him for becoming an independent researcher in my field. Through countless hours and endless patience, he taught me how to do research, from the very start as to find a problem to a higher stage as to write a paper, and finally build a solution and accomplish this thesis. I would thank my associate supervisor Dr Alan Colman who helped me a lot to understand the field of Access Control. He is an experienced scientific supervisor, and a life guider as well.

I am deeply thankful to Dr Saiful Islam and Dr Samiran Mahmud for many fruitful and inspiring discussions that influenced the outcome of this thesis. I would like to thank and appreciate other members in our group at Swinburne particularly Dr Adeel Talib, Dr Malinda Kaushalye Kapuruge, Dr Tuan Nguyen, Dr Ashad Kabir, Dr Mahmoud Hussein, Dr Tharindu Patikirikora, Dr Ayman A. Amin and Mr. Indika Kumara for their support and encouragement. In addition, I thankfully acknowledge my country mates at Swinburne particularly Dr Shimul Nath, Dr Tanay Paul, Mr. Shibli Saleheen and Mr. Arup Sarker for their invaluable help and assistance over the years.

I would like to show my appreciation to my research progress review panels Professor Yun Yang, Dr Jean-Guy Schneider and Dr Wei Lai and external reviewers for their feedbacks and comments. I would also like to thank my thesis examiners for their recommendation and penetrating questions. They helped me a number of points to clarify my arguments, for which I am grateful. I am indebted to Swinburne University of Technology for creating a wonderful environment for intellectual exploration and discovery. I would like to remember Chittagong University of Engineering and Technology where I have built my foundation in Computer Science and Engineering.
I should specially acknowledge my wife, Fatma Mohammed who has sacrificed her career and accompanied me through all this period of pursuing my PhD. Her considerate inspirations and blind faith have been the infinite power of my step forwards. Last but not least, I would like to show my everlasting thanks to my family members, especially to my parents for their endless love and prayers to support me and for understanding my years away from their sides. This thesis is all dedicated to my parents!
Declaration of Authorship

This is to certify that,

- this thesis contains no material which has been accepted for the award of any other degree or diploma, except where due reference is made in the text of the examinable outcome; and

- to the best of my knowledge contains no material previously published or written by another person except where due reference is made in the text of the thesis; and

- where the work is based on joint research or publications, discloses the relative contributions of the respective workers or authors.

________________________
A. S. M. Kayes
September, 2014
Melbourne, Australia
Imagination is more important than knowledge.

- Albert Einstein
List of Publications

Part of the materials of this thesis has been published as follows:

Conference Papers:


**Journals:**


**Posters:**


Contents

Part I  1

1 Introduction  3
  1.1 Context-Aware Access Control  4
      1.1.1 Access Control  4
      1.1.2 Context-Aware Access Control  5
      1.1.3 Discussion  9
  1.2 Research Issues  10
  1.3 An Approach to Context-Aware Access Control for Software Services  11
  1.4 Research Contributions  13
  1.5 Thesis Outline  15

2 Research Motivation  19
  2.1 Application Scenario  19
  2.2 Scenario Analysis  22
  2.3 Requirements for Developing CAAC Applications  25
  2.4 Summary  28

3 Background and Literature Review  29
  3.1 Access Control  30
      3.1.1 Access Control History  30
      3.1.2 Access Control Matrix  31
      3.1.3 Access Control List and Capability-Based Access Control  32
      3.1.4 Role-Based Access Control  33
      3.1.5 Attribute-Based Access Control  34
# Contents

3.1.6 Discussion .................................................. 35

3.2 Context-Aware Role-Based Access Control .................................. 37

  3.2.1 Temporal Role-Based Access Control .................................. 37

  3.2.2 Spatial Role-Based Access Control .................................. 38

  3.2.3 Spatial and Temporal Role-Based Access Control ..................... 38

  3.2.4 User-Specific Context and Role-Based Access Control .............. 39

  3.2.5 Contextual-Role and Role-Based Access Control .................... 40

  3.2.6 Team-Based Access Control ....................................... 41

  3.2.7 Organization-Based Access Control .................................. 41

  3.2.8 Context-Specific Role-Based Access Control ....................... 42

  3.2.9 Comparative Analysis and Discussion .................................. 44

3.3 Context-Aware Attribute-Based Access Control ................................. 46

  3.3.1 Comparative Analysis and Discussion .................................. 49

3.4 Situation-Aware Access Control ............................................. 50

  3.4.1 Comparative Analysis and Discussion .................................. 52

3.5 Context Modelling, Reasoning and Management ................................ 53

  3.5.1 Discussion ......................................................... 55

3.6 Overall Discussion ..................................................... 55

3.7 Summary ............................................................... 57

## Part II

4 Conceptual Framework for Context-Aware Access Control ...................... 61

  4.1 Background ............................................................ 62

  4.2 The Conceptual Framework for CAAC ..................................... 63

  4.3 Basic Elements of CAAC Framework ...................................... 67
## Contents

4.3.1 Context Information ........................................ 67
4.3.2 Situation Information ..................................... 68
4.3.3 Context Specification Language ................................. 70
4.3.4 Role Hierarchy .............................................. 73
4.3.5 Resource Hierarchy ........................................... 74
4.3.6 CAAC Policy Specification .................................. 76

4.4 Summary .......................................................... 80

5 Context Model for CAAC ......................................... 81

5.1 Defining Context for CAAC ...................................... 82
5.2 Design Considerations for the Context Model .................. 85
5.3 Representing COAC Core Concepts ............................... 88
  5.3.1 Core Context Entities ...................................... 89
  5.3.2 Core Context Information .................................. 92
5.4 Representing COAC Domain-Specific Concepts ................. 99
  5.4.1 Domain-Specific Context Entities .......................... 99
  5.4.2 Domain-Specific Context Information ...................... 105
5.5 Reasoning About Contexts ..................................... 107
  5.5.1 Example Rules ............................................... 108
5.6 Summary .......................................................... 112

6 Situation Model for CAAC ......................................... 115

6.1 Defining Situation for CAAC ................................. 116
  6.1.1 Inferring Purpose .......................................... 116
  6.1.2 Atomic Situation ........................................... 117
  6.1.3 Composite Situation ....................................... 118
## Contents

6.2 Ontology-Based Situation Model ........................................ 120
   6.2.1 Core Concepts .................................................. 123
   6.2.2 Modelling and Inferring Atomic Situation ....................... 125
   6.2.3 Inferring Purpose and Composite Situation ..................... 126
6.3 Summary .............................................................. 130

7 Policy Model for CAAC .................................................... 131
   7.1 Defining CAAC Policy ............................................... 131
      7.1.1 CAURA Policy ................................................ 132
      7.1.2 CARPA Policy ............................................... 134
   7.2 Ontology-Based Policy Model ....................................... 135
      7.2.1 Policy Specification .......................................... 136
      7.2.2 Example Policies ............................................ 142
   7.3 Summary .............................................................. 148

Part III ................................................................. 149

8 Implementation of the CAAC Framework .................................. 151
   8.1 Design and Implementation of the CAAC Framework .................. 151
      8.1.1 System Architecture of the Prototype Framework ............. 152
      8.1.2 Data Flow Sequences of the Prototype Components ............ 154
   8.2 Technology Used to Implement Prototype Components ............... 155
      8.2.1 Ontology KB ................................................ 156
      8.2.2 Rule Base .................................................. 158
      8.2.3 Context Manager ............................................. 160
      8.2.4 Situation Manager .......................................... 161
      8.2.5 Access Control Manager .................................... 162
## Contents

8.2.6 Communication Among Software Components ........................................ 162

8.3 Application: Patient Medical Records Management (PMRM) System ............... 163

8.4 Summary ..................................................... 164

9 Evaluation of the CAAC Framework ......................................................... 167

9.1 CAAC Application: Patient Medical Records Management .......................... 167

9.2 Case Study: Patient Medical Records Management .................................... 169

9.2.1 Incorporating Context Information into CAAC Policy ............................ 170

9.2.2 Incorporating Situation Information into CAAC Policy .......................... 177

9.3 Performance Evaluation ................................................................. 179

9.3.1 Experimental Setting I: Incorporating Context Information into CAAC Policy ............................................. 179

9.3.2 Experimental Setting II: Incorporating Situation Information into CAAC Policy ............................................. 184

9.3.3 Overall Discussion .................................................... 188

9.4 Comparative Assessment .............................................................. 189

9.4.1 Key Features of Our CAAC Framework .......................................... 189

9.4.2 Comparative Analysis .................................................... 190

9.5 Summary ..................................................... 194

10 Conclusion and Future Work .............................................................. 195

10.1 Contributions ..................................................... 195

10.2 Future Work ..................................................... 199

Bibliography .................................................................................. 203
List of Figures

1.1 A High-Level Approach to Context-Aware Access Control 12
2.1 The Relationship Chain From User to Resource Access 25
3.1 Role-Based Access Control 34
4.1 Core Context-Aware Access Control Model 63
4.2 Example of Role Hierarchy 73
4.3 Example of Resource Hierarchy 74
5.1 The COAC Upper Context Ontology 89
5.2 The COAC Upper Context Ontology: Context Entities 89
5.3 The COAC Upper Context Ontology: Context Information 92
5.4 Graphical Representation of the RelationshipInfo Class 94
5.5 Graphical Representation of the StatusInfo Class 95
5.6 Graphical Representation of the ProfileInfo Class 96
5.7 Graphical Representation of the LocationInfo Class 97
5.8 Graphical Representation of the TemporalInfo Class 97
5.9 Graphical Representation of the HistoricalInfo Class 98
5.10 A Part of the Role Ontology for the Healthcare Domain 100
5.11 An Excerpt of the Resource Ontology for the Healthcare Domain 101
5.12 An Excerpt of the Relationship Ontology for the Healthcare Domain 104
5.13 An Excerpt of the Place Ontology for the Healthcare Domain 105
5.14 Graphical Representation of the Full StatusInfo Class 106
5.15 Graphical Representation of the Full ProfileInfo Class 107
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>The Core Situation Ontology</td>
<td>122</td>
</tr>
<tr>
<td>6.2</td>
<td>An Excerpt of Purpose Ontology</td>
<td>127</td>
</tr>
<tr>
<td>7.1</td>
<td>The Core Policy Ontology</td>
<td>136</td>
</tr>
<tr>
<td>7.2</td>
<td>The CAURA Policy Ontology</td>
<td>137</td>
</tr>
<tr>
<td>7.3</td>
<td>The CARPA Policy Ontology</td>
<td>140</td>
</tr>
<tr>
<td>8.1</td>
<td>The Overall Architecture of the CAAC Prototype Framework</td>
<td>153</td>
</tr>
<tr>
<td>8.2</td>
<td>Technology Used to Implement Prototype Components</td>
<td>156</td>
</tr>
<tr>
<td>8.3</td>
<td>A Snapshot of the Context Ontology</td>
<td>157</td>
</tr>
<tr>
<td>8.4</td>
<td>A Snapshot of the Domain-Specific Role Ontology</td>
<td>158</td>
</tr>
<tr>
<td>8.5</td>
<td>A Java Code Snapshot in Eclipse for the Specification of User-Defined Rule</td>
<td>160</td>
</tr>
<tr>
<td>8.6</td>
<td>A Part of the Implementation of Our Inference Engine: A Java Code Snapshot in Eclipse</td>
<td>161</td>
</tr>
<tr>
<td>8.7</td>
<td>A Java Code Snapshot in Eclipse for CAAC Decision Making</td>
<td>163</td>
</tr>
<tr>
<td>9.1</td>
<td>An Excerpt of Context Ontology (Context Entities in Scene #1)</td>
<td>171</td>
</tr>
<tr>
<td>9.2</td>
<td>A Screenshot of Our Prototype</td>
<td>173</td>
</tr>
<tr>
<td>9.3</td>
<td>An Excerpt of Context Ontology (Context Entities in Scene #2)</td>
<td>175</td>
</tr>
<tr>
<td>9.4</td>
<td>Average Response Time over Different Numbers of Context Types</td>
<td>181</td>
</tr>
<tr>
<td>9.5</td>
<td>Average Response Time over Different Numbers of Context-Aware Policies</td>
<td>182</td>
</tr>
<tr>
<td>9.6</td>
<td>Average Response Time over Various Size of the Ontology KB</td>
<td>183</td>
</tr>
<tr>
<td>9.7</td>
<td>Ontology Loading Time over Different Numbers of Context-Aware Policies</td>
<td>184</td>
</tr>
<tr>
<td>9.8</td>
<td>Average Response Time Over Different Numbers of Access Control Policies</td>
<td>186</td>
</tr>
<tr>
<td>9.9</td>
<td>Average Response Time Over Different Size of the Ontology KB</td>
<td>187</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Example Plain-Language Policy Rules ........................................ 20
2.2 Example Plain-Language Policy Rules ........................................ 21
3.1 Access Control Matrix ............................................................ 31
3.2 Access Control List ............................................................... 32
3.3 Capability-Based Access Control ................................................. 32
3.4 Comparative Analysis of the Existing Context-Aware RBAC Frameworks . 45
3.5 Comparative Analysis of the Existing Context-Aware ABAC Frameworks . 49
3.6 Comparative Analysis of the Existing Situation-Aware Models and Frameworks .......................................................... 52
4.1 Context-Aware User-Role Assignment .......................................... 77
4.2 Context-Aware User-Role Assignment Policy .................................. 78
4.3 Context-Aware Role-Permission Assignment Policy ............................ 79
5.1 Resource Identities of Classes in the Resource Ontology .................... 103
5.2 User-Defined Reasoning Rules ................................................. 110
5.3 User-Defined Reasoning Rules (Continued) .................................... 111
6.1 An Example Atomic Situation: RNFromGWForDO .......................... 125
6.2 An Example Atomic Situation: RNAtDTForDO .............................. 126
6.3 An Example SWRL Rules Set that Captures the Purpose is DO .......... 128
6.4 Definition of Different Situations .............................................. 128
6.5 An Example Composite Situation: RNFromGWAtDTForDO ............... 129
7.1 An Example Context-Aware User-Role Assignment Policy .................. 134
<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2</td>
<td>An Example Context-Aware Role-Permission Assignment Policy</td>
<td>135</td>
</tr>
<tr>
<td>7.3</td>
<td>An Example CAURA Policy for the Emergency Doctor Role</td>
<td>143</td>
</tr>
<tr>
<td>7.4</td>
<td>An Example CARPA Policy for the Emergency Doctor</td>
<td>144</td>
</tr>
<tr>
<td>7.5</td>
<td>An Example CAURA Policy for the Registered Nurse Role</td>
<td>146</td>
</tr>
<tr>
<td>7.6</td>
<td>An Example CARPA Policy for the Registered Nurse</td>
<td>147</td>
</tr>
<tr>
<td>8.1</td>
<td>An Example Reasoning Rule</td>
<td>159</td>
</tr>
<tr>
<td>9.1</td>
<td>Context Information Characterizing the Context Entities in Scene #1</td>
<td>172</td>
</tr>
<tr>
<td>9.2</td>
<td>An Example Access Query</td>
<td>172</td>
</tr>
<tr>
<td>9.3</td>
<td>Access Query Results</td>
<td>172</td>
</tr>
<tr>
<td>9.4</td>
<td>Context Information Characterizing the Context Entities in Scene #2</td>
<td>175</td>
</tr>
<tr>
<td>9.5</td>
<td>An Example Access Query</td>
<td>176</td>
</tr>
<tr>
<td>9.6</td>
<td>Access Query Results</td>
<td>176</td>
</tr>
<tr>
<td>9.7</td>
<td>An Example Access Query</td>
<td>176</td>
</tr>
<tr>
<td>9.8</td>
<td>Access Query Results</td>
<td>177</td>
</tr>
<tr>
<td>9.9</td>
<td>Definition of Situation</td>
<td>178</td>
</tr>
<tr>
<td>9.10</td>
<td>An Example Access Query</td>
<td>178</td>
</tr>
<tr>
<td>9.11</td>
<td>Access Query Result</td>
<td>178</td>
</tr>
<tr>
<td>9.12</td>
<td>Comparative Analysis between Our Context-Aware Access Control Framework and the</td>
<td>191</td>
</tr>
<tr>
<td></td>
<td>Related Access Control Frameworks</td>
<td></td>
</tr>
</tbody>
</table>
Part I
In recent years, the rapid advancement of computing technologies has led to the computing paradigm shift from fixed desktop to ubiquitous context-aware environments [95]. Such a shift brings with it opportunities and challenges. On the one hand, users demand access to services in an anytime, anywhere fashion. On the other hand, such access has to be carefully controlled due to the additional challenges coming with the dynamically changing environments, so as not to compromise the relevant privacy and security requirements of the stakeholders. Therefore, there is the increasing demand for new access control approaches to link their decision making abilities with context-awareness in recent years. As a consequence, the dynamically changing contextual information [34, 35] (e.g., the states of the relevant access control-specific entities), which is relevant to the user-application (e.g., access control) interaction, needs to be taken into account when making access control decisions.

This thesis introduces a context-aware access control framework called CAAC, to address the above-mentioned challenges. In this chapter, we discuss the necessity of a new CAAC framework in the context of today’s dynamic and context-aware environments. Section 1.1 introduces the context of this research, briefly discussing the strengths of the existing access control approaches, by pointing out the general issues to develop a new context-aware access control framework. In Section 1.1.1, we provide an overview of the basic access control approaches, and how these approaches are applied to provide appropriate software services to users. Section 3.2 discusses the extended role-based and attribute-based access control approaches by incorporating the different types of context and situation information into the access control policies. In Section 1.2, we present the research issues for the development of a new context-aware access control framework. An
overview of our proposed context-aware access control approach is presented in Section 1.3 while Section 1.4 lists the key contributions of this thesis. Finally, Section 1.5 provides an outline of the thesis structure.

1.1 Context-Aware Access Control

1.1.1 Access Control

Access control is one of the fundamental security mechanisms needed to protect information resources and software services against unauthorized access according to a security policy, verifying whether a user is allowed to carry out a specific action on a service.

In the literature, a significant number of access control solutions/techniques have been developed over the last decade ranging from general approaches to application-specific approaches. Two general and standard access control paradigms for granting accesses to services are Role-based Access Control (RBAC) [40, 84] and Attribute-based Access Control (ABAC) [72, 92].

RBAC typically evaluates access permission through roles assigned to users and each role assigns a collection of permissions to users who are requesting the access to the resources [40, 84]. It simplifies the management of access control policies by creating user-role and role-permission mappings. A recent study shows that the RBAC has become the most widely used access control paradigm [75]. RBAC has received broad support as a generalized approach to access control, and is well recognized for its many advantages in performing large-scale authorization management [41]. In small-scale domains, the existence of an administrator to manage all the user-role and role-permission assignments in the system is possible. However, large-scale domains may have hundreds of roles and hundreds of thousands of users. In such cases, it is impractical for an administrator to manage all of these associations, which make it a less effective solution for modern dynamic environments. Moreover, the basic RBAC approach does not provide adequate functionality to incorporate and adapt to dynamically changing contexts and situations. Therefore, in order to provide secure access to software services in a context-aware
manner, there is a need for an access control framework with dynamic associations of both user-role and role-permission capabilities.

On the other hand, ABAC grants accesses to resources and services based on attributes of entities (e.g., the attributes possessed by the requester) [92]. The ABAC approach has similar drawbacks in considering the context-awareness capabilities. Besides, the attribute-based approaches are not applicable in large-scale domains because the identity-based approaches do not scale well in large open systems [71].

In today’s dynamic and context-aware environments, the above-mentioned general and traditional access control paradigms/approaches are inadequate to manage access to appropriate resources and services by the right users in a context-aware manner. This gap in the literature suggests that there is a need for a new approach to context-aware access control for software services.

1.1.2 Context-Aware Access Control

Context-aware access control is one of the security mechanisms needed to provide a flexible control of users’ access to resources according to the currently available context information [27].

Sandhu and Samarati have stated that access control is not only a solution for controlling resource access, it should be coupled with auditing also; where auditing requires the available attributes of the users, and their activities [85]. The second part of this statement has become a major concern over the years. Towards this end, a number of context-aware access control approaches have been developed over the last decade ranging from role-based to the attribute-based access control approaches.

In this research, our context-aware access control is an improved access control mechanism, to control resource and service access on the basis of dynamic contexts and situations and thereafter reauthorization of access are taken if context/situation changes.
Chapter 1. Introduction

Context Model for Access Control

Many researchers have attempted to define the concept of context. The generalized definition of context given by Dey, who defines context as any information that can be used to characterize the situation of an entity (an entity is a person, a place or an object) [34, 35]. This definition does not specify the different types of entities specific to access control or the wide range of context information characterizing these entities (e.g., the relationships between relevant entities). In this thesis, we specialize Dey’s definition of context to specifically cover context-aware access control applications. We define the context to mean any information that can be used to characterize the state of the relevant access control-specific entities and the state of the relevant relationships between different persons (as entities).

During the past decades, several research works have extended the basic RBAC approach [40, 84] by incorporating some specific types of context information: temporal information (e.g., [18, 55]), spatial information (e.g., [19]), and both the time and location (e.g., [25]). Recently, Kulkarni et al [68], He et al [48], Huang et al [53], and Schefer-Wenzl et al [87] have adopted and extended the basic RBAC approach, but they have considered some further context information other than the temporal and spatial dimensions, such as the resource and environment dimensions as well as the user dimension. These approaches and their associated context models are still limited in capturing a wide range of important context information relevant to access control.

Several further research works (e.g., [28, 54, 90, 29]) have adopted and extended the basic ABAC approach [92] with context-awareness capabilities by modelling context aspects of the user, resource and environment dimensions as attributes. Other recent attribute-based access control approaches consider the relationship dimension: ReIBAC [102] considers the permissions of a subject on an object and ReBAC [42] considers the relationships between individual users (e.g., friend and friend-of-friend) as context information. These approaches and their associated context models have similar drawbacks in capturing the specific types of context information and in inferring the richer context information in a systematic manner.
1.1. Context-Aware Access Control

In the field of specific context modelling and management approaches, there have been a significant amount of research works (e.g., [26, 93, 50, 39, 52, 96, 79]) for modelling context information representing the context entities such as person, place and object, without considering the access control-specific context entities and information.

In general, the existing access control approaches and their associated context models are not adequate to capture and reason about all the different types of contextual information such as the information characterizing the relationships between entities. On the other hand, the specific context modelling and management approaches do not provide direct context modelling support for concepts related to access control. Therefore, there is a need for a comprehensive and extensible access control-specific context model for modelling a wide range of dynamic context information. In addition, there is a need for inferring high-level (implicit) context information from the other available context information.

Situation Model for Access control

In the literature, many researchers have attempted to define the concept of situation. As defined by Endsley [38, 37], the situation comprises the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future. This definition has been widely accepted by the information technology community. In the context-awareness literature (e.g., [16], [20], [93]), Wang et al [93] have described user-specific situation-awareness only concentrating on the state of the user.

Yau and others have defined the situation as a set of context attributes of users, systems and environments over a period of time affecting future system behavior [98]. Later, Yau and Liu have presented a Situation-Aware Access Control (SA-AC) based privacy-preserving service matchmaking approach [99]. The SA-AC approach incorporates situation-aware constraints into basic RBAC approach, such that the dynamic states of service providers, requesters and environments are considered, in order to make access
control decisions. Kim and Lim have proposed the Situation-Aware RBAC (SA-RBAC) approach [65], which has extended the basic RBAC approach and grants permissions to users based on the situation information. The SA-RBAC approach is used to deal with the situation information by considering the combination of the required credentials of context entities. Recently, Yau and Huang [97] have presented a situation-aware ontology in service-based system to ensure the satisfaction of user’s quality-of-service (QoS) requirements in dynamically changing environments. They consider situation as a set of context attributes of users and other objects (entities) involved in computing.

The above mentioned situation-aware approaches consider situation as the specific combinations of context information (e.g., the states of the relevant entities). In dynamic and context-aware environments, however, the states of the relevant relationships between different context entities is also an important consideration in access control decision making. Furthermore, the purpose or user’s intention in accessing resources and services is another important aspect that need to consider. Therefore, a purpose-oriented situation model to identify relevant situation information is needed.

Policy Model for Access Control

In access control approaches, the policies are the rules that determine how accesses to resources and services are controlled and according to which access control decisions are consequently made [85].

Research continues to the present age to extend the basic RBAC [40, 84] and ABAC approaches [92] in support of new policies to integrate dynamically changing (context and situation) information into their policy models.

The context-aware role-based approaches incorporate the spatial and temporal information to specify the user-role and role-permission assignment policies (e.g., [18], [55], [19], [25], [68], [87]). These approaches provide useful insight into considering user-role and role-permission assignment concepts in our research. While presenting context and policy models for access control to specific services, these approaches are still limited in
1.1. Context-Aware Access Control

integrating diverse context information into these assignments.

On the other hand, the context-aware attribute-based approaches integrate the user, resource, relationship, and environment dimensions into their access control policies (e.g., [28, 54, 90, 102, 42, 29]). The main building block of these attribute-based approaches is the attributes. They grant accesses to services based on attributes of entities (e.g., the attributes possessed by the users) rather than direct role-based support. However, their policy models do not have the functionalities to provide user-role and role-permission assignments.

Other than the role-based and attribute-based approaches, the situation-aware access control approaches integrate the specific combinations of certain contextual information in the access control policies. The SA-RBAC [65] and SA-AC [99] approaches extend the basic RBAC model and incorporate the situation-aware constraints into user-role and role-permission assignments, such that the states of service providers, requesters and environments. However, they do not consider some other situation-specific constraints such as the states of the relationships between entities and the user’s intention or purpose into these assignments.

Therefore, there is still need for a new context-aware policy model to incorporate diverse context and situation information into both user-role and role-permission assignments for dynamic access control decision making.

1.1.3 Discussion

Motivated by the above-mentioned observations and the gaps in the literature, we feel that there is a need for a new context-aware access control (CAAC) framework for information resources and software services. For the CAAC framework, there is a need for a context model to capture different types of dynamic context information and to reason about richer context information from the available information in a systematic manner. Furthermore, a situation model to identify and infer relevant situation information based on the captured context information is needed. Finally, a policy model is also needed for the CAAC
framework in order to incorporate this context and situation information into providing context-aware access control to resources and services.

1.2 Research Issues

Even though some context-/situation-specific access control approaches have already been proposed, a comprehensive context-aware access control policy framework, flexible and dynamic enough to deal with context-aware user-role and role-permission assignments, is still missing. For example, consider the roles of patient and nurse in a hospital context. A nurse Mary may be allowed to access the medical health records of a patient Bob for daily operational purposes, if she has been assigned to look after Bob but only when she is located in the general ward. That is, Mary can play the nurse role if she fulfills these dynamic conditions and consequently can access Bob’s medical health records. When the context/situation changes (e.g., Bob’s health condition becomes critical or Mary leaves the ward), decisions on further access requests by Mary to Bob’s medical records by playing the nurse role, may change accordingly (e.g., denied). Therefore, an access controller needs to evaluate such dynamic attributes when enabling user-role and role-permission assignments. However, existing access control approaches do not have adequate functionalities to incorporate a wide range of important contextual information into user-role and role-permission assignments. As such kinds of dynamic attributes need to be integrated into the basic role-based access control approach, some important issues arise to realize flexible and dynamic access control. The technical issues and their related requirements are as follows:

(1) Understanding and identifying the context information and associated context entities relevant to access control, thereby formulating an appropriate context model for capturing the dynamic context information.

(2) Developing a suitable reasoning technique to infer richer and more complex context information according to user-defined rules.

(3) Identifying the user's intention in accessing the services and correlating with specific combinations of selective context information concerning that intention, thereby for-
1.3. An Approach to Context-Aware Access Control for Software Services

To address and tackle the identified research issues and challenges, the overall goal of this research is to develop a new access control framework, named Context-Aware Access Control (CAAC), that is capable of providing secure access to information resources and software services according to the dynamic context information; and supports software engineers to improve privacy and security when building CAAC applications.

1.3 An Approach to Context-Aware Access Control for Software Services

This research is concerned with how to support and manage a fine-grained access control to software services in a context-aware manner. We show how the context-/situation-awareness capabilities and role-based access control (RBAC) decision making abilities can be integrated, allowing us to control access to resources and services, while retaining the benefits of RBAC, such as role hierarchy, role inheritance, etc. Towards this goal, we present an approach to context-aware access control (CAAC) for software services.

Our CAAC approach extends the basic RBAC approach [40, 84]. It enables dynamic privileges assignment at two levels: firstly, the roles are assigned to the users based on the relevant dynamic information, then, the permissions are assigned to that roles based on the relevant dynamic information. Towards this end, we introduce several new concepts: the identification of access control-specific context information and purpose-oriented situation information, and the specification of context-aware user-role
assignment and role-permission assignment policies.

Figure 1.1 illustrates a 5-phase Context-Aware Access control (CAAC) approach to provide context-specific access to software services.

- **Capture context information.** It is the process of capturing the basic context information (low-level information) from the interaction environment, and consequently inferring high-level context information that are not directly available, but can be derived from the available context information using user-specified rules.

- **Identify situation information.** It is the process of identifying the purpose or user’s intention in accessing services and consequently identifying relevant atomic situations using the elementary information from phase 1, and inferring new complex (or composite) situations from the basic atomic situations.

- **Specify user-role assignment.** It is the process of specifying context-aware user-role assignment policies in order to dynamically activate and revoke role according to the relevant information captured/identified in phases 1 and 2.

- **Specify role-permission assignment.** It is the process of specifying context-aware role-permission assignment policies in order to dynamically manage user’s access to
1.4. Research Contributions

services (by playing appropriate role) according to the relevant information captured/identified in phases 1 and 2.

• **Make access decision.** It is the process of making access control decisions by enforcing the access control policies specified in the previous phases, and consequently taking necessary action in a timely and effective manner.

1.4 Research Contributions

Motivated by the above-identified research issues and challenges, in this research, we have developed a **Context-Aware Access Control (CAAC) Framework** to provide context-aware access control support for software services. It makes the following contributions.

(C1) **A Theoretical Context-Aware Access Control Approach.** For our CAAC framework, we introduce a conceptual approach to context-aware access control. Our CAAC Approach adopts and extends the basic RBAC approach. The novel feature of this approach is a formal language for specifying the elements of our CAAC framework. The approach has introduced the dynamic context-awareness capabilities into both user-role and role-permission assignments. Our approach provides the notion of dynamic context information and purpose-oriented situation information relevant to access control. We take context to mean any information that can be used to characterize the state of a relevant entity and/or the state of a relevant relationship between different entities. A **purpose-oriented situation** is defined as a specific combination of certain context information that is relevant to the purpose or user’s intention in accessing service.

(C2) **A Context Model for Access Control.** We propose an ontology-based context model that is specific to access control and is capable to represent different types of context entities and context information and infer relevant high-level context information relevant to access control in today’s open and dynamic environments.

• **Representation of Basic Contexts:** The context model uses the ontology language OWL to represent and capture different types of context information in a systematic way. In particular, our context model can capture a wide
range of context entities, and the relevant information about them and their relationships. For example, in a scenario where a registered nurse Mary can access a patient Bob’s past medical history if Jane who is a general practitioner is present in the same location with Mary, the relevant context entities include Mary (the user), Bob (the resource owner), Jane (an environmental person), the interpersonal (“care”) relationship between Mary and Bob, and the co-location relationship between Mary and Jane. The relevant context information about these entities is required in making such an access control decision.

- **Reasoning about High-Level Contexts:** The context model is extended with user-defined SWRL rules to reason about high-level implicit context information that is not directly available but can be derived from other information. In particular, our reasoning model uses the context representation model to infer richer context information at different abstraction levels based on the basic context information. For example, our reasoning model can capture the derived relationship (e.g., interpersonal relationship between the registered nurse and patient is “assigned nurse”) from the available (basic) context information (patient’s profile, nurse’s profile, etc.).

(C3) **A Situation Model for Access Control.** Our model uses the purpose-oriented situation information in order to provide situation-specific access to software services (authorization). The situation model uses the same ontology languages OWL and SWRL to model and reason about situations. The novel features of this approach are as follows:

- **Atomic Situations:** Our situation model uses the relevant elementary information from the context model (in order to identify the relevant states and the user’s intention in accessing the services) and defines the atomic situations. The atomic situation can be composed of the relevant states of the entity, states of the relationships between entities and the user’s intention or purpose.

- **Composite Situations:** The situation model is extended with user-defined reasoning rules to infer composite situations that can be derived from one or more atomic situations.
1.5. Thesis Outline

(C4) **A Policy Model for Access Control.** We introduce a policy model to define and enforce context-aware access control policies that take into account relevant dynamic context and situation information as captured/identified in the context and situation model. The policy model also uses OWL and SWRL. The main novel point in our policy model is that it provides an ontology-based model for specifying context-aware access control policies in terms of dynamic context-aware user-role assignment and role-permission assignment. Our policy model uses the dynamic information from the context and situation models, and allows the users to access information resources of different granularities depending upon the dynamically changing context and situation information.

(C5) **Implementation of the CAAC Framework.** Other than the above four major contributions, we provide a software prototype of our CAAC framework for the development of the context-aware access control applications, in order to demonstrate the practical applicability of our approach.

(C6) **Evaluation of the CAAC Framework.** We present a case study in the healthcare domain using our prototype framework. The test cases in our case study have shown that our framework is effective. We have also conducted a range of experiments on a simulated environment, *patient medical records management (PMRM)*, to retrieve and modify different medical records of patients based on dynamic information. Experimental results quantify the performance of our CAAC framework. In addition, we analyse and compare our framework with several related frameworks. The comparative assessment highlights the benefits and strengths of our CAAC framework. The comparative assessment has shown that our framework offers a range of new capabilities such as purpose-oriented situation and context-aware user-role and role-permission assignments.

### 1.5 Thesis Outline

This thesis is structured into three main parts consisting of ten chapters. Part I consists of the introduction (Chapter 1), an application scenario (Chapter 2) and a literature review (Chapter 3) to provide the required background knowledge for this thesis. Part II presents
Chapter 1. Introduction

the key research contributions of this thesis: the formal specifications of our context-aware access control (CAAC) approach (Chapter 4), the design of the context model (Chapter 5), the design of the purpose-oriented situation model (Chapter 6) using the contextual information from the context model, and the design of the context-aware policy model (Chapter 7) by integrating the dynamic context and situation information from the context and situation model. In addition to these four major contributions, Part III provides a prototype implementation for our context-aware access control framework (Chapter 8), the evaluation of our CAAC framework (Chapter 9) and finally a conclusion recapping the contributions of this thesis and related topics for future investigation (Chapter 10).

- Chapter 2 provides an application scenario in the domain of patient medical records management, concerning support for health professionals (e.g., doctor, nurse) getting access to the patients’ electronic health records in a context-aware manner. This scenario is used as a running example in the remaining chapters of this thesis. The scenario captures the technical challenges that a context-aware access control framework has to meet in order to access appropriate resources and services. Based on these challenges, we discuss the general requirements to control context-specific access to software services.

- Chapter 3 reviews the existing approaches to access control for information resources and software services ranging from the traditional forms of access control to context-specific access control technologies. Some specific context modelling and management approaches are also discussed in Chapter 3 as they are relevant to our context-aware access control research.

- In Chapter 4, we propose a theoretical context-aware access control framework, specifying the elements of the framework. We define the context information by identifying the context entities and context information relevant to access control, define the purpose-oriented situation information specific to access control, and describe the use of this context and situation information for dynamic association of user-role and role-permission assignments.

- Chapter 5 presents an access control-specific context model that is capable of representing different types of context entities and context information representing these
entities. The model adopts the OWL language to represent a wide range of dynamic context information. A reasoning approach is also proposed to derive high-level context information from the available context information where the SWRL language is adopted to specify user-defined reasoning rules.

- Chapter 6 presents a purpose-oriented situation model that adopts semantic technologies (OWL and SWRL) in modelling the relevant situations. The model uses the relevant context information from the context model, in order to identify the relevant states (of the entities and the relationships between entities), the user’s intention or purpose in accessing the services, and consequently model and represent the atomic situations. It also has the reasoning capability, to infer the composite situations from the relevant atomic situations through user-defined reasoning rules.

- Chapter 7 presents a policy model for specifying context-aware access control policies in terms of dynamic context-aware user-role assignment and role-permission assignment. The model uses the relevant context and situation information that reflects the dynamically changing conditions of the environments. The policy model adopts the same semantic languages OWL and SWRL to specify and evaluate access control policies.

- In Chapter 8, we present a prototype implementation of our CAAC framework for developing context-aware access control applications. We also present a context-aware access control application developed in the area of patient medical records management (PMRM).

- Chapter 9 presents a case study in the healthcare domain using our prototype framework and PMRM application, to demonstrate the effectiveness of our CAAC framework. We provide two test cases that highlight context-aware access control requirements of the PMRM application. The case study shows the use and applicability of our framework in developing context-aware access control applications. We also evaluate the runtime performance overhead of the CAAC framework by conducting a range of experiments. The results of these experiments demonstrate the feasibility of developing context-aware access control applications using our CAAC framework. In addition to the case study and performance evaluation, we also evaluate the ef-
effectiveness of our framework by presenting a comparative assessment of our CAAC framework and the related frameworks. It highlights the strengths of our framework in the context of the existing access control frameworks.

- Finally, Chapter 10 concludes the thesis by summarizing the key contributions of this thesis and discussing some interesting research directions for future investigation.
In this chapter, we present a motivating scenario in the domain of patient medical records management and its associated requirements. The objective of this chapter is twofold. The first objective is to present an application scenario which illustrates context-aware access control. We analyze the scenario and illustrate the need for the incorporation of the dynamic contextual information in the access control process. We have used the scenario throughout the next chapters to explain our CAAC framework concepts by providing suitable examples. The second objective is to identify the general requirements of developing context-aware access control applications by analyzing the application scenario.

The chapter is organized as follows. In Section 2.1, we present an application scenario of a healthcare domain to motivate our work. In Section 2.2, we analyze the scenario to capture the technical challenges that a context-aware access control framework has to meet. Based on these challenges, in Section 2.3, we identify a number of general requirements for controlling the access to information resources or services in a context-aware manner. Finally, Section 2.4 summarizes the chapter.

2.1 Application Scenario

As an example of the type of situations that a context-aware access control framework has to consider, in this section we present a motivating scenario. We consider the area of patient medical records management (PMRM) in the healthcare domain as a motivating scenario. The scenario illustrates the need for the incorporation of dynamic contextual information in the access control process.
Chapter 2. Research Motivation

Let us consider the following business scenario (Scene #1 and Scene #2).

- **Scene #1**: The scenario begins with patient Bob who is in the emergency room due to a heart attack. While not being Bob’s usual treating physician, Jane, a general practitioner at the hospital, is required to treat Bob and needs to access Bob’s emergency medical records from the emergency room.

RBAC [84] is the widely accepted traditional access control technique, that evaluates access permission through roles assigned to users by taking into account the static user-role and role-permission assignments. It does not have functionality to integrate dynamically changing contextual information in the access control process. The different types of information involved in this scenario are highly dynamic, and granting a user access without taking the dynamic information into account can compromise security. Therefore, RBAC will not grant Jane the access to the requested services (access to the Bob’s emergency medical records) under these dynamic conditions (e.g., the emergency room, the treating physician relationship). Concerning this scenario, we need the context-aware access control policies. One of the relevant policies (plain-language policy rules) is shown in Table 2.1 (see Policy #1).

Table 2.1: Example Plain-Language Policy Rules

<table>
<thead>
<tr>
<th>No</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A general practitioner who is a treating doctor of a patient, is allowed to read/write the patient’s emergency medical records in the hospital for emergency treatment purpose. However, all general practitioners should be able to access the patient’s emergency medical records in the hospital (by playing the emergency doctor role).</td>
</tr>
</tbody>
</table>

- **Scene #2**: After getting emergency treatment, Bob is shifted from the emergency department to the general ward of the hospital and has been assigned a registered nurse, Mary, who has regular follow-up visits to monitor his health condition. Mary
2.1. Application Scenario

needs to access several types of Bob’s records (daily medical records, past medical history and private medical records) from the general ward with certain conditions.

The different types of dynamically changing information are also involved in this scenario (e.g., registered nurse relationship). The corresponding context-aware access control policies (plain-language policy rules) are shown in Table 2.2.

Table 2.2: Example Plain-Language Policy Rules

<table>
<thead>
<tr>
<th>No</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy #1</td>
<td>A registered nurse within a hospital is granted the right to read/write a patient’s daily medical records during her ward duty time and from the location where the patient is located for daily operational purpose.</td>
</tr>
<tr>
<td>Policy #2</td>
<td>In addition, the nurse is allowed to read the patient’s past medical history, if a general practitioner is present in the same location.</td>
</tr>
<tr>
<td>Policy #3</td>
<td>Moreover, the nurse can access the patient’s private medical records, if she is an assigned nurse of that patient.</td>
</tr>
</tbody>
</table>

These context-aware access control policies (shown in Tables 2.1 and 2.2) are based on a set of constraints on the user roles (e.g., general practitioner, emergency doctor, registered nurse) and services/resources (e.g., daily medical records, past medical history, private medical records). These policies need to be evaluated in conjunction with some further dynamic information such as the contextual information (e.g., location, ward duty time, interpersonal relationship) and the purpose-oriented information (e.g., emergency treatment purpose, daily operational purpose).

The above mentioned PMRM application is an example of how to realize context-aware access control decisions. In our application scenario, we only consider the context-aware access control policies for the general practitioners and registered nurses (2 roles). For our PMRM application, we consider the number of policies up to 500 with respect to 138 different health professionals [1] (i.e., 138 roles). The detail analysis of our PMRM application and case study are presented in Chapters 8 and 9.
2.2 Scenario Analysis

In this section, we analyze the above-mentioned application scenario, in order to capture the technical challenges to manage the access control to resources or services.

The following context entities are involved in the scenario: (i) user, (ii) owner, (iii) other person, (iv) role, (v) relationship, (vi) resource and (vii) place.

- A resource requester or *User* is an entity, usually a human-being, whose access request is being controlled (e.g., in Scene #1, Jane is a resource requester or user).

- A resource *Owner* is an entity that owns the different component parts of a resource (e.g., in Scene #2, Bob owns the private medical records).

- An environmental person (*Other Person*) is an entity, who is neither a resource requester nor a resource owner but relevant in a certain situation. In the example scenario (Scene #2) where a registered nurse Mary can access patient Bob’s past medical history if Jane (by playing a general practitioner role) is present in the same location as Mary, for instance, Mary is a user, Bob is an owner, and Jane is another relevant person.

- A *Role* is an entity that reflects user’s job function within an organization. In Scene #1, Jane can play the emergency doctor or general practitioner role.

- A *Relationship* is an entity that contains the different types of relationship information. For example, in Scene #2, the interpersonal relationship between Mary (resource owner) and Bob (resource requester) is “assigned nurse”.

- A *Resource* is an entity that contains the different component parts of a resource (e.g., in Scene #2, the daily medical records, past medical history and private medical records are seen as the different component parts of Bob’s medical health records).

- A *Place* is an entity that has the location-specific information (e.g., location address, temperature) from which the access request is initiated. In Scene #1, Jane’s access request comes from the “emergency room”.

22
2.2. Scenario Analysis

The contextual information describing the above-identified context entities relevant to
access control in the scenario (and its relevant policies are shown in Tables 2.1 and 2.1) includes:

- The identity of the User (e.g., the identity of Jane in Scene #1).
- The identity of the Role.
- The identity and granularity level (different component parts) of a Resource.
- The location of the Users and Patient Bob.
- The relationship type, e.g., interpersonal relationship between User and Patient (e.g.,
  non-treating physician, in Scene #1 and assigned nurse, in Scene #2), location-centric
  or co-location relationship (e.g., Mary and Bob are co-located, in Scene #2).
- The specified time of the User - ward duty time (Scene #2).
- The health status of the patient Bob - critical in Scene #1 and normal in Scene #2.

Other than the above-identified contextual information, the following purpose-oriented
information is also involved in the scenario:

- The purpose in accessing the resources (emergency treatment purpose in Scene #1
  and daily operational purpose in Scene #2).

The application scenario illustrates many of the key requirements of the research presented
in this thesis. As different types of context entities and dynamically changing information
are involved in the access control process, some important technical issues arise.

Two different types of contextual information are involved in the application scenario: the basic low-level contextual information and the derived high-level contextual information. For example, in Scene #2, the derived relationship or interpersonal relationship between the registered nurse and patient is “assigned nurse”. It is not obtainable directly but can be derived from the available information (e.g., the nurse’s profile, patient’s profile, etc). In general, how to identify the different types of dynamic (basic and derived) contextual information which are integrated into the access
control process is the first important issue.

Other than the low-level and high-level contextual information, the user's intentions in accessing the resources and services are also involved in the application scenario. For example, in Scene #1, the user's intention is emergency treatment purpose. It can be identified from the available contextual information (who wants to access what and when) at runtime. Therefore, how to identify the different types of purpose-oriented situation information which are integrated into the access control process is the second important issue.

Normally, only a patient's treating physician is able to access the patient's all electronic health records. In the mentioned emergency scenario (Scene #1), Jane, while not being the treating doctor, can access Bob's emergency medical records from the emergency room of the hospital by playing emergency doctor role. That is, Jane can play the emergency doctor role when he is present in the emergency room and Bob's health condition is critical. He also can play general practitioner role. Thus, rather than having static user-role assignment, how to dynamically assign the roles to users according to contextual information is the third technical issue.

In general, a registered nurse is granted the right to read a patient's daily medical records within a hospital. But, she should only be able to read the medical records from the location where the patient is located, during her ward duty time. In the application scenario (Scene #2), Mary can access Bob's daily medical records during her ward duty time because she is present with Bob in the general ward. Furthermore, when the situation changes (e.g., Bob's health condition becomes critical again, like the Scene #1), decisions on further access requests by Mary to Bob's daily medical records, may change accordingly (e.g., denied). Therefore, how to assign the permissions to roles based on the relevant contextual information is also an important issue.
2.3 Requirements for Developing CAAC Applications

In this section we identify the general requirements of developing context-aware access control applications, based on the above motivating scenario.

Looking at the application scenario and technical issues identified in the previous sections, we can make some important observations concerning context-aware access control. Figure 2.1 illustrates the relationship chain among user, role and resource.

The relationship chain contains mainly two parts: the user-role mappings based on the relevant contextual information, and the role-permission (resource access permission) mappings based on the relevant contextual information. To support such context-aware access control in a computer application like the medical record management system, we need to consider the 3Ws: who (the appropriate users by playing appropriate roles) wants to access what (the appropriate parts of the resources), and when (the dynamic contextual information). In particular, a general *context-aware access control (CAAC) framework* is required to control the access to resources in such applications by taking into account the different types of contextual information that impact on the access control decisions. The
general requirements of developing context-aware access control applications are as follows:

(Req.1) **Representation of basic contexts**: What access control-specific basic contextual information should be considered as part of building context-aware access control? In general, there is a need to identify the different types of context entities and the basic contextual information that characterizes these entities. *How to model and capture the context entities and basic contextual information in an effective way?* There is a need to formulate a dynamic context model to represent the basic contextual information, in order to facilitate CAAC use.

(Req.2) **Inferring high-level contexts**: How to express user-defined reasoning rules to capture knowledge which is required but only implicitly present, thereby extending the dynamic context model? There are several types of contextual information that are not directly obtainable from the context entities but can be derived from their basic information. For example, in our application scenario, an *interpersonal relationship* between the doctor and patient that is not obtainable directly but can be inferred from other available information (e.g., the *doctor’s profile*, *patient’s profile*, etc). Thus, there is the need to express user-defined rules to represent and capture such complex and high-level implicit contextual information, and extend the dynamic context model with these newly created rules.

(Req.3) **Representation of basic purpose-oriented situations**: What access control-specific basic situation information should be considered as part of building context-aware access control? In general, there is a need to identify the user’s intention or purpose information using the available context information from the dynamic context model. *How to model and identify the basic purpose-oriented situations in an effective way?* There is a need to formulate a dynamic situation model to represent the atomic situations (basic purpose-oriented situations).

(Req.4) **Inferring composite situations**: How to express user-defined reasoning rules to infer a new composite situation from the atomic situations, thereby extending the dynamic situation model? There are some complex situations also involved in the access control process that are not directly obtainable but can be inferred from the one or more already identified situations. Thus, there is the need to express
user-defined reasoning rules to infer the complex and composite situations, and extend the dynamic situation model with these newly created rules.

(Req. 5) **Specification of user-role assignment policies:** *How to specify the user-role assignment policies based on the relevant context and situation information?* In general, there are different types of context and situation information involved in the user-role mappings. For example, in the application scenario, Mary can play the *nurse* role under certain contextual conditions (if she is located in the general ward, during her ward shift time and Bob’s health condition is normal). If the context/situation changes (e.g., Mary leaves the general ward or Bob’s health condition changes from normal to critical), the system will not allow Mary to play the *nurse* role. Thus, there is a need to specify dynamic user-role assignment policies based on the relevant context and situation information present.

(Req. 6) **Specification of role-permission assignment policies:** *How to specify the role-permission assignment policies based on the relevant context and situation information?* For example, in the application scenario, when the context/situation changes (e.g., Bob has come out of emergency and moved to a general ward), decisions on further access control to resources (e.g., Jane access to Bob’s emergency medical records by playing the general practitioner role) may change accordingly (e.g., denied). Thus, there is a need to specify context-aware role-permission assignment policies based on the relevant context and situation information.

(Req. 7) **Enforcement of access control policies:** *How to evaluate the access control policies based on the relevant contextual information present to realize a flexible and dynamic access control scheme?* There is a need for an access controller to evaluate the context-aware user-role and role-permission assignment policies based on the relevant contextual information, and manage re-authorization of access as situation changes.
2.4 Summary

In this chapter, we have presented a motivating scenario that requires a dynamic access control support and its associated requirements.

Firstly, we have outlined an application scenario in the area of patient medical records management in the healthcare domain. We have also presented some relevant access control policies related to the scenario. Secondly, we have analyzed the application scenario and identified the technical challenges to control access to resources/services in a context-aware manner. Finally, we have identified the general requirements for developing context-aware access control applications. One of the key requirements that we have identified is the incorporation of context-awareness capability in the access control process, to specify and evaluate the dynamic user-role and role-permission assignment policies.

In order to address the key requirements identified in this chapter, we introduce in the following chapters a theoretical CAAC framework (see Chapter 4), including the context model (Chapter 5), situation model (Chapter 6), and policy model (Chapter 7) for our CAAC framework.
Background and Literature Review

In this chapter, we review the existing access control approaches and frameworks that are related to our research. The aim of this chapter is to highlight the strengths and limitations of these access control approaches ranging from the traditional forms of access control approaches to application-specific approaches. How these approaches are applied to enable appropriate access to resources and services by right users in a context-aware manner are also discussed.

We distinguish four different categories of access control approaches and frameworks for the analysis of this chapter. Firstly, we introduce the traditional access control approaches and frameworks and highlight their benefits and shortcomings. These include traditional role-based access control framework which has become the most widely used access control model [75] and the attribute-based access control model (Section 3.1). Secondly, we describe the access control frameworks that incorporate the dynamically changing contextual information into role-based access control (Section 3.2). Thirdly, we describe the context-aware attribute-based access control frameworks that incorporate different types of context information as attributes (Section 3.3). Fourthly, we describe the situation-aware access control frameworks (Section 3.4). We highlight the limitations and shortcomings of these context and situation-specific access control frameworks that motivate us to develop a new context-aware access control framework for software services. In addition to these four categories of access control frameworks, we also describe the specific context modelling and management approaches that model and reason about the contextual information in pervasive and ubiquitous computing environments (Section 3.5). In Section 3.6, we discuss the literature in terms of context, situation and policy modelling issues and highlight the need for a new context-aware access control framework.
Chapter 3. Background and Literature Review

and its associated context, situation and policy concepts. Finally, section 3.7 summarizes
the chapter.

3.1 Access Control

Access control is one of the fundamental security mechanisms needed to protect information
resources and software services. It determines whether a request to access the resources
and services provided by a system should be permitted or denied. Access control has three
major components: subjects, objects and operations. In formal terms, subjects represent
the users or requesters in a system performing actions on the objects, objects represent the
requested resources, and operations represent the actions that the subjects can perform on
the objects.

3.1.1 Access Control History

In this section, we analyse the history of traditional access control models and frameworks.

Access control technology has a long history that started in the late 60s. It was
first introduced by Lampson in 1971 [69], who proposed a formal and mathematical
description of a basic access control model named access control matrix. In 1983, access
control technology took a significant step forward when the U.S. Department of Defence
(DoD) defined discretionary access control (DAC) and mandatory access control (MAC)
models [10]. In general, access control models can be classified as discretionary access
control and non-discretionary access control (NDAC) or mandatory access control.

DAC is defined by the trusted computer system evaluation criteria (TCSEC) [10]
as a means of restricting access to objects based on the identity of subjects and/or groups
to which they belong. In the DAC-based access control models, the object owners specify
who can access what objects through access control policies. Some of the traditional
access control models such as access control matrix (AM) [69, 70, 47, 86], access control
list (ACL) [45, 82, 86] and capability-based access control (CBAC) [85, 86] are well known
DAC-based solutions.
3.1. Access Control

MAC is defined by the trusted computer system evaluation criteria (TCSEC) [10] as a means of restricting access to objects based on the sensitivity of the information contained in the objects and the formal authorization of subjects to access information of such sensitivity. In the MAC-based solutions, the individual owners do not have any choice to specify who can access what objects through access control policies. The access control mechanisms usually control accesses to the objects and the users cannot alter the access permissions. Role-based access control (RBAC) [40, 84] and attribute-based access control (ABAC) are two widely-accepted MAC-based solutions.

In the following sections, we discuss the traditional forms of access control models and frameworks: access control matrix, access control list, capability-based access control, role-based access control and attribute-based access control.

<table>
<thead>
<tr>
<th>Subject</th>
<th>file1</th>
<th>file2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>read</td>
<td>read, write</td>
</tr>
<tr>
<td>Jane</td>
<td>read, write</td>
<td>read</td>
</tr>
</tbody>
</table>

3.1.2 Access Control Matrix

Lampson [69] has proposed the first basic model of access control for controlling access rights in the DAC-based systems, which is called as access control matrix or access matrix (AM). It is an abstract and formal security model of protection state in computer systems. It describes the relationships between the subjects (e.g., users) and objects (e.g., files, processes, resources, services) in a system. AM has three major components: a set of objects \(O\), a set of subjects \(S\) and an access function \(A\). An object is an entity in the system which needs to be protected and a subject is an entity in the system who wants to access one or more objects. Which subjects can access which objects is determined by the access function. The rows of the AM are labeled with the subject names and its column by the object names. Each element or cell of the AM consists of a set of strings called
access attributes, such as read and write. For example, Mary has ‘read’ access on file1, and read and write access on file2 (see Table 3.1).

Though access control matrix has been used in both operating and database systems, in the large-scale systems in which more than one user need same objects, it is very hard to specify the relationships between the users and objects using access control matrix [86].

| Table 3.2: Access Control List |
|-------------------|-------------------|
| file1             | Mary, read        |
|                   | Jane, write       |

3.1.3 Access Control List and Capability-Based Access Control

Access control list [45, 82, 86] is another DAC-based access control solution in which each column in an access control matrix is translated to an access control list (ACL). ACL contains entries (called access control entries, ACEs) for the subjects, which describe the operations that the subjects can execute on the given objects. For example, a file has an ACL that contains (Mary, read), gives Mary permission to read the file (see Table 3.2).

| Table 3.3: Capability-Based Access Control |
|-------------------|-------------------|
| Mary              | file1, read       |
|                   | file2, write      |

Capability-based access control (CBAC) [85, 86] is also a DAC-based access control solution. It is implemented with a row centric view of an access control matrix in which each row of a matrix is translated to a capability. In CBAC, the subjects have their capability lists from the access control matrix to access the given objects. For example, Mary has the following capability lists: (file1, read) and (file2, write), i.e., Mary has permissions to read file1 and to write into file2 (see Table 3.3).
3.1. Access Control

The simplicity and flexibility are the key reasons why these two DAC-based solutions (ACL and CBAC) are widely accepted and mostly used by the operating systems and database systems. However, ACL and CBAC do not provide adequate functionalities to model the global policy rules for the large group of users.

3.1.4 Role-Based Access Control

A well-accepted traditional access control model based on the roles of the users is role-based access control (RBAC) [40, 84, 83, 86], which was introduced to tackle the problems of DAC and MAC-based access control solutions.

The Role-based Access Control (RBAC) concept emerged in the early 90s as a proven technology for managing and enforcing security in large-scale domains. Ferraiolo et al [40] have proposed the RBAC model, named the RBAC92 model, identifying the fundamental concepts related to the concept of roles. The RBAC92 model has been subsequently extended by Sandhu et al [84] in order to propose a conceptual RBAC framework that can be used as a basis for implementing RBAC-based solutions, named the RBAC96 model.

In RBAC96, it is not important to identify the individual who makes the access request, but to know his position in an organization. This concept is known as role in the RBAC96 model, in which the access permissions are not assigned directly to a particular user, but to the user’s role.

The RBAC96 model is conceptually simple, and it has four main concepts: users, roles, sessions and permissions (see Figure 3.1). It ensures that only an authorized user is given access to a certain resource, and is based on a user’s role in an organization. In the RBAC96 model, subjects are typically represented by users, who are assigned to roles based on their credentials in the organizations, permissions (resource access permissions) are assigned to roles and roles represent the job functions, describing the authorities and responsibilities conferred on the users assigned to these roles, within the organizations. The roles form the basis of RBAC96 policies and users acquire resource access permissions
Users' assignments in roles can be revoked easily, new assignments can be established as needed and roles can be updated without updating the permission for every user on an individual basis. Users can create the sessions and choose to activate some subset of user roles and the sessions terminate at the users' initiative.

Though role-based access control provides many advantages in large-scale domains, it does not provide adequate functionalities to incorporate contextual information in the access control process, which is needed to specify context-aware user-role assignment and role-permission assignment policies.

### 3.1.5 Attribute-Based Access Control

The attribute-based access control (ABAC) approach has also received considerable attention and popularity in the IT industry [72, 92]. ABAC grants accesses to resources and services based on the attributes of relevant entities.
3.1. Access Control

Wang et al [92] have proposed a logic-based framework for attribute-based access control specification and enforcement. The ABAC model is a rule-based approach to access control. ABAC makes an authorization decision based on attributes to allow access to a resource according to policy rules created through the security administrator. The attributes of the subjects and objects are used in the ABAC model [92]. The ABAC policy rules define which constraints need to be satisfied based on attributes of entities (e.g., user is located at emergency room), determine how accesses to resources are controlled, and according to which access control decisions are consequently made.

The attribute-based access control model, which grants accesses to resources and services based on the attributes of entities (e.g., user’s identity - the attribute possessed by the user/requester) rather than direct role-based support, does not have adequate functionalities to manage large number of policy rules for the huge number of users in large-scale domains [71]. Though ABAC provides effective solution in a relatively simple domain, considering complicated administration tasks and management efforts, it is not adequate in large-scale domains. Furthermore, similar to RBAC, it does not have dynamic access control decision making capabilities.

3.1.6 Discussion

One of the biggest limitations of the early basic access control models (such as AM [69], ACL [45, 82, 86] and CBAC [85, 86]) is that they manage access privileges based on individuals, instead of groups of individuals. Whenever the numbers of subjects and objects are high, the number of authorizations can become extremely large, which complicates the administration tasks. This brings high complexity of security administration and significant cost of management of growing large-scale systems.

RBAC normally provides effective support in closed and relatively unchangeable systems. Moreover, in the RBAC model, the user and permission assignments are static without taking into account the dynamic contextual information, such as the location and time of the users. For example, in our application scenario (presented in Chapter 2), access to medical records of the patients is dynamically controlled depending on the
Chapter 3. Background and Literature Review

contextual information (e.g., the location and request time of the nurses and doctors). A nurse who is not located in the hospital should not have access to medical records of the patients. Moreover, this access is permitted only during her ward duty time. RBAC does not provide adequate functionalities to incorporate and adapt to the contextual information, in determining whether access permissions are granted or denied.

On the other hand, ABAC [72, 92] is easy to set up but complex to manage in large-scale domains, because of the huge number of attributes of entities. Therefore, the attribute-based approach does not scale well in large open systems [71].

Above all, the traditional access control models and solutions were developed when computing was typically in a static environment. Today’s environment is highly dynamic and needs to be context-aware, and granting a user access without taking the dynamically changing information into account can compromise security as the user’s access privileges not only depend on “who the resource requester and resource owner are” but also on “what the relevant dynamically changing contextual information is”. As such, the traditional access control models and solutions do not provide adequate support for today’s dynamic and context-aware environments.

Although a significant amount of research efforts have been done in the area of traditional forms of access control, limited research efforts have been undertaken to incorporate context-awareness capabilities into making access control decisions. Towards this goal, in the next sections, we briefly analyse the existing research efforts that are related to our research. We discuss the context and situation-aware access control approaches and frameworks that consider contextual information when making access control decision. These access control approaches and frameworks adopt and extend the basic RBAC and ABAC approaches in order to incorporate relevant contextual information into the access control processes.

We classify the existing context-specific access control approaches and frameworks into three groups:
3.2. Context-Aware Role-Based Access Control

- Context-aware role-based access control
- Context-aware attribute-based access control
- Situation-aware access control

3.2 Context-Aware Role-Based Access Control

Some context-sensitive access control approaches exist in the literature that adopt the basic RBAC approach [40, 84] and integrate different types of dynamic contextual information into user-role and role-permission assignments (e.g., [18, 55, 19, 25, 68, 48, 53, 87]).

3.2.1 Temporal Role-Based Access Control

Bertino et al [18] have proposed the temporal RBAC (TRBAC) approach, which extends the traditional RBAC approach in order to support temporal constraints on enabling/disabling roles. They have defined a concept, named role enabling base (REB), to describe temporal constraints on the enabling of roles. The REB is composed of periodic events (PE) and role triggers (RT). Periodic events have the form \((I, P, p:E)\), where \(I\) is a time interval, \(P\) is a period expression, and \(p:E\) is a prioritized event expression. For example, the periodic events and role triggers in the REB state that the doctor-on-night-duty role must be enabled during the night, whereas the role doctor-on-day-duty must be enabled during the day.

Joshi et al [55] have extended the TRBAC approach proposed in [18]. They proposed a generalized temporal role based access control (GTRBAC) model that allows specification of a comprehensive set of time-based access control policies, incorporating the temporal constraints in both user-role and role-permission assignments.

The above mentioned temporal RBAC approaches [18, 55] take into account temporal information when enforcing access control policies. In our application scenario (presented in Chapter 2), a nurse should not have access to medical records of the patients other than satisfying some contextual conditions (during ward duty time, from
the hospital location, relationship between them are assigned nurse, etc.). Other than the
temporal information, it is also desirable to consider other relevant contextual information
when defining and enforcing access control policies.

3.2.2 Spatial Role-Based Access Control

A well-known spatial access control approach is GEO-RBAC [19], which extends the
traditional RBAC approach with the concept of spatial role. A spatial role represents a
departmentally bounded organizational function. The boundary specifies the spatial extent
in which the user is located and enabled to play such a role (spatial role). In GEO-RBAC,
access control to resources and services is based on the location of the users and their
assigned roles.

Zhang et al [101] have proposed a location-aware RBAC approach named LR-BAC. In the LR-BAC approach, the concepts of spatial role and effective role (like RBAC roles)
are introduced. LR-BAC models the user locations and departmentally bounded roles. These roles are automatically activated/deactivated by the locations of the users. In
LR-BAC, both the activated roles of the users and their locations are taken into account
in order to evaluate the access control policies.

These spatial RBAC approaches [19, 101] take into account the spatial information
when enforcing access control policies. However, they do not consider other relevant
contextual information.

3.2.3 Spatial and Temporal Role-Based Access Control

Chandran et al [25] have extended the GTRBAC approach [55], and proposed a location
and time-based RBAC approach, named LoT-RBAC. LoT-RBAC addresses access control
requirements of dynamic environments in order to provide location and time-based access
control. LoT-RBAC has three main RBAC entities, users, roles and permissions, and
considers temporal and spatial contextual information. LoT-RBAC adopts and extends
the concepts of role activation and role assignment from the basic RBAC approach,
3.2. Context-Aware Role-Based Access Control

according to temporal and spatial contextual information. In particular, a role is activated by a user from the location \( l \) at time \( t \), if the location and temporal information of a user associated with the role activation are satisfied.

Bhatti et al [21] have proposed a context-aware access control approach named X-GTRBAC. The X-GTRBAC approach extends the generalized temporal role-based access control (GTRBAC) approach [55]. GTRBAC provides a mechanism to express a diverse set of fine-grained temporal constraints on user-role and role-permission assignments in order to meet the dynamic access control requirements of an enterprise. X-GTRBAC adopts the temporal-aware user-role and role-permission assignment policies from the GTRBAC approach, and considers spatial context information in these assignments.

The above mentioned spatio-temporal RBAC approaches [25, 21] consider spatial and temporal information when specifying and enforcing access control policies. These approaches have similar drawbacks in capturing only the specific types (temporal and spatial) of contextual information.

3.2.4 User-Specific Context and Role-Based Access Control

Al-Kahtani and Sandhu have introduced an extended RBAC approach through rules, called rule-based RBAC (RB-RBAC) [12]. In this approach, users are dynamically assigned to roles based on a finite set of assignment rules derived from the security policy. These rules take into consideration of the attributes of users as contextual information. Similar to RB-RBAC, Kern and Walhorn have adopted RBAC approach and proposed a rule-based provisioning system for the RBAC approach based on a limited set of user attributes as contextual information [64]. These approaches have the limitation of considering only the contextual information about users as policy constraints.

Zheng et al [103] have proposed a dynamic role-based access control (DRBAC) approach, which incorporates the required credentials of users as contextual information when making user-role and role-permission assignments. The DRBAC approach extends the basic RBAC approach and it dynamically grants and adapts permission to users
according to current contexts. Context information is used to decide which role is active, and which permission is active for that role. In the DRBAC approach, the authors only present the concepts and requirements of the dynamic access control, without providing context and policy modelling supports for dynamic context-aware decision making.

3.2.5 Contextual-Role and Role-Based Access Control

Covington et al. [30, 31] have addressed the problem of securing applications in the home of the future, i.e., smart homes, and proposed a generalized role-based access control (GRBAC) approach. GRBAC adopts and extends the basic RBAC approach by incorporating the notion of object and environment roles with the traditional subject roles. Subject roles (e.g., adult, child) are analogous to the traditional RBAC roles for users, whereas object roles (e.g., image, source code) are basically a group of object instances computed algorithmically on the basis of certain security-related properties of the objects. The environment roles are used to model the environment context (e.g., time, location). Environment roles can be activated based on the value of conditions in the environment where the request has been made. By defining these three types of roles, the GRBAC approach uses information gathered from the environment sensors as a determining factor for making access decisions.

Park et al. [77] have presented a context-role based access control (CRBAC) approach aiming to increase resource privacy and confidentiality. The basis of this approach is RBAC with extension of a context-role hierarchy. Context-role represents the environment state of the system when making access control decisions. In the CRBAC approach, a transaction specifies a particular action to be performed on an object based on environment context and it is represented as a tuple in the form of $<\text{user\_role, context\_role, permission}>$.

Wang et al. [91] have proposed a context-aware environment-role-based access control (CERBAC) approach for web services. They consider subject roles and environment roles, and access control decisions in CERBAC not only depend on the subject roles but also on environment roles. In CERBAC, the environment conditions are specified and modeled by the contextual information and they are used to define the environment roles.
3.2. Context-Aware Role-Based Access Control

The unification of all relevant states (user, resource and environment states) into a single concept (roles) makes access control policies significantly easier to define and implement.

However, these approaches [30, 31, 77, 91] are not suitable for large and complex organizations, because of the many roles (subject/user roles, object/resource roles and environment/context roles) making the system very hard to maintain.

3.2.6 Team-Based Access Control

The contextual team-based access control (C-TMAC) approach [44] extends the basic team-based access control (TMAC) approach [89] from two viewpoints, to give a framework to integrate TMAC concepts with RBAC, and to extend TMAC to use general contextual information. In the C-TMAC approach, the users obtain permissions on resources according to their roles and the teams in which they are involved. A team is defined as a group of users acting in different roles with the objective of completing a certain task, for example, a group of physicians and nurses attending in a patient’s care. C-TMAC includes the general context information (such as temporal, spatial and transaction-specific contexts) as predicates to specify access policy rules. Thus, in C-TMAC the team concept is used to associate users (and user playing roles) with contexts, like user-role assignments in RBAC [40, 84]. Though C-TMAC considers some important contextual dimensions, the approach is still limited in considering diverse contextual information for specifying context-aware user-role and role-permission assignment policies.

3.2.7 Organization-Based Access Control

The above-mentioned access control approaches address role-based access control decision making in the context of a single organization. Researchers have also been addressing the challenges of developing languages and frameworks for cross-organizational access control (e.g., [56], [33], [78]).

The organization-based access control (OrBAC) approach [56] is an access control policy model that is explicitly designed to allow management policies to be applied in the
multi-organizational settings. OrBAC tries to overcome the limitations of previous access control model such as RBAC [84] by considering the concept of organization together with the concept of context. Cuppens et al [33] specifies the contextual security policies of organizations. They incorporate the contextual information into the organization-based access control policies. OrBAC has two levels of abstraction: an organizational level that includes the role, activity, view and context concepts, and a concrete level that includes the subject, action and object concepts. The central entity in the OrBAC approach is called Organization.

Preda et al [78] have extended the basic OrBAC approach for dynamically deploying security policies and proposed context-aware access control policies in terms of resource access and resource availability, system vulnerabilities and threats. The core and extended OrBAC approaches consider different types of contextual information (such as temporal, spatial, provisional, prerequisite and user-declared contexts) as policy constraints, however they are still limited considering several further contextual information such as the relationships or some implicit information through context inference. In addition, these approaches do not provide technical details of how the important contextual information specific to access control can be captured/inferred dynamically and incorporated into user-role and role-permission assignments for dynamic access control decision making.

3.2.8 Context-Specific Role-Based Access Control

Several other access control approaches (e.g., [58, 68, 48, 87]) have adopted and extended the basic RBAC approach by considering some important contextual information.

Kapsalis et al [58] have presented a context-aware access control approach based on the basic RBAC approach. The approach incorporates dynamic context information, in the form of context constraints. The authors classify context, based on the requirements imposed by the provision of e-services to the industrial domain, into four main categories (user, resource, environment and history dimensions). They provide a multilevel context model based on the simple and composite context conditions. Also, the paper deals with
3.2. Context-Aware Role-Based Access Control

implementation issues and presents a prototype framework that uses the proposed access control mechanism in a web service infrastructure.

Kulkarni et al [68] have proposed a context-aware RBAC (CA-RBAC) approach for pervasive applications. They consider user and resource attributes as the context constraints and use this context information in role admissions policies, revocation of role memberships and permission activations. Users can activate personalized access permissions in addition to their roles, thus having a dynamic role-permission assignment.

Recently, He et al [48] have proposed an access control approach for web services based on user role and presented a context-aware policy model considering the user, resource and environment attributes.

Another recent approach to context-aware RBAC for business processes in ubiquitous computing environments has been proposed by Schefr-Wenzl and Strembeck [87], which is an extension of the traditional RBAC approach. The authors propose a formal metamodel that extends the UML, to integrate context attributes into RBAC. As such, they specify the context-aware RBAC approach for business processes. In this approach, the context attributes represent a certain properties of the environment whose actual values change dynamically (such as time, date and location).

While these recent role-based approaches [58, 68, 48, 87] consider a broader range of contextual dimensions and provide useful insight for modeling context-aware and role-based access control concepts in our CAAC approach, they are still limited in considering other types of environmental factors or context information (e.g., different types of relationships) and context inference to reason about some richer context information from the available information. In general, there lacks an extensible context model that can capture the wider range of important context information, a reasoning technique to infer richer context information, and a context-aware policy model to incorporate these different types of contextual information to provide context-specific access to resources and services.
3.2.9 Comparative Analysis and Discussion

Concerning the application scenario presented in Chapter 2, our CAAC framework needs
(i) a **context model** to represent and capture the relevant context entities and contextual
information, and consequently needs (ii) a **policy model** to support dynamic access control
decision making, using dynamic associations of users to roles and roles to permissions.

- Capturing the broad categories of access control-specific context entities (CE).
- Capturing the basic contextual information characterizing the entities (EC).
- Capturing the basic contextual information characterizing the relationship between
  entities (RC).
- Reasoning about high-level contextual information from the basic information
  through context inference (CI).
- Specifying context-aware user-role assignments (CAURA).
- Specifying context-aware role-permission assignments (CARPA).

Table 3.4 shows a comparative analysis of the existing context-aware role-based frame-
works. With respect to our CAAC framework needs, we use a ✓ if a related framework
provides similar and/or comparable support for a certain aspect, a △ if a related
framework provides partial support for a particular aspect, and a – if a related framework
provides no support for a particular aspect.

The basic role-based access control (RBAC) approach has become the most widely
used access control model [75]. It typically evaluates access permission through roles
assigned to users without taking into account the contextual information. On the
other hand, some of the extended RBAC approaches incorporate temporal and spatial
information into user-role and role-permission assignments (e.g., [18, 55, 19, 25]). Re-
cently, several other extended RBAC approaches have considered some further context
information other than the temporal and spatial dimensions, such as the resource and
environment dimensions (e.g., [58, 48]) as well as the user dimension (e.g., [12, 103]).
3.2. Context-Aware Role-Based Access Control

Table 3.4: Comparative Analysis of the Existing Context-Aware RBAC Frameworks

<table>
<thead>
<tr>
<th>RBAC Frameworks</th>
<th>Context Model</th>
<th>Policy Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CE</td>
<td>EC</td>
</tr>
<tr>
<td>Temporal RBAC [18, 55]</td>
<td>-</td>
<td>Δ</td>
</tr>
<tr>
<td>Spatial RBAC [19, 101]</td>
<td>-</td>
<td>Δ</td>
</tr>
<tr>
<td>Spatio-Temporal RBAC [25, 21]</td>
<td>-</td>
<td>Δ</td>
</tr>
<tr>
<td>User-Specific Context and RBAC [12, 103]</td>
<td>Δ</td>
<td>Δ</td>
</tr>
<tr>
<td>Contextual-Rule and RBAC [30, 31, 77, 91]</td>
<td>Δ</td>
<td>Δ</td>
</tr>
<tr>
<td>TMA [44, 89]</td>
<td>-</td>
<td>Δ</td>
</tr>
<tr>
<td>OrBAC [56, 33, 78]</td>
<td>-</td>
<td>Δ</td>
</tr>
<tr>
<td>Context-Specific RBAC [58, 68, 48, 87]</td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

Even though the existing role-based approaches consider some important context information as constraints in the access control policies, there is still a need for a general context representation and reasoning model to capture, infer and manage the relevant context information. To fill this gap, in this thesis we have introduced a general and extensible ontology-based context model. It supports the modelling and capture of access control-specific context information (about the relevant entities and the relationships between entities) and the inference of high-level implicit context information. In particular, we have introduced several new general concepts for context modelling and reasoning, including resource owner, relationship between different persons and resource granularity. Other than context modelling and reasoning, there is also a need for a new policy model, to incorporate diverse contextual information into access control policy for dynamic access control decision making. Towards this end, in this thesis we have also introduced a policy model that extends the RBAC policy model, by incorporating diverse contextual information and providing access control decision making at different levels of resource granularity in terms of context-aware user-role and role-permission assignments.
3.3 Context-Aware Attribute-Based Access Control

Some access control approaches have integrated different types of contextual information into the attribute-based access control (ABAC), to provide access control to information resources and software services in a context-aware manner (e.g., [13, 28, 54, 90, 32, 29, 53]).

Muhtadi et al [13] have presented a context-aware security scheme named Cerberus, which integrates identification, authentication, context-awareness and reasoning. Cerberus enhances the security of ubiquitous applications that are built using the Gaia context infrastructure [81]. Cerberus considers contexts as first-order predicates. The name of the predicate is the type of context that is being described (like location, temperature and time). Corradi et al [28] have proposed an approach to context-aware access control in ubiquitous computing environments, in which permissions are directly associated with contexts, instead of user roles. They distinguish two different kinds of context information: physical contexts (geographical coordinates and user device) and logical contexts (temporal conditions, user activities, resource availability and resource status). Moreover, they propose a dynamic security middleware, called UbiCOSM (ubiquitous context-based security middleware), that adopts context as the basic concept for the security policy specification and enforcement processes. However, these approaches do not consider some other important context information such as the relationships between different entities.

In focusing on access control in pervasive computing environments, Toninelli et al [90] have proposed a semantic approach to context-aware access control based on context-aware policies for secure collaboration in pervasive computing environments. The approach provides resource access permission on the basis of context (resource availability, roles/identities of actors and some other environmental conditions such as time and location). In this approach, the authors consider three categories of contextual information: actor (requester and owner), resource and environment dimensions. This access control approach considers context as a first-class principle for policy specification. The authors propose an ontology-based context model and based on this model, they propose a context-aware policy model. Similar to the above attribute-based approaches, however,
3.3. Context-Aware Attribute-Based Access Control

its context model does not consider several concepts that are important for access control in today’s dynamic and context-aware environments, including interpersonal relationships between user and owner, the derived entity status (e.g., health status) information, etc. In addition to these concepts, our approach also supports resource hierarchy, and consequently user’s access to resources at different levels of granularity.

Hulsebosch et al [54] have proposed a context-sensitive access control (CSAC) approach based on the user-specific context information, such as user’s location information and user’s previous access history. The approach consists of setting up an access control architecture related to context-aware service provisioning. It provides an access controller which grants or denies users to perform operations on the resources according to the access policies. The CSAC approach also has limitations in considering a limited set of contextual information in the access control policies, such as user’s location and access history are considered in this approach.

Covington et al [32] have introduced a contextual attributes-based access control (CABAC) approach, integrating contextual attributes in the access control process. They consider four types of contextual attributes, i.e., attributes associated with users, resources, transactions and environments. In CABAC, a contextual attribute represents a measurable contextual primitive, such as location and time (environmental attributes). This approach also has similar drawback in considering a limited set of contextual information and context inference to infer richer contextual information from this information.

A recent attribute-based context-aware access control framework for the web of data is grounded on two ontologies, which deal with the context concepts and the core access control policy concepts [29]. Its consideration of context is still limited to the user, device and environment dimensions. Furthermore, it does not have the capability of inferring high-level implicit contexts.

Another recent access control approach to integrating ABAC with RBAC has been proposed by Huang et al [53]. In this work, the authors consider attribute-based policies
and provide a formal language for their modelling. The main building blocks of this approach are the attributes of entities. It considers the attributes of entities (attributes of user, role and resource and environmental attributes such as time and device). In particular, the approach defines roles in terms of attributes. While providing useful insight into considering different types of attributes as policy constraints, this approach does not have a context model, to capture and infer dynamic contextual information. Consequently it does not offer any specific ways of modelling and incorporating context information in the access control policies.

Other recent attribute-based research works have integrated the relationship dimension into policy specifications (e.g., [42], [102]). Zhang et al [102] have proposed a relation-based access control approach, RelBAC, in which the permissions are formalized as binary relations between subjects and objects. They consider subject, object and permission as the compulsory access control components. Fong and Siahaan [42] have proposed a relationship-based access control approach, ReBAC, which considers the relationships between individual users (e.g., friend, friend-of-friend) in the access control policies. Similar to the other traditional access control approaches, these relationship-based approaches do not consider context-awareness, even though they have provided us with useful insights for modelling relationship information.

XACML is a general-purpose authorization policy approach standardized by OASIS [73]. It adopts the basic attribute based access control approach, supports attributes to subjects and objects, and the specification and enforcement of access control policies using these attributes. A standard role-based access control profile for XACML has been issued by OASIS [14]. In this profile, roles are expressed as XACML subject attributes. While providing resource access permission by considering the attributes of user (identity/role), resource (identity) and environment (location and time), XACML does not consider the broader range of context-awareness capabilities such as the relationships or context inference.
3.3. Context-Aware Attribute-Based Access Control

3.3.1 Comparative Analysis and Discussion

Similar to role-based access control frameworks described in Section 3.2, the above-mentioned attribute-based frameworks also incorporate different types of contextual information into the access control policies. We analyse these attribute-based access control frameworks based on the same requirements identified in Section 3.2.9, context entities (CE), entity context (EC), relationship context (RC), context inference (CI), context-aware user-role assignments (CAURA), context-aware role-permission assignments (CARPA). The only difference here in policy model, as attribute-based frameworks directly incorporate the contextual information into the access control policies, context-aware user-resource assignments (CAURResA).

Table 3.5: Comparative Analysis of the Existing Context-Aware ABAC Frameworks

<table>
<thead>
<tr>
<th>ABAC Frameworks</th>
<th>Context Model</th>
<th>Policy Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CE</td>
<td>EC</td>
</tr>
<tr>
<td>Cerberus [13]</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>CAAC [28]</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>CAAC [90]</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>CSAC [54]</td>
<td>–</td>
<td>△</td>
</tr>
<tr>
<td>CABAC [32]</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>CAAC [29]</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>ABRBAC [53]</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>ReBAC [42]</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>XACML [73, 14]</td>
<td>△</td>
<td>△</td>
</tr>
</tbody>
</table>

Table 3.5 shows a comparative analysis of the existing context-aware attribute-based frameworks. With respect to our CAAC framework needs, we use a $\checkmark$ if a related framework provides similar and/or comparable support for a certain aspect, a $\triangle$ if a related framework provides partial support for a particular aspect, and a – if a related framework provides no support for a particular aspect.

In general, the existing attribute-based access control approaches support the speci-
Chapter 3. Background and Literature Review

Specification and enforcement of access control policies in terms of specific types of contextual information, with each of them having different origins, pursuing different goals and often, by nature, being highly domain-specific.

These attribute-based approaches and their associated context models are still limited in capturing and inferring a wide range of contextual information relevant to access control. Consequently, the policy models associated with these approaches do not have adequate functionalities to incorporate diverse contextual information (dynamic attributes) into the access control processes.

Furthermore, the attribute-based access control approaches have major limitations when applied in large-scale domains because they grant access permissions based on attributes (e.g., user’s identity), and consequently they do not scale well in large open systems [71]. In addition, these attribute-based approaches do not directly treat roles as first class entity in their access control policies. As a consequence, they are not able to offer the advantages of role-based access control (i.e., the role hierarchy, role inheritance, user-role assignments, role-permission assignments, etc.).

Though in this thesis, we adopt and extend the basic role-based access control concept, the attribute-based access control concept provides us useful insight for modelling a wide range of contextual information.

3.4 Situation-Aware Access Control

The above-mentioned context-aware access control approaches are the improved/extended access control mechanisms, to control resource access on the basis of dynamic contextual information. On the other hand, the situation-aware access control approaches control resource access on the basis of situation information (i.e., specific combinations of contextual information).

Some situation-aware access control approaches adopt and extend the basic role-
3.4. Situation-Aware Access Control

Based access control (RBAC) approach [84], incorporating different types of dynamic contextual information into their access control policies (e.g., [98, 99, 65, 43, 97]).

Yau and others have defined situation as a set of context attributes of users, systems and environments over a period of time affecting future system behavior [98]. Based on this definition, recently, Yau and Huang [97] have presented an ontology-based situation model for service-based systems to ensure the satisfaction of user’s quality-of-service (QoS) requirements in dynamically changing environments.

Yau and Liu have presented a situation-aware access control (SA-AC) based privacy-preserving service matchmaking approach [99]. The SA-AC approach incorporates situation-aware constraints into basic RBAC approach, such that the dynamic states of service providers, requesters and environments are considered, in order to make access control decisions. The SA-AC model includes these constraints into both user-role and role-permission assignments as situations. SA-AC provides an XML-based access control language for specifying flexible access control policies.

Kim and Lim have proposed the situation-aware RBAC (SA-RBAC) approach [65], which has extended the basic RBAC approach. SA-RBAC grants users to roles and roles to permissions based on the situation information. The SA-RBAC approach is used to deal with the situation-aware decision making by considering the combination of the required credentials of user as situation.

Garcia-Morcion and Wehrle [43] have proposed a two-layer modular context-aware access control approach for medical sensor networks comprising a data layer and an engine layer. The data layer comprises all the information (including context information such as location and time) that is required for access control decisions. The engine layer manages the access control decisions in critical, emergency and normal situations using this context information and access control policies. This access control approach adopts the basic RBAC approach [84] and incorporates the situation information into access control process.
3.4.1 Comparative Analysis and Discussion

Concerning the application scenario presented in Chapter 2, our CAAC framework needs (i) a **situation model** to identify and infer the relevant situations, using the contextual information from the context model, and consequently needs (ii) a **policy model** to support dynamic access control decision making, using dynamic associations of users to roles and roles to permissions.

- Identifying the states of the context entities (**ES**).
- Identifying the states of the relationships between entities (**RS**).
- Identifying the user’s intention or purpose of resource access (**UI**).
- Correlation of ES, RS and UI to infer the atomic situations (**AS**).
- Inferring the composite situations (**CS**) from the atomic situations.
- Specifying situation-aware user-role assignments (**SAURA**).
- Specifying situation-aware role-permission assignments (**SARPA**).

<table>
<thead>
<tr>
<th>Models and Frameworks</th>
<th>Situation Model</th>
<th>Policy Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ES  RS  UI  AS</td>
<td>SAURA  SARPA</td>
</tr>
<tr>
<td><strong>Situation Models</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Situation Model [98]</td>
<td>△  -  -  -   -</td>
<td>-  -</td>
</tr>
<tr>
<td>Situation Ontology [97]</td>
<td>△  -  -  △  △</td>
<td>-  -</td>
</tr>
<tr>
<td><strong>Situation-Aware Access Control Frameworks</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA-AC [99]</td>
<td>△  -  -  -   -</td>
<td>△  △</td>
</tr>
<tr>
<td>SA-RBAC [65]</td>
<td>△  -  -  -   -</td>
<td>△  △</td>
</tr>
<tr>
<td>Modular SAAC [43]</td>
<td>△  -  -  -   -</td>
<td>-  -</td>
</tr>
</tbody>
</table>

Table 3.6 shows a comparative analysis of the existing situation-aware models and frameworks. With respect to our CAAC framework needs, we use a √ if a related...
framework provides similar and/or comparable support for a certain aspect, a △ if a related framework provides partial support for a particular aspect, and a — if a related framework provides no support for a particular aspect.

The existing situation-specific approaches consider situation as the specific combinations of context information (e.g., the states of the relevant entities). They do not consider other important concepts which are relevant in dynamic and context-aware environments: the states of the relationships between different entities and the purpose or user’s intention in accessing resources. Therefore, on top of context model, there is still a lack of a situation model specific to access control by considering these situation constraints (the states and purpose), to identity the relevant atomic situations and reason about composite situations. There is also a lack of a policy model to incorporate these situations into both user-role and role-permission assignments for dynamic access control decision making.

3.5 Context Modelling, Reasoning and Management

In the field of context representation models, there have been several research works (e.g., [26, 93, 50, 46, 39, 52, 96, 79]) for modelling context information representing the entities such as person, place and object.

Chen et al [26] have proposed the SOUPA ontology based on OWL [11] for ubiquitous and pervasive applications where the basic concepts are person, agent, time, space, action, policy, belief-desire-intention (BDI) and event. Wang et al [93] have presented an OWL encoded CONtext ONtology (CONON) for modelling context in pervasive computing environments. The CONON model helps to share a common understanding of the structure of contextual information concerning users, places and devices in order to support semantic interoperability and reuse of domain knowledge. Using the CONON model, through the creation of reasoning rules, high-level contextual information (e.g., user is ‘sleeping or watching-TV’) can be deduced from the relevant low-level contextual information [46]. Henricksen et al [50] have developed a context modelling language
(CML) and a tool that translates CML-based context models to an OWL representation for the purpose of utilizing the OWL technology in pervasive computing systems. The context model proposed by Esposito et al [39] provides some important concepts for the domain-specific healthcare ontology. The context models proposed by Zhang et al [52], Xiong et al [96] and Reichle et al [79] consider application-specific context modelling concepts in pervasive computing environments.

The above general and domain/application-specific context models do not provide direct support for context entities and information related to access control such as context entities and information related to user roles, resource owners, or for relationships between context entities such as that between the resource requesters (users) and the resource owners.

Some context representation and reasoning techniques (e.g., ontology-based) have been developed over the last decade ranging from very simple early models to the current state-of-the-art context models (e.g., [100, 24, 57, 17, 51, 20, 80]).

Chaari et al [24] have analysed the state of the art context modelling approaches and propose a way of modelling context ontology in pervasive environments at two levels: a generic level (generic/upper context ontology) and a domain specific level (domain-specific ontologies). In this thesis, we follow the modelling of upper and domain-specific concepts.

Bettini et al [20] and Ye et al [100] have presented a survey of ontology based context modelling approaches in pervasive computing systems. They argue that the combined adoption of the ontology-based approach with the logic-based approach is capable of dealing with a wide range of reasoning styles. In this regard, we use the OWL-ontology [11] for representing basic context concepts and SWRL-rule [9] for reasoning about complex and implicit concepts (high-level context information) in this thesis.

Kapitsaki et al [57] have discussed a survey of context handling approaches for
3.6. Overall Discussion

context-aware service engineering. They use a tourist service example to illustrate precisely how these approaches can be used, and propose a context model for this tourist application by considering the location, time, user preferences and weather information as contexts. While the proposed context model is application-specific, it provides a useful insight into modelling the interaction between context and service.

3.5.1 Discussion

Even though the above-mentioned context modelling, reasoning and management approaches consider context information representing the context entities such as person, place and object, they are still limited in considering the access control-specific context entities and information. Therefore, there is a need for a new context model to capture, infer and manage the important context information specific to access control. On top of context model, there is also need for a situation model to identify and reason about relevant situation information using this context information and consequently a need for a policy model to incorporate this context and situation information for dynamic access control decision making.

Ontology-based modelling has been the most practical choice for context-aware applications because of its considered trade-off between computational complexity of reasoning and expressiveness [80]. Existing research efforts (e.g., [20]) also show that ontology-based approaches are very suitable for modelling context entities and information. In this thesis, we adopt an ontology-based approach to model contexts, situations and access control policies.

3.6 Overall Discussion

The existing context-aware access control approaches have only considered specific types of contextual information, each of them having different origins, pursuing different goals and often, by nature, being highly domain-specific. On the other hand, the general context modelling and management approaches do not provide direct context modelling support for concepts related to access control. Overall, access control that considers a wide range
of important dynamic context information with reasoning capabilities is largely missing in the literature. Therefore, there is a need for an access control-specific context model to capture and model the diverse contextual information relevant to access control. In addition, there is a need for inferring high-level (implicit) context information from the available context information using user-defined reasoning rules, extending the context model.

The existing situation-aware access control approaches consider specific combinations of certain contextual information (e.g., the states of the relevant entities). However, they do not consider the states of the relevant relationships between different context entities or the purpose or user's intention in accessing resources and services. In dynamic and context-aware environments, these concepts are also important considerations in access control decision making. Therefore, there is also a need for a new situation model to identify the relevant atomic situations, and a reasoning technique to infer new composite situations from one or more atomic situations using user-defined rules.

The existing policy models associated with access control approaches incorporate the specific types/combinations of contextual information in the access control processes. The attribute-based access control approaches directly integrate this information in the access control policies, whereas the role-based access control approaches incorporate this information into both user-role and role-permission assignments. However, these policy models do not have adequate functionalities to incorporate diverse context and situation information into access control policies for dynamic access control decision making. Therefore, there is also the need for a context-aware policy model for dynamic access control decision making, incorporating context and situation information into access control policies.
3.7 Summary

In this Chapter, we have reviewed the existing access control frameworks for information resources and software services ranging from the traditional forms of access control approaches to application-specific approaches. We have divided the related research into five categories.

Firstly, we have described the traditional access control models and technologies. Secondly, we have discussed the access control frameworks that incorporate different types of context information into role-based access control. Thirdly, we have described the access control frameworks that incorporate the contextual information in the access control processes as attributes. Fourthly, the access control frameworks that incorporate the situation information into the access control processes are discussed. Finally, we have discussed the context modelling, reasoning and management frameworks for representing and inferring different types of context information. We have highlighted the limitations and shortcomings of these frameworks that motivate us to develop a new context-aware access control framework for software services.
Part II
Due to the rapid advancement of communication technologies, the ability to support access control to information resources or software services in open and dynamic environments is crucial. On the one hand, users demand access to resources or services in an anywhere, anytime fashion. On the other hand, additional challenges arise in ensuring the privacy and security requirements of the stakeholders in dynamically changing environments. Towards this goal, in this Chapter we propose a novel Context-Aware Access Control (CAAC) framework that extends the basic Role-Based Access Control (RBAC) framework with dynamic associations of user-role and role-permission assignments. We introduce a formal model for specifying our CAAC framework including its basic elements. The framework uses the relevant dynamic information (context/situation) in order to enable context-aware user-role assignment and context-aware role-permission assignment. We take context to mean any information that can be used to characterize the state of a relevant entity or the state of a relevant relationship between different entities. A situation is defined as a specific subset of the complete state of the universe of the relevant entities that are relevant to a certain goal or purpose of a service access request.

The Chapter is organized as follows. Section 4.1 presents background, the basic concepts of RBAC. Section 4.2 presents a formal model of our context-aware access control framework. Section 4.3 provides the formal description of the basic elements of the framework. Finally, Section 4.4 summarizes the chapter.
Chapter 4. Conceptual Framework for Context-Aware Access Control

4.1 Background

The traditional role-based access control (RBAC) model has become the most widely used access control paradigm for managing and enforcing security in large-scale domains [75]. The model has the following main elements: users, roles and permissions. In RBAC, the users are human-beings, who are assigned to roles based on their credentials in the organizations, roles represent the job functions within the organizations, describing the authorities and responsibilities conferred on the users assigned to these roles, and permissions are the approvals to perform certain operations on resources.

In the RBAC model, the access permissions are not assigned directly to the particular users, but to the user’s roles. That is, the central notion of RBAC is that users are assigned to appropriate roles, permissions are assigned to roles, and users can have appropriate permissions by being members of such roles. RBAC ensures that only an authorized user is given access to a certain permission, and is based on a user’s role in an organization. As such, it simplifies the management of access control policies by creating user-role and role-permission assignments.

Currently, RBAC has received broad support as a generalized access control model and is well recognized for its many advantages in performing large-scale authorization management [41]. RBAC can be configured to support a wide variety of access control policies, including traditional discretionary access control (DAC) and mandatory access control (MAC) [10], as well as organization-based access control policies [56, 78]. However, the RBAC model does not directly adapt the requirements of the context-aware access control applications (presented in Chapter 2). In particular, it does not provide adequate functionalities to incorporate and adapt to dynamically changing contextual information. Therefore, there is a need for an access control framework with dynamic associations of both user-role and role-permission capabilities.

In this thesis, we introduce a new context-aware access control (CAAC) framework for information resources and software services. CAAC extends the basic concepts of RBAC to use dynamic contextual information while making access control decision. The
4.2 The Conceptual Framework for CAAC

In this section, we present a formal model of our context-aware access control (CAAC) framework. Figure 4.1 shows the CAAC model and the relationships between its elements.

Our Context-Aware Access Control (CAAC) enables dynamic privileges assignment at two steps, letting users to access resources and services when a set of contextual conditions are satisfied. At the first step, the users are dynamically assigned to the roles when a set of conditions are satisfied. At the next step, when the roles are activated, then the permissions (service access permissions) are dynamically assigned to that roles

basic concepts of our CAAC framework are that users are dynamically assigned to roles by satisfying the relevant contextual conditions, permissions (service access permissions) are dynamically assigned to roles by satisfying the relevant conditions and users acquire appropriate permissions by being members of such active roles.

Figure 4.1: Core Context-Aware Access Control Model
Chapter 4. Conceptual Framework for Context-Aware Access Control

when a set of relevant contextual conditions are satisfied. Towards this goal, we introduce two main concepts context-aware user-role assignments and context-aware role-permission assignments.

Based on the formalization of the traditional Role-Based Access Control (RBAC) model [84], we present a formal definition of the CAAC model.

**Definition 4.1. (Core CAAC Model).** Our CAAC model \( M \) is a tuple (see Formula 4.1):

\[
M = (M_S, M_R)
\]

\[
\]

\[
M_R = (RH, CAURA, ResH, OpA, CARPA)
\]

The elements of this model are explained below. First of all, we define the following seven elements of our CAAC model:

- **Users (U):** \( U \) represents a set of users. The users are human-beings (who are service requesters) interacting with a computing system, whose access requests are being controlled.

  \[
  U = \{u_1, u_2, u_3, ..., u_m\}
  \]

- **Roles (R):** \( R \) represents a set of roles. The roles reflect user's job functions within the organizations (e.g., healthcare domain).

  \[
  R = \{r_1, r_2, r_3, ..., r_n\}
  \]

- **Resources (Res):** \( Res \) represents a set of resources. The resources are the objects protected by access control that represent the data/information container (e.g., the different parts of a patient’s medical records).

  \[
  Res = \{res_1, res_2, res_3, ..., res_o\}
  \]
4.2. The Conceptual Framework for CAAAC

- **Operations (Op):** $Op$ represents a set of operations on the resources. The operations are the actions that can be executed on the resources, for instance, read and write.

\[ Op = \{op_1, op_2, op_3, ..., op_p\} \]  \hspace{1cm} (4.5)

- **Permission (P):** $P$ represents a set of permissions. The permissions are the approvals to perform certain operations on resources, by the users who initiate access requests.

\[ P = \{p_1, p_2, p_3, ..., p_q\} = \{(res_i, op_j)|res_i \in Res, op_j \in Op\} \]  \hspace{1cm} (4.6)

Where $i = \{1, 2, 3, ..., o\}$, $j = \{1, 2, 3, ..., p\}$, $Res$ is a set of resources, and $Op$ is a set of operations on the resources.

- **Expressions (Exp):** $Exp$ represents a set of contextual expressions. The expressions are used to express the contextual conditions (using relevant context and situation information) in order to specify the user-role and role-permission assignment policies. The contextual expressions are specified in the context specification language (CSL).

\[ Exp = \{exp_1, exp_2, exp_3, ..., exp_r\} \]  \hspace{1cm} (4.7)

The detailed analysis of contextual expressions is given in the next section (see Section 4.3.3).

- **Policies (Pol):** $Pol$ represents a set of context-aware access control policies. CAAC has two sets of policies: the context-aware user-role assignment policies and context-aware role-permission assignment policies.

\[ Pol = \{Pol_{CAURA}, Pol_{CARPA}\} \]  \hspace{1cm} (4.8)

Originating from the above elements, the CAAC model has five other elements (using the relationships between the different sets of the above elements). These elements are defined formally as follows:
Chapter 4. Conceptual Framework for Context-Aware Access Control

- **Role Hierarchy (RH):** RH is a partial order on $R$ to serve as the role hierarchy, which supports the concept of role inheritance. The role is considered in a hierarchical manner in that if a permission assigned to a junior role, then it is also assigned to all the senior roles of that role.

$$RH \subseteq R \times R$$ \hspace{1cm} (4.9)

- **Context-Aware User-Role Assignment (CAURA):** CAURA is a context-aware user-role assignment relation, which is a many-to-many mapping between a set of users and roles, when a set of dynamic contextual conditions are satisfied.

$$CAURA \subseteq U \times R \times Exp$$ \hspace{1cm} (4.10)

- **Resource Hierarchy (ResH):** ResH is a partial order on $Res$ to serve as the resource hierarchy, which supports a user to access the different granularity levels of resources. The resource is considered in a hierarchical manner in that if a user has the right to access a resource with the highest granularity level, then he also has the right to access the lower granularity levels of that resource.

$$ResH \subseteq Res \times Res$$ \hspace{1cm} (4.11)

- **Operation Assignment (OpA):** OpA is a many-to-many operation-to-resource mapping. Each operation could be associated with many resources, and for each resource could be granted to many operations.

$$OpA \subseteq Res \times Op$$ \hspace{1cm} (4.12)

A set of operation assignment relations can be written as,

$$OpA = \{(res, op) | res \in Res, op \in Op\}$$ \hspace{1cm} (4.13)

- **Context-Aware Role-Permission Assignment (CARPA):** CARPA is a context-aware role-permission assignment relation, which is a many-to-many mapping between a set of roles and permissions, when a set of dynamic contextual conditions are satisfied.

$$CARPA \subseteq R \times P \times Exp$$ \hspace{1cm} (4.14)
4.3 Basic Elements of CAAC Framework

The main concepts that we have introduced in our context-aware access control (CAAC) model are the context-aware user-role assignments and context-aware role-permission assignments. Towards this end, we incorporate dynamically changing contextual conditions (what we have called as contextual expressions) into RBAC for dynamic user-role and role-permission assignments.

In the following Sections, we give some further formal description of context information, situation information, context specification language, role hierarchy, resource hierarchy and CAAC policy specification.

4.3 Basic Elements of CAAC Framework

4.3.1 Context Information

In the context-awareness literature, many researchers have defined the concept of context. According to Dey, context as any information that can be used to characterize the situation of an entity (person, place or object) [34].

For our purpose, however, existing context definitions do not specify the different types of entities specific to access control and the context information characterizing these entities. In this thesis, we define context as any relevant information about the state of an entity relevant to access control or the state of a relevant relationships between persons (as entities).

Context information has different levels of interpretation as it is structured as simple context and complex context, i.e., context information (C) is the set of all simple contexts ($C_s$) and all complex contexts ($C_c$).

\[
C = \{C_s, C_c\}
\]  \hspace{1cm} (4.15)

A simple context `$c_s$' ($c_s \in C_s$) is a context attribute of an entity that directly depends on a raw context fact. It depends on the value of a context attribute that characterizes the
Chapter 4. Conceptual Framework for Context-Aware Access Control

state of an entity, based on a single context information source.

\[ C_s = \{c_{s_1}, c_{s_2}, c_{s_3}, ..., c_{sr}\} \]  

(4.16)

Considering our application scenario (presented in Chapter 2), user identity is a simple context that represents a property of the user (resource requester).

A complex context ‘\(c_c\)’ (\(c_c \in C_c\)) is a combination of certain context information. It depends on the values of context attributes that characterize the state of one or more entities, based on the various context information sources.

\[ C_c = \{c_{c_1}, c_{c_2}, c_{c_3}, ..., c_{ct}\} \]  

(4.17)

Considering our application scenario, interpersonal relationship is a complex context that represents a property which is related to the user and resource owner.

The interpersonal relationship between user and owner can be inferred from available context information (e.g., user and owner profile information) through user-defined reasoning rules. Further details of the context information modelling, can be found in the ontology-based context model of the Chapter 5.

4.3.2 Situation Information

In the context-awareness literature, existing situation definitions typically describe the state of the specific types of context entity (e.g., the user’s state) [93, 99, 97]. However, these definitions are limited when considering access control in open and dynamic environments, where a user wants to access specific resources/services from a particular environment (e.g., a patient is in a critical health condition) for a certain purpose of resource access. In addition to the relevant states (the states of the entities or the states of the relevant relationships between entities), there exists a goal or purpose in every situation concerning the resource access. A security policy normally states that the particular software services can be invoked only for the specific purpose; and it describes the reason for which organizational resources are used [22]. For example, in the medical
4.3. Basic Elements of CAAC Framework

domain the American Health Information Management Association (AHIMA) identifies 18 health care scenarios across 11 purposes (treatment, payment, research, etc.) for health information exchange [36].

In this thesis, we define situation as a specific subset of the complete state of the universe of access control-specific context entities that are relevant to a certain goal or purpose of a resource access request.

Here, the universe is formed or defined by the context entities. A situation is a set of values whose types are defined by the domain-specific context predicates at a particular time. These values are determined by what the system needs to know given its current state in order to make the context-aware access control decision.

A purpose is the user’s intention to access the resources and services. It is a domain-dependent concept, and can be identified based on an access request (i.e., based on the relevant context values that are present in which a user is requesting access to the resources).

Considering our application scenario (presented in Chapter 2), the state of the relationship between user and owner is, interpersonalRelationship(Jane, Bob) = "nonTreatingPhysician" and the purpose of resource access is, purpose = "EmergencyTreatment".

Situation information has different levels of interpretation: atomic situation and composite situation, i.e., situation information (S) is the set of all atomic situations ($S_a$) and all composite situations ($S_c$).

$$S = \{S_a, S_c\}$$  \hspace{1cm} (4.18)

For any particular resource access request, an atomic situation $s_a$ ($s_a \in S_a$) depends on the values of relevant states and a purpose or user’s intention in accessing the resource.

$$S_a = \{s_{a1}, s_{a2}, s_{a3}, ..., s_{au}\}$$  \hspace{1cm} (4.19)
A composite situation \('s_c'\) \((s_c \in S_c)\) is a combination of one or more already defined situations. It depends on the values of one or more situations.

\[
S_c = \{s_{c1}, s_{c2}, s_{c3}, ..., s_c\}
\]

(4.20)

Further details of the situation information modelling, can be found in the ontology-based situation model of the Chapter 6.

4.3.3 Context Specification Language

Our proposed model includes a simple language (context specification language, CSL) for expressing simple context, complex context, atomic situation, composite situation and contextual expression. Based on this language, it is possible to specify these constraints in context-aware access control policies.

**Definition 4.2.** *(Simple Context \((C_s)\)).* Let \(E\) be the set of context entities, and \(C_s\) be the set of simple context information, then we define a simple context as a tuple in the form of

\[
< e.c_s, rel.op, v >, [\text{where}, e \in E, c_s \in C_s, and rel.op = \{<, \leq, >, \geq, =, \neq\}] \tag{4.21}
\]

In the above formula, \('e'\) denotes a context entity, \('c_s'\) denotes a simple context, \('e.c_s'\) denotes a simple context about an entity (i.e., context attribute of an entity), \('rel.op'\) denotes a relational operator (the set of \('rel.op'\) can be extended to accommodate user-defined operators (e.g., \('entering'\)), and \('v'\) denotes the value assigned to the context entity \('e'\) for the simple context \('c_s'\).

**Example 1.** *A patient or resource owner’s heart rate is less than 65 (or is abnormal), which is represented as,*

\[
c_{s1} = (Owner.heartRate < 65) \text{ or } c_{s1} = (Owner.heartRate = "Abnormal")
\]

(4.22)
A simple context is a simple constraint/expression in the CSL language. It is possible to construct more complex expressions logically combining (conjunction (\(\land\)), disjunction (\(\lor\)) and negation (\(\neg\))) constraints (simple and/or complex contexts).

**Definition 4.3.** (Complex Context \((C_c)\)). The complex contexts can be defined by performing logical composition on the simple or complex contexts.

\[
\begin{align*}
  c_{c1} &= c_1 \land c_2 \\
  c_{c2} &= c_3 \lor c_4 \\
  c_{c3} &= \neg c_5
\end{align*}
\]  

where \(c_1, c_2, c_3, c_4\) and \(c_5\) are already defined contexts \((c_i \in C_s\) or \(c_i \in C_c\)) and \(c_{c1}, c_{c2}\) and \(c_{c3}\) are new complex contexts \((c_{ci} \in C_c)\).

**Example 2.** Concerning our application scenario, consider an access control policy: A registered nurse (who is assigned for a patient) within a hospital is granted the right to read the patient’s daily medical records from the general ward where the patient is located, during her ward duty time. Using logical composition, this is represented as

\[
c_{c1} = ((\text{User.locationAddress} = \text{“GeneralWard”}) \land (\text{User.requestTime} = \text{“DutyTime”}) \land (\text{colocatedRelationship(User,Owner) = “Colocated”}))
\]  

**Definition 4.4.** (Atomic Situation \((S_a)\)). An atomic situation \(s_{a1}\) as a tuple in the form of

\[
s_{a1} = < p_1, st >
\]  

where, ‘\(p_1\)’ denotes a purpose, ‘\(st\)’ denotes the relevant states, and ‘\(s_{a1}\)’ denotes an atomic situation. The states can be formed based on the simple context or complex context.

\[
st = c_s \mid c_c
\]  

where, ‘\(c_s\)’ is a simple context and \(c_c\) is a complex context.
Example 3. Concerning our application scenario, consider an access control policy: A registered nurse (who is assigned for a patient) within a hospital is granted the right to read the patient’s daily medical records from the general ward where the patient is located, during her ward duty time for daily operational purpose. Using logical composition, this is represented as an atomic situation ($s_{a1} = p_1 \land c_{c1}$).

$$s_{a1} = ((\text{purpose} = \text{"DailyOperational"}) \land (\text{colocatedRelationship(User, Owner) = \text{"Colocated"}}) \land (\text{User.locationAddress} = \text{"GeneralWard"}) \land (\text{User.requestTime} = \text{"DutyTime"}))$$

(4.27)

Definition 4.5. (Composite Situation, $S_c$). The composite situations can be defined by performing logical composition on the other situations.

$$s_{c1} = s_1 \land s_2$$
$$s_{c2} = s_3 \lor s_4$$
$$s_{c3} = \neg s_5$$

(4.28)

where $s_1, s_2, s_3, s_4$ and $s_5$ are already defined atomic or composite situations ($s_i \in S_a$ or $s_i \in S_c$) and $s_{c1}, s_{c2}$ and $s_{c3}$ are new composite situations ($s_{ci} \in S_c$).

Definition 4.6. (Contextual Expression). By definitions 4.2, 4.3, 4.4 and 4.5, a contextual expression $\text{exp}$ ($\text{exp} \in \text{Exp}$) can be formed by using a set of contextual conditions. It can be a simple context, complex context, atomic situation or composite situation.

$$\text{exp} = c_s \mid c_c \mid s_a \mid s_c$$

(4.29)

Where,

- $c_s$ denotes a simple context ($c_s \in C_s$).
- $c_c$ denotes a complex context ($c_c \in C_c$).
- $s_a$ denotes an atomic situation ($s_a \in S_a$).
- $s_c$ denotes a composite situation ($s_c \in S_c$).
4.3. Basic Elements of CAAC Framework

Figure 4.2: Example of Role Hierarchy

4.3.4 Role Hierarchy

Like RBAC [84], our CAAC model introduces role hierarchies. We consider the role in a hierarchical manner, in order to structure roles to reflect an organization’s lines of authority and responsibility.

If a role ‘r’ is activated, then all the ancestors of ‘r’ in the hierarchy are activated. Activated roles are the basis for determining whether to allow or deny access permissions. An activated role is invalid or needed to be deactivated (called role revocation) when related context/situation state is changed. If a role ‘r’ is deactivated, then the descendant roles in the hierarchy are also deactivated. Figure 4.2 shows an example of a hierarchy of roles for our example application. Based on our application scenario, the RegisteredNurse role is activated when the nurse is present in the general ward and at duty time. Based on the role hierarchy (see Figure 4.2), we have partial ordering,

\[ NurseManager \preceq RegisteredNurse \]
\[ Nurse \preceq RegisteredNurse \]  

If a role RegisteredNurse is activated, then Nurse and NurseManager are activated as well. If RegisteredNurse role is deactivated (when the nurse leaves from the general ward or after ward shift time), then Nurse and NurseManager are deactivated as well.
Let $Exp$ be a set of contextual expressions, and $R$ be the set of roles, then we define a function named context-dependent role activation or revocation (CRAR),

$$CRAR : Exp \times R \rightarrow BOOL$$  \hspace{1cm} (4.31)

such that, given an expression $'exp'$ ($exp \in Exp$) and a role $'r'$ ($r \in R$), $CRAR(exp, r) = TRUE$, if the relevant contextual conditions are satisfied. Otherwise, $CRAR(exp, r) = FALSE$. Here, an expression $'exp'$ can be a simple context, complex context, atomic situation or composite situation.

In order to handle the role hierarchy and role inheritance, other than the is-a \textit{subclass-of} relations, we also consider the relations (\textit{interpersonal relationships}) between different roles, e.g., the relationships between doctors and nurses (the detail is presented in "Relationship Ontology" in Chapter 5, see Figure 5.12).

In modelling different types of health professionals (roles), we follow the Australian Standard Classification of Occupations (ASCO) of the Health Professionals [1] (the detail is presented in "Role Ontology" in Chapter 5, see Figure 5.10).

4.3.5 Resource Hierarchy

Like role hierarchy, we consider the resource (e.g., patients' electronic health records) in a hierarchical and service oriented manner, in order to provide fine-grained access control.
and grant the right access to the appropriate parts of a resource by the appropriate
users. In such a way, access to the resource and its components at different levels of
granularity can be managed. For example, there is the healthcare Resource hierarchy (the
different component parts of the patients' Electronic Health Records (EHR)) (Figure 4.3
shows a part of the EHR hierarchy). EHR has the following sub-components: a patient’s
Daily Medical Records (DMR), which includes Daily Observations Reports (DOR),
Physiological Records (PR), Physician Prescriptions (PP), etc.; a patient’s Past Medical
History (PMH); a patient’s Identification Records (IR and Demographics Records (DR);
and so on. The hierarchical Resource structure plays an important role in our model that
regulates access to different parts of a resource. In our application scenario, for example,
Mary can access Bob’s DMR, i.e., she can access all other sub-components of the DMR,
but not all the sub-components of the EHR.

Other than the part-of relations between resources, we also consider the granularity
levels of a patient’s medical records. Towards this end, we consider an important data
type property granularityLevel in our resource model that is used to regulate access to
different parts of a resource individually (the detail is presented in "Resource Ontology"
in Chapter 5, see Figure 5.11).

In formulating the structure of the patient electronic health records (EHR), we fol-
low the Health Level Seven (HL7) standard [2], the most implemented standard of
electronic data for healthcare.

A service is a well-defined and self-contained software entity with an invocable in-
terface to provide certain capability to perform certain operations on resources, by the
users who initiates access request. A service ‘ser’ (ser ∈ Ser) can be seen as a pair <res,
ap> with ‘res’ being a requested resource (res ∈ Res) and ‘ap’ being the action/operation
on the resource (ap ∈ Op).

\[
Ser \subseteq Res \times Op
\]

(4.32)

For example, the write operation on the emergency medical records is defined as <EMR,
write> or writeEMR(). In this way, the fine-grained access control to resources can be
realized by managing the access to the service operations.

In this example, a medical practitioner can access a patient’s EMR means, he/she can invoke read, write or both operations on the patient’s EMR records.

4.3.6 CAAC Policy Specification

In this section, we present the CAAC policy specification: integrating relevant context and situation information into the access control policies. CAAC provides two sets of policies, the context-aware user-role assignment and context-aware role-permission assignment policies.

Context-Aware User-Role Assignment (CAURA) Policy

Our CAAC model extends the concept of common user-role assignment in RBAC, by introducing the concept of context-aware user-role assignment (CAURA).

Traditional RBAC model defines user-role assignments (URA) simply as a mapping of users to roles.

\[ URA \subseteq U \times R \]  

(4.33)

We have extended this URA notion by introducing dynamic contextual expressions (integrating context and situation information).

**Definition 4.7.** (CAURA). Let U be the set of users, R be the set of roles and Exp be the set of contextual expressions, then CAURA is a many-to-many user-role assignment relation associated with certain contextual expressions.

\[ CAURA = \{(u_1, r_1, exp_1), (u_2, r_2, exp_2), \ldots, (u_i, r_j, exp_k)\} \subseteq U \times R \times Exp \]  

(4.34)

where ‘u’ denotes a user (\( u \in U \)), ‘r’ denotes a role (\( r \in R \)) and ‘exp’ denotes a contextual expression (\( exp \in Exp \)).

Context-aware user-role assignments can be expressed in tabular form (see Table
4.3. Basic Elements of CAAC Framework

4.1). The second row in Table 4.1 describes when Mary is present in the general ward, at ward shift time (duty time), then she can be assigned to the registered nurse role.

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
<th>Context Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>EmergencyDoctor</td>
<td>(User.locationAddress = “EmergencyRoom”)</td>
</tr>
<tr>
<td>Mary</td>
<td>RegisteredNurse</td>
<td>(User.locationAddress = “GeneralWard”) ∧ (User.requestTime = “DutyTime”)</td>
</tr>
</tbody>
</table>

**Definition 4.8.** (CAURA Policy). Let $U$ be the set of users, $R$ be the set of roles and $Exp$ be the set of contextual expressions. A CAURA policy ($Pol_{CAURA}$) is defined as follows:

\[ \forall u \forall r \forall exp (state(u) \land state(r) \land state(exp)) \]

\[(u, r, exp) \in CAURA\]  

(4.35)

where $state(u)$ denotes a user state, referring to a user $u$ ($u \in U$), e.g., $User.identity$ = “Mary00X”; $state(r)$ denotes a role state, referring to a role $r$ ($r \in R$), e.g., $Role.name$ = “RegisteredNurse”; $state(exp)$ denotes a context or situation state, referring to a contextual expression $exp$ ($exp \in Exp$), defined in the CSL.

For the sake of readability, a CAURA policy is also defined as in the format below:

\[pol_{CAURA} = (u, r, exp)\]  

(4.36)

**Example 4.** Concerning our application scenario, let us consider an access control policy: Mary can play the registered nurse role during her ward shift time and when she is located in the ward. Based on the CAURA policy definition (see Definition 4.8), the following rule (shown in Table 4.2) expresses the context-aware user-role assignment policy, i.e., a User $u$ ($u \in U$) can play a Role $r$ ($r \in R$) in contextual condition $exp$ ($exp \in Exp$).

In Table 4.2, Role ‘r’ denotes RegisteredNurse role, User ‘u’ is Mary, and Contextual Condition ‘exp’ is the combination of the following raw facts (the ward ‘duty time’ and ‘location’ of Mary).
Table 4.2: Context-Aware User-Role Assignment Policy

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAURAPolicy(caura) ∧ User(u) ∧ Role(r) ∧</td>
<td>canPlay(u, r)</td>
</tr>
<tr>
<td>hasUser(caura, u) ∧ hasRole(caura, r) ∧</td>
<td></td>
</tr>
<tr>
<td>plays(u, r) ∧ ContextualCondition(exp) ∧ hasCondition(caura, exp)</td>
<td></td>
</tr>
</tbody>
</table>

Context-Aware Role-Permission Assignment (CARPA) Policy

Similar to context-aware user-role assignment, our CAAC model extends the concept of role-permission assignment in RBAC with contextual expressions, called as context-aware role-permission assignment (CARPA).

Traditional RBAC model defines role-permission assignments (RPA) simply as a mapping of roles to permissions.

\[ RPA \subseteq R \times P \] (4.37)

We have extended this RPA notion by introducing dynamic contextual expressions (integrating context and situation information).

**Definition 4.9.** (CARPA). Let \( R \) be the set of roles, \( Exp \) be the set of contextual expressions, and \( P \) be the set of permissions, then CARPA is a many-to-many role-permission assignment relation associated with certain contextual expressions.

\[ CARPA = \{(r_1, p_1, exp_1), (r_2, p_2, exp_2), \ldots, (r_i, p_j, exp_k)\} \subseteq R \times P \times Exp \] (4.38)

where ‘\( p \)’ denotes a permission \( (p \in P) \), ‘\( r \)’ denotes a role \( (r \in R) \) and ‘\( exp \)’ denotes a contextual expression \( (exp \in Exp) \).

**Definition 4.10.** (CARPA Policy). Let \( R \) be the set of roles, \( P \) be the set of permissions, and \( Exp \) be the set of contextual expressions. A CARPA policy \( \text{Pol}_{\text{CARPA}} \) is defined as
4.3. Basic Elements of CAAC Framework

follows:

\[
\forall r \forall p \forall exp (\text{state}(r) \land \text{state}(p) \land \text{state}(exp))
\]

\[(r, p, exp) \in \text{CARPA}
\]

(4.39)

where \text{state}(r) denotes a role state, referring to a role ‘r’ \((r \in R)\); \text{state}(p) denotes a permission ‘p’ \((p \in P)\); \text{state}(exp) denotes a context or situation state, referring to a contextual expression ‘esp’ \((exp \in Exp)\), defined in the CSL language.

For the sake of readability, a CARPA policy is also defined as in the format below:

\[
pol_{\text{CARPA}} = (r, p, exp)
\]

(4.40)

Example 5. Concerning our application scenario, let us consider an access control policy:

A registered nurse within a hospital can read a patient’s daily medical records (DMR) for daily operational purpose when the patient’s health status is normal and she is present with the patient. Based on the CARPA policy definition, the following rule (shown in Table 4.3) expresses the context-aware role-permission assignment policy.

Table 4.3: Context-Aware Role-Permission Assignment Policy

| If | CARPA Policy(carpa) \land Role(r) \land Resource(res) \land hasRole(carpa,r) \land hasResource(carpa,res) \land Contextual Condition(exp) \land hasCondition(carpa,exp) \land Operation(op) \land hasOperation(carpa,op) |
| Then | canAccess(r,p) |

In Table 4.3, Role ‘r’ denotes RegisteredNurse role, permission is (DMR, Read) (i.e., Resource ‘res’ is DMR and Operation ‘op’ is Read), and Contextual Expression ‘exp’ is the combination of the following raw facts (‘normal health status’ of the patient, the patient and nurse are ‘colocated’, and the access request is for the ‘daily operational purpose’).
Further details of the policy specification can be found in the policy model of the Chapter 7.

4.4 Summary

In this Chapter, we have explained the need for an access control framework for modern dynamic and context-aware environments and presented a new context-aware access control framework, an extension of the basic RBAC framework [84], satisfying such a need. We have presented a formal model for specifying our access control framework including its basic elements. By introducing the concepts of context-aware user-role assignments and context-aware role-permission assignments, the context-aware associations of users to roles and roles to permissions can be achieved. For this purpose, we have incorporated the relevant context and situation information (which formed the contextual expressions) into the RBAC model. By following the proposed formal model concepts and based on our application scenario, we have discussed the context-aware access control policy specification: context-aware user-role and role-permission assignment policies. The first set of policies specifies that users can play a particular role when a set of contextual conditions are satisfied. The second set of policies specifies that users having roles are allowed to carry out an operation on the resources (service invocation) when a set of relevant conditions are satisfied.

In the following Chapters, we present an ontology-based context model (see Chapter 5), an ontology-based situation model (see Chapter 6) and a context-aware policy model (see Chapter 7) for CAAC, in order to capture/identify the dynamic attributes (context and situation information) at run time and to integrate this information into the access control processes.
In modern communication environments, the ability to provide access control to information resources or services in a context-aware manner is crucial. By leveraging the dynamically changing context information, we can achieve context-specific control over access to such resources and services, better satisfying the security and privacy requirements of the stakeholders. One of the key limitations of the existing access control frameworks is the lack of systematic capture and use of context information in making access control decisions. Towards this goal, Chapter 4 presented theoretical foundations for the construction of Context-Aware Access Control (CAAC) framework. For our CAAC framework, we in this chapter present an Ontology-Based Context Model, in modelling a wide range of dynamic context information (relevant low-level and high-level contexts). It includes a context ontology that is specific to access control and is capable of representing different types of context entities and context information relevant to access control. The context ontology also incorporates the ability to infer high-level implicit context information according to user-defined rules. To model this context information, the context ontology adopts the semantic technologies (OWL and SWRL).

The Chapter is organized as follows. Section 5.1 defines the context concepts (context entity and information) for context-aware access control. Section 5.2 discusses the design considerations of our context model using semantic technologies. In Section 5.3, we present an upper context ontology representing the general elements (context entities and context information) and the relationships between these elements, which are identified from the access control perspective. In Section 5.4, we present the domain-specific context ontologies representing the domain-specific elements and their relationships for the healthcare domain on the basis of the upper context ontology. In Section 5.5, we
present a reasoning model to infer high-level implicit context information that is not
directly available but can be derived from available information. Finally, Section 5.6
summarizes the chapter.

5.1 Defining Context for CAAC

In the context-awareness literature, many researchers have attempted to define the concept
of context. According to Dey, the general context entities are Person, Place and Object
[34]. He defines context as any information that can be used to characterize the situation
of an entity (person, place or object). Dey’s entity-centric concept of context provides
a general characterization of context entities in pervasive computing environments. For
our purpose, however, it does not specify the different types of entities specific to access
control and the context information characterizing these entities. We specialize Dey’s
definition of context to specifically cover access control applications. We define context
and context-awareness as follows:

**Definition 5.1. (Context Information)** Context Information used in an access control
decision is defined as any relevant information about the state of a relevant entity or the
state of a relevant relationship between relevant persons (as entities).

**Definition 5.2. (Context-Awareness)** Context-awareness relates to the use of context
information for context-aware access control decision making.

For example, the identity/role of the user and the interpersonal/colocated relationship
between user and owner are context information.

Focusing on the context entities relevant to making access control decisions, we
have classified the access control-specific context entities \( E \) into two groups: core entities
\((CE)\) and environmental entities \((EE)\).

\[
E = \{CE, EE\}
\]

(5.1)
5.1. Defining Context for CAAC

User, Resource, and resource Owner are the core access control entities as they are core concepts of access control. In order to adopt the advantages of RBAC, which regulates access to services based on user roles rather than individual users, our context model has Role as a core entity. The Relationship between Persons is another class of core entity, and has its own characterizing context information.

\[ CE = \{User, Role, Resource, Owner, Relationship\} \]  \hspace{1cm} (5.2)

The environmental entities are the other entities that are relevant to access requests. They include EnvPerson (environmental person), Place, and Device. In the example scenario (Scene #2) where a registered nurse Mary can access patient Bob’s past medical history if Jane (by playing a general practitioner role) is present in the same location as Mary, for instance, Mary is a User, Bob is an Owner, and Jane is an EnvPerson.

\[ EE = \{EnvPerson, Place, Device\} \]  \hspace{1cm} (5.3)

Focusing on the context information types that are relevant to making access control decisions, we have classified the context information into the following sub-categories.

- **Relationship Context Information** is any relevant information about relationships between person entities, e.g., the interpersonal relationship between Jane and Bob.

- **Status Context Information** is any relevant information about an entity at a particular time, e.g., the health status of patient Bob.

- **Profile Context Information** is any relevant information about an entity (such as User, Owner, Resource, or EnvPerson) e.g., the social profile of patient Bob.

- **Location Context Information** is any relevant information that is used to describe the spatial characteristics of an entity, e.g., the physical address of the Emergency Room.

- **Temporal Context Information** is any relevant information that is used to describe the temporal characteristics of an entity, e.g., the time when nurse Mary initiated the request.
Historical Context Information is any historical information of an entity that is relevant to access control decision making, e.g., the previous access history of user Jane.

The context information has different levels of interpretation as it is structured as low-level context and high-level context, i.e., Context Information (CI) is the set of all Low-level Context Information (LCI) and all High-level Context Information (HCI).

\[ CI = \{LCI, HCI\} \quad (5.4) \]

**Definition 5.3. (Low-level Context Information)** Low-level context is a context attribute of an entity (E) that directly depends on a raw context fact that characterizes the state of the entity.

For example, a low-level context user identity represents a property of the User, and therefore

\[ \{identity(u) \mid u \in User\} \subseteq LCI \quad (5.5) \]

In short, we represent a user’s identity as userIdentity.

In our model, we consider the low-level contexts, such as userIdentity, roleIdentity, resourceIdentity, granularityLevel, locationAddress, requestTime, heartRate, bodyTemperature, roomTemperature, humidity, weatherCondition, etc.

**Definition 5.4. (High-level Context Information)** High-level context (i.e., derived or inferred context) is a context attribute of an entity. It depends on the values of low-level contexts or other high-level contexts that characterize the state of these entities.

As an example, a high-level context interpersonal relationship represents a property which is related to the User and Owner. Therefore, we have

\[ \{interRelationship(u, o) \mid u \in User, o \in Owner\} \subseteq HCI \quad (5.6) \]

In short, we denote an interpersonal relationship between user and owner as interRelationship.
5.2. Design Considerations for the Context Model

The *interRelationship* between user and owner can be inferred from available context information (e.g., user and owner profile information). The further details are discussed in “Reasoning About Contexts” section (see Section 5.5).

In our model, we consider the high-level context information, such as *interRelationship, colocatedRelationship, socialProfile, healthStatus*, etc.

5.2 Design Considerations for the Context Model

In the previous section, we have already identified the context entities and context information relevant to access control. Based on these concepts, we in this section discuss the design considerations of our context model by using the semantic technologies for capturing and inferring relevant context information. Experiences from existing research works (e.g., [20, 80]) show that ontology-based approaches are very suitable for modelling context entities and information for pervasive computing applications. To meet the requirements of (1) representing contexts and (2) inferring high-level contexts, we propose an ontology-based context model, namely the Context Ontology for Access Control (COAC) (see Sections 5.3 and 5.5).

Our *Context Ontology for Access Control* represents *context entity* and *context information* as ontology elements. It is capable of representing general concepts, representing domain-specific concepts, and supporting reasoning according to user-defined rules (to obtain high-level context information).

- **General concepts** represent the general elements and the relationships between these elements. These elements and their relationships are not application specific but are specifically identified from the access control perspective. These general concepts are modelled in the *upper context ontology*.

- **Domain-specific concepts** represent the domain-specific elements and the relationships between these elements. These elements and their relationships are application
(domain) specific, and are modeled in the *domain-specific context ontologies* based on the upper ontology.

- **Reasoning rules** express the user-defined rules to derive high-level context information that is only implicitly present but can be inferred from available context information. As such, these rules extend the COAC model with the ability to reason about high-level context information.

Our COAC context model includes both the upper and domain-specific concepts for access control applications as described in the following parts. Domain-specific concepts are specializations of concepts from the upper ontology for specific domains. This separation between the upper context ontology (upper model) and the domain-specific context ontologies (domain models) encourages the reuse of general concepts, and provides a flexible basis for specifying domain-specific concepts. Our COAC upper ontology can be applicable in various application domains (e.g., healthcare, university, and office). Contexts in these specific domains share or have their roots in the general concepts in the upper ontology.

In general, three types of elements (concepts, properties, and property restrictions) form the foundation of ontologies:

1. **Classes and Subclasses**: The concepts include the *classes* or *subclasses* of the ontology which are the basic unit to represent context entities and context information.

2. **Object Type Properties**: The relationships between two classes are used to represent the *object type properties*.

3. **Data Type Properties**: The relationships between a class and its attribute values are used to represent the *data type properties*.

4. **Domain/Range Restrictions**: The *domain/range restrictions* appear between two class objects or between a class object and its attribute values.

5. **Cardinality Restrictions**: The *cardinality restrictions* appear between a class object and its attribute values to a specified number of times.
5.2. Design Considerations for the Context Model

To model context information, we adopt the OWL language [11], which is a family of knowledge representation languages and is endorsed by the World Wide Web Consortium (W3C). OWL has been the most practical choice for most ontological applications because of its considered trade-off between computational complexity of reasoning and expressiveness [80]. In particular, we use OWL-based ontological modelling in this research to represent the structure of context entities and context information.

In order to support the process of inferring high-level implicit context information from low-level context information (provided by the users, captured from the sensors, etc.), a set of reasoning rules need to be defined that associate the low-level contexts. OWL-based reasoning rules using Description Logic (DL) may not always be sufficient to infer the high-level implicit contexts [15]. For example, DL rules can not express the child of married parents (because it can not express the relationship between parents instances, i.e., the children whose parents are married), or the non-monotonicity concepts (e.g., birds fly, penguin is a bird, but penguin does not fly) [67]. Following our application scenario, we need to handle contexts using not only class definitions but also property path relationships between two classes or between a class and its attributes. On the other hand, SWRL rules [9] can express these concepts through the user-defined reasoning rules. In addition, some reasoning rules require mathematical computation (e.g., a is greater than or equal to b, a mod b), which is not supported by the OWL language. This is done through built-ins functions [8] in SWRL (SWRL Built-Ins for Comparisons, SWRL Math Built-Ins). The expressivity of OWL can be extended by adding SWRL rules to an ontology. We express the user-defined reasoning rules using the SWRL rule language, which provide the ability to infer additional information in our context ontology.

From the viewpoint of automated reasoning, the extension of OWL with SWRL is undecidable in the general case [74], because of some reasoning rules that will never terminate (reasoning in recursion). For example, the following SWRL rule can be used to capture whether two Users are co-located or not, given classes User and Location, and properties hasLocation, coLocated, and nonCoLocated.
Chapter 5. Context Model for CAAC

Notice that if this rule fires, some individuals of the class \textit{User} are classified to be \textit{coLocated}. Inference with such rule will never terminate, however, because there may exist other individuals of the class \textit{User} which do not satisfy the example rule above. That is, it cannot be determined how two \textit{Users} are classified as \textit{nonCoLocated}. However, we can ensure the decidability of the formalism, by adding alternative rules for every inference. This would be possible to classify every individual of the class \textit{User}. For example, the following SWRL rule can be used with the above example rule.

\[
\text{User(?x) \land User(?y) \land Location(?L1) \land Location(?L2) \land hasLocation(?x, ?L1) \land hasLocation(?y, ?L2) \land swrlb:notEqual(?L1, ?L2) \rightarrow nonCoLocated(?x, ?y)}
\]

In such a way, all individuals (pairs) of the class \textit{User} can be classified accordingly. In this work we follow this formalism, in order to specify user-defined rules using the SWRL rule language (see reasoning rules in Tables 5.2 and 5.3).

We have already discussed the basics of how to develop the COAC context model and what is included in our model, including the technologies that are used to develop the model. The further details of OWL can be found in the OWL specification tutorials [11].

### 5.3 Representing COAC Core Concepts

In the previous sections, we have already identified the context entities and context information relevant to access control, and discussed the design considerations to model these concepts. In this section, we introduce our \textit{COAC upper context ontology} for access control.

Figure 5.1 shows the top-level conceptual view of our \textit{COAC upper context ontology}. The upper ontology defines the following two main concepts: \textit{ContextEntity} and \textit{ContextInfo}. The \textit{ContextEntity} models the different types of context entities specific to access control and \textit{ContextInfo} models the context information types that are relevant to making access control decisions.
5.3. Representing COAC Core Concepts

![Figure 5.1: The COAC Upper Context Ontology](image)

5.3.1 Core Context Entities

The graphical representation of the context entities is shown in Figure 5.2. Following Dey's general context entities (Person, Place and Object) [34], we consider the following main concepts under the hierarchy of ContextEntity, namely the Person, Place, and Object. We consider Resource, Role, Relationship, and Device as subclasses of the class Object and User, Owner and EnvPerson as subclasses of the class Person.

![Figure 5.2: The COAC Upper Context Ontology: Context Entities](image)
The following code in OWL shows the class \textit{ContextEntity} has three main subclasses \textit{Person}, \textit{Place}, and \textit{Object} (see Definition 5.5).

\textbf{Definition 5.5.} ('ContextEntity' Class and its Three Main Subclasses).

\begin{verbatim}
<owl:Class rdf:ID="Person"/>
  <rdfs:subClassOf rdf:resource="#ContextEntity"/>
</owl:Class>

<owl:Class rdf:ID="Place"/>
  <rdfs:subClassOf rdf:resource="#ContextEntity"/>
</owl:Class>

<owl:Class rdf:ID="Object"/>
  <rdfs:subClassOf rdf:resource="#ContextEntity"/>
</owl:Class>
\end{verbatim}

Users can be grouped using the \textit{Role} concept/class. A \textit{hasRole} object property is used to relate the \textit{User} and \textit{Role} classes for representing the fact that a user has a role. \textit{User} is linked to \textit{Resource} by the \textit{hasReqRes} object property for capturing a user’s access interest in a resource. The relationship between a \textit{Resource} and its \textit{Owner} is captured by the object property \textit{isOwnedBy}. The following code in OWL (see Definition 5.6) shows the class \textit{Resource} has an object property \textit{isOwnedBy} with \textit{domain/range restrictions}, which is used to link the classes \textit{Resource} and \textit{Owner}.

\textbf{Definition 5.6.} ('isOwnedBy' Object Property Definition).

\begin{verbatim}
<owl:Class rdf:ID="Resource"/>
<owl:Class rdf:ID="Owner"/>
<owl:ObjectProperty rdf:ID="isOwnedBy">
  <rdfs:domain rdf:resource="#Resource"/>
  <rdfs:range rdf:resource="#Owner"/>
</owl:ObjectProperty>
\end{verbatim}
5.3. Representing COAC Core Concepts

An object property `hasRelationship` is used to link the `Person` and the `Relationship`. The following OWL code shows that the class `Person` has an object property `hasRelationship` (see Definition 5.7).

**Definition 5.7.** (`hasRelationship' Object Property Definition).

```
<owl:Class rdf:ID="Person"/>
<owl:Class rdf:ID="Relationship"/>
<owl:ObjectProperty rdf:ID="hasRelationship">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Relationship"/>
</owl:ObjectProperty>
```

The `EnvPerson` is a person who is neither a `User` nor an `Owner` but relevant in a certain situation (i.e., in such a case, these three classes `User`, `Owner`, and `EnvPerson` are disjoint with each other). The following code in OWL shows the definition of the class `User` as a subclass of the class `Person`, in which a class `EnvPerson` is specified as the disjoint class of the class `User` (see Definition 5.8).

**Definition 5.8.** (`Disjoint Class' Definition).

```
<owl:Class rdf:ID="User"/>
  <rdfs:subClassOf rdf:resource="Person"/>
  <owl:disjointClass>
    <owl:Class rdf:ID="EnvPerson"/>
  </owl:disjointClass>
</owl:Class>
```

The `Device` is the communication device that the `User` uses to issue the access request, and consequently the `User` is linked to `Device` through `hasDevice`. Furthermore, a `Person` is at a particular place and therefore is connected to `Place` with an object property `hasPlace`.

91
5.3.2 Core Context Information

The COAC upper ontology also has the following core concepts concerning the different types of dynamic context information (see Figure 5.3): the RelationshipInfo, StatusInfo, ProfileInfo, LocationInfo, TemporalInfo, and HistoricalInfo classes. These classes are organized into a hierarchy under ContextInfo.

The following OWL code shows that the class ContextInfo has six subclasses, namely RelationshipInfo, StatusInfo, ProfileInfo, LocationInfo, TemporalInfo, and HistoricalInfo (see Definition 5.9).

Definition 5.9. (‘ContextInfo’ Class and its Subclasses).

```
<owl:Class rdf:ID="RelationshipInfo"/>
<rdfs:subClassOf rdf:resource="ContextInfo"/>
</owl:Class>
<owl:Class rdf:ID="StatusInfo"/>
```
5.3. Representing COAC Core Concepts

The details of the subclasses and properties (i.e., object and data type properties) of these context information types are presented in the following subsections.

Relationship Information

In the application scenario, we have situations where a User's access permission is granted to the Resource only if some Relationship exists between the User and Owner or between the User and EnvPerson. We classify the relationship information into Person-Centric and Location-Centric (see Figure 5.4). This categorization is to express the existence of different types of relationships at a fine grained level. The Person-Centric relationship class contains a data type property (xsd:string type) named interRelationship, which indicates the interperson relationship between the Persons concerned such as "treating physician-patient" or "nurse-general practitioner".

The Location-Centric relationship class contains a data type property (xsd:boolean type) named isColocatedWith, which indicates that the Persons concerned are co-located, such as the nurse and the general practitioner in a ward. Some cardinality constraints can
also be defined for the concepts/classes. For example, a data type property \textit{isColocatedWith} which specifies the cardinality of the class \textit{Location-Centric}: each \textit{Location-Centric} relationship instance has exactly one \textit{isColocatedWith} attribute value, which means the value of the \textit{isColocatedWith} attribute may be "true" or "false". The cardinality restrictions on the properties may remove data duplication and can detect missing data. Figure 5.4 illustrates the representation of the \textit{RelationshipInfo} class, and its subclasses and data type properties.

![Diagram of RelationshipInfo class](image)

**Figure 5.4: Graphical Representation of the RelationshipInfo Class**

The following code in OWL (see Definition 5.10) shows that the class \textit{Location-Centric} has an attribute \textit{isColocatedWith} that is of boolean type (data type property with domain/range restriction).

**Definition 5.10. ('isColocatedWith' Data Type Property Definition).**

\[
<\text{owl:DatatypeProperty} \text{ rdf:ID="isColocatedWith"} > \\
<\text{rdfs:domain} \text{ rdf:resource="#Location-Centric"} > \\
<\text{rdfs:range} \text{ rdf:resource="&xsd:boolean"} > \\
</\text{owl:DatatypeProperty}>
\]

The following OWL code specifies the \textit{cardinality restriction} of the class \textit{Location-Centric} on the property \textit{isColocatedWith} (see Definition 5.11). The value of \textit{isColocatedWith} may be "true" or "false".
5.3. Representing COAC Core Concepts

![Graphical Representation of the StatusInfo Class](image)

Figure 5.5: Graphical Representation of the StatusInfo Class

**Definition 5.11.** (Cardinality Constraint Definition of a Property ‘isColocatedWith’).

```xml
<owl:Class rdf:ID="Location-Centric">
  <owl:Restriction>
    <owl:onProperty rdf:resource="#isColocatedWith"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
</owl:Class>
```

**Status Information**

The current `personal status` (busy or not), and the current `location status` (comfortable or not) are also part of the context information concerning the `status` of some entities. To capture the current status, we introduce a class named `StatusInfo`, which has the following subclasses and their corresponding data type properties: `PersonalStatus` (the `currentStatus` property) and `LocationStatus` (the `locationStatus` property). Figure 5.5 illustrates the representation of the `StatusInfo` class, and its subclasses and data type properties.

The following code (see Definition 5.12) in OWL shows that the `PersonalStatus` class has a data type property, named `currentStatus` (xsd:string type).
Chapter 5. Context Model for CAAC

ProfileInfo

Personal Profile

Resource Profile

userIdentity

roleIdentity

granularityLevel

resourcIdentity

xsd:string

xsd:int

Profile Info

Figure 5.6: Graphical Representation of the ProfileInfo Class

Definition 5.12. (‘currentStatus’ Data Type Property Definition).

\[ \text{<owl:DatatypeProperty rdf:ID=\"currentStatus\">} \]
\[ \text{<rdfs:domain rdf:resource=\"#PersonalStatus\">} \]
\[ \text{<rdfs:range rdf:resource=\"&xsd;string\">} \]
\[ \text{</owl:DatatypeProperty>} \]

Profile Information

In some circumstances, the profile information concerning entities is also relevant in making context-aware access control decisions. To represent the profile information we define a class ProfileInfo and its two subclasses PersonalProfile and ResourceProfile. We also define some of the data type properties: userIdentity and roleIdentity are the attributes of the class PersonalProfile, and granularityLevel and resourcIdentity are the attributes of the class ResourceProfile. For example, each Resource entity has ResourceProfile information (for assigning the resourcIdentity and granularityLevel attribute values). This relationship is captured through the "ContextEntity has ContextInfo" relationship. Figure 5.6 illustrates the representation of the ProfileInfo class, and its subclasses and data type properties.
5.3. Representing COAC Core Concepts

Location Information

To capture the different attributes (e.g., physical address, weather condition, and temperature) of a place, we introduce a class *LocationInfo*, which has two subclasses *IndoorLocation* and *OutdoorLocation*. We define the data type properties *locationAddress*, *luminosity*, *temperature*, and *noiseLevel* for the class *IndoorLocation*, and *locationAddress*, *humidity*, *temperature* and *weatherCondition* for the class *OutdoorLocation*. Figure 5.7 illustrates the representation of the *LocationInfo* class, and its subclasses and data type properties.

Figure 5.7: Graphical Representation of the LocationInfo Class

Figure 5.8: Graphical Representation of the TemporalInfo Class
Temporal Information

In order to capture the different temporal attributes we define a class TemporalInfo. To capture the user's duty time and request time, we define two classes SessionTime and RequestTime, which are the subclasses of TemporalInfo. We also define two subclasses and data type properties of the class SessionTime: SessionStartTime (startTime property) and SessionEndTime (endTime property) which are used to capture the session start time and end time. The requestTime property is an attribute of the class RequestTime. Let us return to our application scenario, where Mary can only read the records during specified session time (e.g., from 08:00:00 to 18:00:00). Consider another policy where a guest researcher can only access a patient's past medical history between 2013-07-16T09:00:00 and 2013-07-20T17:00:00. These two policies are associated with different representations of temporal constraints. Figure 5.8 illustrates the representation of the TemporalInfo class, and its subclasses and data type properties.

Historical Information

In some cases, the historical context is also relevant in making context-aware access control decisions. To this end, we consider a class named HistoricalInfo which captures the historical context information. It has the following subclasses and properties in order to capture the different types of previous access history. The UserRelated class is used to capture 'who was requesting access' using the userID attribute; the ResourceRelated class captures 'what type of resource was requested' using the resourceID attribute; and the

Figure 5.9: Graphical Representation of the HistoricalInfo Class
5.4. Representing COAC Domain-Specific Concepts

*InteractionRelated* class is used to capture ‘what was their interaction environment’ using the *accessDecision* ("granted’ or "denied"), and *dateTime* ("how frequently accessed") attributes. Figure 5.9 illustrates the representation of the *HistoricalInfo* class, and its subclasses and data type properties.

The domain-specific subclasses and properties of the context entities and context information types are presented in the ‘Representing COAC Domain-Specific Concepts’ section (Section 5.4).

5.4 Representing COAC Domain-Specific Concepts

For the application scenario, we define the *COAC domain-specific ontologies* for the healthcare domain on the basis of the *COAC upper context ontology* (which is presented in the previous section (Section 5.3)). Including domain-specific concepts in the ontology is important for the application developers.

5.4.1 Domain-Specific Context Entities

In this section, we focus on the representation of the *Role*, *Resource*, *Relationship* and *Place* ontologies for the healthcare domain. These COAC domain-specific ontologies are the specialization of concepts from the COAC upper ontology.

**Role Ontology**

The Role can be categorized as *InDomainUser* or *OutDomainUser*, and Figure 5.10 shows a part of the *Role* ontology. This categorization is to facilitate fine-grained control for different types (roles) of users. This is also beneficial from the representation viewpoint, as adding these two new general role classes will improve the structure of the roles and reduce the number of roles at one hierarchy level by distributing them into lower, more specific hierarchy levels. Additionally, this structure is beneficial from the reasoning viewpoint, as some resources/services are only relevant to these introduced role subclasses, i.e., *InDomainUser* or *OutDomainUser*, in the example. With this specialization, the
resources can be easily carried out in a fine-grained manner.

In the healthcare application, we model InDomainUser roles (see Figure 5.10), relying on the Australian Standard Classification of Occupations (ASCO) of the Health Professionals [1]. NursingProfessional, MedicalPractitioner, and OtherProfessional are subclasses of InDomainUser. NurseManager and RegisteredNurse are, in turn, subclasses of NursingProfessional; GeneralPractitioner, SpecialistPractitioner, TrainingPractitioner and EmergencyDoctor are subclasses of MedicalPractitioner; Pharmacist and Dietitian are subclasses of OtherProfessional, etc. Besides, some other users such as GuestResearcher,
5.4. Representing COAC Domain-Specific Concepts

HealthTrainer, etc., are not healthcare members but may need to access some of the patient information, and therefore are classified under OutDomainUser. This superclass-subclass hierarchy can facilitate access control in a way similar to the RBAC’s senior-junior role concept [84]. For example, a user playing the role MedicalPractitioner can access a patient’s daily medical records, which means a user playing the role GeneralPractitioner, SpecialistPractitioner, TrainingPractitioner or EmergencyDoctor (which are the subclasses of the class MedicalPractitioner), is also permitted to access the patient’s daily medical records. However, the converse is not true. For example, a user playing the role GeneralPractitioner can perform the test order task of a patient, but not all the MedicalPractitioner users (specifically, not a TrainingPractitioner user).

Resource Ontology

The different components at various granularity levels of a patient’s medical records are individually identifiable, so as to achieve fine-grained control over access to them. As such, there is the health record Resource hierarchy in the domain ontology (Figure 5.11 shows a part of this Resource ontology). The different component parts of the Resource
are linked with the Owner. The object property isOwnedBy is used to capture the relationship between a Resource component and its Owner. For the sake of simplicity, the resourceId and isOwnedBy attribute values are not shown in the Resource ontology (Figure 5.11).

In formulating the structure of the patient medical records, we follow the Health Level Seven (HL7) standard [2], the most implemented standard of electronic data for healthcare. We outline below some of its component parts and their corresponding domain-specific representation in the ontology model. A patient’s complete medical records would be obtained by joining all the component parts of the medical records, which is identified by the Electronic Health Records class (EHR). EHR is a subclass of the Resource class. In the Resource ontology, the corresponding component parts are represented by the hasPart object property in a tree structure (see Figure 5.11). The categorization of the Emergency Medical Records (EMR, at granularityLevel 0) class is important, because in emergency situations (e.g., a patient is in a critical health condition as in our application scenario), all general practitioners can access all the emergency component parts of a patient’s medical records. EMR has the following sub-components: PMH, DMR, PR, PP, and so on. Past Medical History (PMH, at granularityLevel 1) includes a patient’s past surgical history, obstetric history, previous medications, habits, immunization history, growth chart, development history, etc., i.e., what has happened to the patient since birth. A patient’s Daily Medical Records (DMR, at granularityLevel 1), includes Physiological Records (PR), Physician Prescriptions (PP) and Daily Observations Reports (DOR) at granularityLevel 2. PR includes a patient’s medical information such as blood pressure, body temperature, etc., possibly gathered from the electronic medical devices; PP contains a patient’s current medications; DOR includes the daily updates by all related members of the healthcare into the medical records documenting clinical changes, new information, etc. Private Medical Records (PMR, at granularityLevel 1) includes a patient’s personal health information, Identification Records (IR, at granularityLevel 1) includes a patient’s identity information, and Demographics Records (DR, at granularityLevel 1) includes patient information such as name, address, sex, age, contact number, to represent a patient’s profile.
5.4. Representing COAC Domain-Specific Concepts

Table 5.1: Resource Identities of Classes in the Resource Ontology

<table>
<thead>
<tr>
<th>resourceIdentity</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Emergency Medical records (EMR)</td>
</tr>
<tr>
<td>2</td>
<td>Past Medical History (PMH)</td>
</tr>
<tr>
<td>3</td>
<td>Daily Medical Records (DMR)</td>
</tr>
<tr>
<td>4</td>
<td>Private Medical Records (PMR)</td>
</tr>
<tr>
<td>5</td>
<td>Identification Records (IR)</td>
</tr>
<tr>
<td>6</td>
<td>Demographics Records (DR)</td>
</tr>
<tr>
<td>7</td>
<td>Physiological Records (PR)</td>
</tr>
<tr>
<td>8</td>
<td>Physician Prescriptions (PP)</td>
</tr>
<tr>
<td>9</td>
<td>Daily Observations Reports (DOR)</td>
</tr>
</tbody>
</table>

We set the resourceIdentity value for all the nodes (classes) in the Resource ontology, which is a xsd:int data type property that specifies the privacy level settings of different component parts of the Resource (patient’s medical record). For example, the component parts Emergency Medical Records (EMR), Daily Medical Records (DMR), and Private Medical Records (PMR) are set with the resourceIdentity values 1, 3, and 4 respectively.

Table 5.1 depicts the resourceIdentity values for some of medical record components. The granularityLevel is another important data type property (xsd:int type) in our model that is used to regulate access to different parts of a resource individually. For example, in the application scenario, Jane can access Bob’s EMR, at granularityLevel 0 (highest level), i.e., including all other sub-components of the EMR (at the lower granularity levels), while Mary can only access Bob’s DMR (at granularityLevel 1), i.e., the sub-components of the DMR (at granularityLevel 2), but not all the sub-components of the EMR (at granularityLevel 0).

Relationship Ontology

The different relationships at various granularity levels of a user’s service access request are individually identifiable, so as to achieve fine-grained relationship-specific control over access to information resources or software services.
Towards this end, we consider a relationship ontology with different granularity levels of relationships. Figure 5.12 shows an excerpt of the representation of the Relationship ontology for the healthcare domain. Our ontology only captures the interpersonal relationships between user and owner (User-Owner class). Similar to resource ontology, the Relationship ontology contains the granularityLevel property, which indicates the relationship granularity levels. For instance, a medical practitioner can access a patient’s some electronic health records (e.g., daily medical records (DMR)) with the doctor-patient relationship, at granularityLevel 0. However, she can access the private medical records (PMR), if she is a treating doctor of the patient (treatingPhysician relationship, at granularityLevel 1).

**Place Ontology**

In an emergency situation (see our application scenario), all general practitioners should be granted the right to access the patient’s emergency medical records from the emergency room of the hospital. In the Role ontology, we have classified the InDomainUser and OutDomainUser role classes for representing all hospital in-domain users or out-domain
users who may have the need to access the patients’ medical records. The concept of Place (e.g., inside of the hospital) can be modelled in a similar way. We classify Place as IndoorPlace and OutdoorPlace (see Figure 5.13). Similar to the Role categorization, this classification of locations is to facilitate fine-grained access control for users from different locations. Additionally, this classification is beneficial from the reasoning viewpoint, for example, the emergency medical records only can be accessible from the inside of the hospital (e.g., the emergency room) in the application scenario.

In the Place ontology for the healthcare domain, we can identify the different buildings, departments and rooms of the hospital. Figure 5.13 presents an excerpt of the Place ontology for the healthcare domain.

5.4.2 Domain-Specific Context Information

Concerning our healthcare application (presented in Chapter 2), we present below some of the relevant domain-specific subclasses and properties (i.e., object and data type properties) of the context information types.
Status Information

In addition to different types of general status information (personal and location status), the current health status (critical or normal) is also a part of the context information concerning the status of some domain-specific entities (e.g., patient’s health status). As such, we introduce a class named HealthStatus (including the healthStatus property), which is a subclass of the core StatusInfo class. Figure 5.14 illustrates the representation of the full StatusInfo class, and its subclasses and data type properties (core and domain-specific).

The following code (see Definition 5.13) in OWL shows that the HealthStatus class has a data type property, named healthStatus (xsd:string type).

**Definition 5.13.** (‘healthStatus’ Data Type Property Definition).

```xml
<owl:DatatypeProperty rdf:ID="healthStatus">
  <rdfs:domain rdf:resource="#HealthStatus"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

Profile Information

The profile information concerning domain-specific entities is also relevant in making context-aware access control decisions. We introduce two different types of profile information named HealthProfile and SocialProfile, which are the subclasses of the core
5.5. Reasoning About Contexts

ProfileInfo class. We also define some of the data type properties: bodyTemperature and heartRate are the attributes of the class HealthProfile, and connectedPeopleIdentity and connectedPeopleRoleIdentity are the attributes of the class SocialProfile. Figure 5.15 illustrates the representation of the full ProfileInfo class, and its subclasses and data type properties (core and domain-specific).

The further details of the ontology concepts for the purpose of user-defined reasoning are discussed in the "Reasoning About Contexts" Section (see Section 5.5).

5.5 Reasoning About Contexts

Our COAC ontology (defined in OWL) is extended with user-defined reasoning rules (specified in the SWRL rule language) to infer high-level implicit context information, and to improve control over access to specific services through automated inference of implicit context information from the basic context information (captured from the sensors, user profile, resource profile, etc.). As discussed in the last section, the basic context information can be explicitly captured by the COAC context ontology. This captured context information is used to infer new context information by using user-defined reasoning rules.
A user-defined reasoning rule has the form:

$$C_{i1} \land C_{i2} \land \ldots \land C_{in} \rightarrow C_{o1} \land C_{o2} \land \ldots \land C_{or}$$

(5.7)

where $C_{i1}, C_{i2}, \ldots C_{in}$ are the input concepts (body of the rule) and $C_{o1}, C_{o2}, \ldots C_{or}$ are the resultant/derived concepts (head of the rule). A derived concept, which is inferred from a reasoning rule, can also act as an input concept in another reasoning rule. Informally, a user-defined reasoning rule, which is an implication from body to head, can be read as follows: \textbf{if} all the conjunctive concepts in the body are true, \textbf{then} the head must also be true. A concept is either a class/object or an object/data type property, defined in the COAC ontology; or a SWRL built-in [8] defined in the SWRL ontology [9]. Concepts in any reasoning rule can be of the form $C(?x)$, $P(?x,?y)$ or \texttt{swrlb:builtin}, where $C$ is a COAC description (classes), $P$ is a COAC property (object/data type properties), and $x,y$ are either variables (variables are identified using the standard convention of prefixing them with a question mark), class individuals (e.g., a Role instance is "GeneralPractitioner_1" (see Rule #1(a) in Table 5.2)) or data values (e.g., heartRate is "abnormal" (see Rule #2(a) in Table 5.2)).

In essence, our reasoning technique using user-defined rules as the extension to the COAC context ontology can be seen as follows:

- \textbf{Reasoning rules that are fully represented in the SWRL rule language:}
  
  We use the SWRL rule language and its built-in functions to infer the high-level context information. For example, Rule #2(a) in Table 5.2 is used to determine \textit{the current health status of a patient} used in a context-aware access control decision making process.

5.5.1 Example Rules

Rule (1) In our COAC context ontology, in order to derive \textit{interRelationship} between user and owner, we specify a set of user-defined reasoning rules (see Rules 1(a) to 1(c) in Table 5.2). For example, Rule #1(a) states that \textbf{if} a User (playing
5.5. Reasoning About Contexts

The `GeneralPractitioner` role has requested access to a `Resource` which is owned by an `Owner` (a Patient) and the `Owner` is not related with the `GeneralPractitioner` through an `interRelationship`, then the association between the `User` and `Owner` is a "nonTreatingPhysician" relationship. Note that this rule has used the required (basic) information, the user’s `personal profile` information and the patient’s `social profile` information. This basic information can be obtained from the data sources. For the application scenario, based on the context ontology, this rule can determine that Jane and Bob has a "nonTreatingPhysician" relationship, because Jane is not assigned as the treating physician of patient Bob. Similarly, Rule 1(b) in Table 5.2 identifies the `interRelationship` between doctor and patient as "treatedPhysician", and Rule 1(c) in Table 5.2 identifies the `interRelationship` between nurse and patient as "assignedNurse".

Rule (2) Another set of SWRL reasoning rules are also specified in Table 5.2, to derive the `healthStatus` of a patient (who is a resource owner) (Rules from 2(a) to 2(c)). For example, Rule #2(a) states that if the patient’s `bodyTemperature` is "normal" and his `heartRate` is "abnormal", then his `healthStatus` is "critical". Similarly, Rule #2(b) states that the patient’s `healthStatus` is "normal" when his `bodyTemperature` is "normal" and `heartRate` is "normal", and Rule #2(c) states that the patient’s `healthStatus` is "critical" when his `bodyTemperature` is "abnormal" and `heartRate` is "normal".

Rule (3) To derive the information about whether two persons are located in the same place or not, we specify a set of reasoning rules (see Rules #3(a) and 3(b) in Table 5.3). For example, Rule #3(a) states that if the user and the environmental person are located in the same place, then they are co-located. For the example scenario, based on the context ontology, this rule can determine that Mary and Jane are co-located. Similarly, Rule 3(b) in Table 5.3 identifies the `location-centric relationship`, when two persons are not located in the same place.
Table 5.2: User-Defined Reasoning Rules

<table>
<thead>
<tr>
<th>No</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rule (C_{i_1} \land C_{i_2} \land ... \land C_{i_n} \rightarrow C_{o_1} \land C_{o_2} \land ... \land C_{o_m}, where each C is a rule concept (i.e., class/object, object property, data property or SWRL built-ins)).</td>
</tr>
<tr>
<td>Rule #1(a)</td>
<td>User(?u) \land Role(?rol) \land hasRole(?u, ?rol) \land swrlb:equal(?rol, &quot;GeneralPractitioner&quot;) \land Resource(?r) \land hasReqRes(?u, ?r) \land Owner(?o) \land isOwnedBy(?r, ?o) \land Relationship(?re) \land hasRelation(?u, ?re) \land hasRelationship(?o, ?re) \land Person-Centric(?rel) \land has(?rel, ?re) \land PersonalProfile(?pp) \land has(?u, ?pp) \land userID(?pp, ?uID) \land roleIdentity(?pp, ?rolID) \land SocialProfile(?sp) \land has(?o, ?sp) \land connectedPeopleIdentity(?sp, ?cpID) \land connectedPeopleRoleIdentity(?sp, ?cpRID) \land swrlb:notEqual(?cpID, ?uID) \land swrlb:notEqual(?cpRID, ?rolID) \rightarrow interRelationship(?re, &quot;nonTretainingPhysician&quot;)</td>
</tr>
<tr>
<td>Rule #1(b)</td>
<td>User(?u) \land Role(?rol) \land hasRole(?u, ?rol) \land swrlb:equal(?rol, &quot;GeneralPractitioner&quot;) \land Resource(?r) \land hasReqRes(?u, ?r) \land Owner(?o) \land isOwnedBy(?r, ?o) \land Relationship(?re) \land hasRelation(?u, ?re) \land hasRelationship(?o, ?re) \land Person-Centric(?rel) \land has(?rel, ?re) \land PersonalProfile(?pp) \land has(?u, ?pp) \land userID(?pp, ?uID) \land roleIdentity(?pp, ?rolID) \land SocialProfile(?sp) \land has(?o, ?sp) \land connectedPeopleIdentity(?sp, ?cpID) \land connectedPeopleRoleIdentity(?sp, ?cpRID) \land swrlb:equal(?cpID, ?uID) \land swrlb:equal(?cpRID, ?rolID) \rightarrow interRelationship(?rel, &quot;treatingPhysician&quot;)</td>
</tr>
<tr>
<td>Rule #1(c)</td>
<td>User(?u) \land Role(?rol) \land hasRole(?u, ?rol) \land swrlb:equal(?rol, &quot;RegisteredNurse&quot;) \land Resource(?r) \land hasReqRes(?u, ?r) \land Owner(?o) \land isOwnedBy(?r, ?o) \land Relationship(?re) \land hasRelation(?u, ?re) \land hasRelationship(?o, ?re) \land Person-Centric(?rel) \land has(?rel, ?re) \land PersonalProfile(?pp) \land has(?u, ?pp) \land userID(?pp, ?uID) \land roleIdentity(?pp, ?rolID) \land SocialProfile(?sp) \land has(?o, ?sp) \land connectedPeopleIdentity(?sp, ?cpID) \land connectedPeopleRoleIdentity(?sp, ?cpRID) \land swrlb:equal(?cpID, ?uID) \land swrlb:equal(?cpRID, ?rolID) \rightarrow interRelationship(?rel, &quot;assignedNurse&quot;)</td>
</tr>
<tr>
<td>Rule #1(d)</td>
<td>...</td>
</tr>
<tr>
<td>Rule #2(a)</td>
<td>Owner(?o) \land HealthStatus(?hs) \land has(?o, ?hs) \land HealthProfile(?hp) \land has(?o, ?hp) \land bodyTemperature(?hp, &quot;normal&quot;) \land heartRate(?hp, &quot;normal&quot;) \rightarrow healthStatus(?hs, &quot;normal&quot;)</td>
</tr>
<tr>
<td>Rule #2(b)</td>
<td>Owner(?o) \land HealthStatus(?hs) \land has(?o, ?hs) \land HealthProfile(?hp) \land has(?o, ?hp) \land bodyTemperature(?hp, &quot;normal&quot;) \land heartRate(?hp, &quot;normal&quot;) \rightarrow healthStatus(?hs, &quot;normal&quot;)</td>
</tr>
<tr>
<td>Rule #2(c)</td>
<td>Owner(?o) \land HealthStatus(?hs) \land has(?o, ?hs) \land HealthProfile(?hp) \land has(?o, ?hp) \land bodyTemperature(?hp, &quot;abnormal&quot;) \land heartRate(?hp, &quot;abnormal&quot;) \rightarrow healthStatus(?hs, &quot;critical&quot;)</td>
</tr>
<tr>
<td>Rule #2(d)</td>
<td>...</td>
</tr>
</tbody>
</table>
Table 5.3: User-Defined Reasoning Rules (Continued)

<table>
<thead>
<tr>
<th>No</th>
<th>Rule (C_{i1} \land C_{i2} \land ... \land C_{in} \rightarrow C_{o1} \land C_{o2} \land ... \land C_{on}, where each C is a rule concept (i.e., class/object, object property, data property or SWRL built-ins)).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule #3(a)</td>
<td>User(?u) \land EnvPerson(?o) \land Place(?p1) \land hasPlace(?u, ?p1) \land Place(?p2) \land hasPlace(?o, ?p2) \land IndoorLocation(?il1) \land has(?p1, ?il1) \land locationAddress(?il1, ?add1) \land IndoorLocation(?il2) \land has(?p2, ?il2) \land address(?il2, ?add2) \land swrlb:equal(?add1, ?add2) \land Relationship(?rel) \land hasRelationship(?u, ?rel) \land hasRelationship(?o, ?rel) \land Location-Centric(?col) \land has(?rel, ?col) \rightarrow isColocatedWith(?col, true)</td>
</tr>
<tr>
<td>Rule #3(b)</td>
<td>User(?u) \land EnvPerson(?o) \land Place(?p1) \land hasPlace(?u, ?p1) \land Place(?p2) \land hasPlace(?o, ?p2) \land IndoorLocation(?il1) \land has(?p1, ?il1) \land locationAddress(?il1, ?add1) \land IndoorLocation(?il2) \land has(?p2, ?il2) \land address(?il2, ?add2) \land swrlb:notEqual(?add1, ?add2) \land Relationship(?rel) \land hasRelationship(?u, ?rel) \land hasRelationship(?o, ?rel) \land Location-Centric(?col) \land has(?rel, ?col) \rightarrow isColocatedWith(?col, false)</td>
</tr>
<tr>
<td>Rule #3(c)</td>
<td>...</td>
</tr>
<tr>
<td>Rule #4(a)</td>
<td>Place(?p) \land LocationStatus(?ls) \land has(?p, ?ls) \land IndoorLocation(?il) \land has(?p, ?il) \land luminosity(?il, ?lu) \land swrlb:equal(?lu, &quot;bright&quot;) \land noiseLevel(?il, ?n) \land swrlb:equal(?n, &quot;low&quot;) \land temperature(?il, ?t) \land swrlb:equal(?t, &quot;warm&quot;) \rightarrow locationStatus(?ls, &quot;excellent&quot;)</td>
</tr>
<tr>
<td>Rule #4(b)</td>
<td>Place(?p) \land LocationStatus(?ls) \land has(?p, ?ls) \land IndoorLocation(?il) \land has(?p, ?il) \land luminosity(?il, ?lu) \land swrlb:equal(?lu, &quot;normal&quot;) \land noiseLevel(?il, ?n) \land swrlb:equal(?n, &quot;low&quot;) \land temperature(?il, ?t) \land swrlb:equal(?t, &quot;warm&quot;) \rightarrow locationStatus(?ls, &quot;comfortable&quot;)</td>
</tr>
<tr>
<td>Rule #4(c)</td>
<td>Place(?p) \land LocationStatus(?ls) \land has(?p, ?ls) \land IndoorLocation(?il) \land has(?p, ?il) \land luminosity(?il, ?lu) \land swrlb:equal(?lu, &quot;normal&quot;) \land noiseLevel(?il, ?n) \land swrlb:equal(?n, &quot;high&quot;) \land temperature(?il, ?t) \land swrlb:equal(?t, &quot;hot&quot;) \rightarrow locationStatus(?ls, &quot;uncomfortable&quot;)</td>
</tr>
<tr>
<td>Rule #4(d)</td>
<td>...</td>
</tr>
<tr>
<td>Rule #5(a)</td>
<td>User(?u) \land RequestTime(?rt) \land has(?u, ?rt) \land requestTime(?rt, ?rt_ms) \land SessionStartTime(?sst) \land has(?u, ?sst) \land startTime(?sst, ?sst_ms) \land SessionEndTime(?set) \land has(?u, ?set) \land endTime(?set, ?set_ms) \land swrlb:greaterThanOrEqual(?rt_ms, ?sst_ms) \land swrlb:lessThan(?rt_ms, ?set_ms) \rightarrow requestTime(?rt, &quot;dutyTime&quot;)</td>
</tr>
<tr>
<td>Rule #5(b)</td>
<td>User(?u) \land RequestTime(?rt) \land has(?u, ?rt) \land requestTime(?rt, ?rt_ms) \land SessionStartTime(?sst) \land has(?u, ?sst) \land startTime(?sst, ?sst_ms) \land SessionEndTime(?set) \land has(?u, ?set) \land endTime(?set, ?set_ms) \land swrlb:lessThan(?rt_ms, ?sst_ms) \land swrlb:greaterThanOrEqual(?rt_ms, ?set_ms) \rightarrow requestTime(?rt, &quot;nonDutyTime&quot;)</td>
</tr>
<tr>
<td>Rule #6(a)</td>
<td>...</td>
</tr>
</tbody>
</table>
Rule (4) We further specify a set of rules in Table 5.3 (see Rules 4(a) to 4(c)), to derive the locationStatus of a place (from which user's access request is originated). For example, Rule #4(a) states that if the luminosity of a place is "bright", and its noiseLevel is "low" and temperature is "warm", then its locationStatus is "excellent". Similarly, Rules 4(b) and 4(c) identify the locationStatus as "comfortable" and "uncomfortable", respectively.

Rule (5) To determine the requestTime "duty time" or "non duty time", we specify a set of reasoning rules (see Rules 5(a) and 5(b) in Table 5.3). Rule #5(a) states that if the requestTime is greater than or equal to duty startTime, and less than duty endTime, then the requestTime is "dutyTime". Otherwise, the requestTime is "nonDutyTime". Here, the requestTime is the time when a user makes a request or the resource is requested, and the session time is the time during which a resource is accessible by the user. Based on the application scenario, let us assume that our temporal part of the context model dynamically captured Mary's request time (15:12:32), session start time (08:00:00) and end time (18:00:00). By using the reasoning Rule #5(a), our COAC context ontology can deduce Mary's requestTime being "dutyTime", because it is between her session start time and end time. Similarly, Rule 5(b) in Table 5.3 identifies the requestTime as "nonDutyTime".

5.6 Summary

In this Chapter, we have proposed an ontology-based context model for our CAAC framework, which is presented in Chapter 4. It includes a general and extensible context model specific to access control, and a reasoning technique for inferring high-level implicit contexts based on user defined rules. Our context model significantly differs from the existing context models in that it considers the relevant access control-specific entities (user, role, resource, owner, relationship, and environmental entities), represents and models a wide range of important dynamic context information characterizing these

\footnote{Note that the \texttt{getTime()} Java method can be used to return the number of milliseconds since 1970-01-01 00:00:00 UTC.}
5.6. Summary

dentities within a single framework, and allows context information representation and
derivation at different levels of abstraction by inferring high-level implicit context (i.e.,
more abstract and richer context information can be derived by using our model).

In the following Chapters, we present the situation model (see Chapter 6) and pol-

cy model (see Chapter 7). The situation model identifies the relevant purpose-oriented

situations using the relevant context information from our context model. Using the

context and situation models, the policy model of the framework provides support for

specifying and enforcing context-aware access control policies.
A security policy normally states that the particular services can be invoked based on the states of the relevant context entities and the specific purpose, which describes the reason for which information resources or software services are used. By identifying the user's intention in accessing software services, we can achieve purpose-oriented control over access to such services. In open and dynamic environments, therefore, situation-aware applications need to capture relevant context information and user intention or purpose, to provide situation-specific access to software services. Existing context/situation-aware access control approaches are highly domain-specific and they control access to services depending on the specific types of context information without considering the purpose or user's intention. Ontology-based approach has been evaluated as the most widely used modelling technique for context/situation modelling because of its considered trade-off between computational complexity of reasoning and expressiveness. To achieve situation-aware access control, in this chapter, we present an Ontology-Based Situation Model that adopts semantic technologies in modelling the relevant purpose-oriented situations (atomic and composite situations). The situation ontology model uses the relevant context information from our context model, which is proposed in the Chapter 5. We take situation to mean the states of the entities and the states of the relationships between entities that are relevant to the purpose of a resource access request.

The chapter is organized as follows. Section 6.1 defines the situation concepts for context-aware access control. In Section 6.2, we present an ontology-based situation model, including the core concepts representing the general elements and the relationships between these elements (see Section 6.2.1), and the identification of atomic situations on the basis of the available context information (see Section 6.2.2). In Section 6.2.3, we
present a reasoning model to infer the purpose and to reason about composite situations that are not directly available but can be derived from available information using ontology-based reasoning rules. Finally, Section 6.3 summarizes the chapter.

6.1 Defining Situation for CAAC

In order to define situations for situation-aware access control applications, on the one hand, it is required to capture the states of the relevant situation-specific context entities (e.g., user, resource, resource owner) and the states of the relevant relationships between different entities (e.g., the interpersonal relationships between the user and the resource owner). On the other hand, it is required to identify the purpose or user’s intention in accessing the information resources or software services. Considering our application scenario (which is presented in Chapter 2), an emergency doctor’s request to invoke a healthcare service (access to the emergency patient’s records when the patient is in a critical health condition) may be possible from the inside of the hospital but may not from the public bus. Also, such service access request can be granted for the emergency treatment purpose.

In our purpose-oriented situation model, a situation consists of the set of elementary information (the combination of the relevant states and the user’s intention or purpose-oriented information). We define the simple situation (atomic situation) and the complex situation (composite situation) that are used in specifying situation-specific access control policies.

6.1.1 Inferring Purpose

In our situation model, a purpose can be identified based on the relevant states: (i) the states of the relevant entity and (ii) the states of the relevant relationships between entities. The states are formed by using the currently available context information from the context model.

Definition 6.1. (Purpose, P). A Purpose is the user’s intention in accessing information
6.1. Defining Situation for CAAC

resources or software services. Like as situation, the purpose is also a domain-specific concept, and its value can be obtained based on the access request.

\[ P = \{p_1, p_2, p_3, \ldots, p_j\} \]  

(6.1)

where ‘\( P \)’ is the set of purposes and ‘\( p_1 \)’, ‘\( p_2 \)’, ‘\( p_3 \)’, ..., ‘\( p_j \)’ are individual purposes.

Consider our application scenario *Scene #1*, which is presented in Chapter 2, where the purpose is: \( p_1 = "\text{EmergencyTreatment}" \) (or ET).

**Example 6.** Consider Policy #1 related to our application scenario: a user, who is a general practitioner, by playing an emergency doctor (ED) role can access a patient’s emergency medical records (EMR) in the hospital for emergency treatment (ET) purpose, when the patient is in a critical condition. The following rule (6.2) is used to identify that the purpose is ‘ET’ (i.e., a user by playing the ‘ED’ role can access a patient’s records for ‘ET’ purpose, when the patient’s health condition is critical),

\[
\begin{align*}
\text{Purpose}(p_1) & \land \text{User}(u) \land \text{Role}(r) \land \text{Owner}(o) \land \text{Resource}(res) \\
& \land \text{hasRole}(u, r) \land \text{equal}(r, "ED") \land \text{isOwnedBy}(res, o) \\
& \land \text{healthStatus}(o, "critical") \rightarrow \text{equal}(p_1, "ET") .
\end{align*}
\]

(6.2)

This inferred purpose and the other relevant states based on the available context information (which is not used for the identification of purpose) are used to form a situation (see an identification of atomic situation in Example 7 and an identification of composite situation in Example 8).

6.1.2 Atomic Situation

Different atomic situations can be defined based on the domain-dependent data/information from the domain (e.g., healthcare domain) through the development of computer-supported models.
Definition 6.2. (Atomic Situation, $S_a$). A Situation used in an access control decision is defined as the states of the entities and the states of the relationships between entities at a particular time that are relevant to a certain goal or purpose of a resource access request. The situation is a domain-dependent concept, and its value can be obtained based on the access request (i.e., from the sensed contexts, derived contexts, inferred contexts, etc.).

\[ S_a = \{s_{a1}, s_{a2}, s_{a3}, \ldots, s_{ai}\} \]  

(6.3)

where \(S_a\) is the set of atomic situations and \(s_{a1}, s_{a2}, s_{a3}, \ldots, s_{ai}\) are individual atomic situations.

An atomic situation \(s_{ai}\) is the logical conjunction of a purpose \(p_j\) and a set of states \(St\).

\[ s_{ai} = p_j \land St \]  

(6.4)

where \(p_j\) \((p_j \in P)\) is a purpose (see Definition 6.1) and \(St\) denotes a set of relevant states (the states of the entities, the states of the relationships between entities).

Considering the same application scenario Scene #1, where the relevant states are: the state of an entity, e.g., location(Jane) = “Hospital”, the state of the relationship between entities, e.g., interpersonalRelationship(Jane, Bob) = “nonTreatingPhysician”.

Example 7. Consider the policy mentioned in Example 6, in which the relevant elementary information is represented as an atomic situation \((s_{a1} \in S_a)\) (see rule (6.5)),

\[
\begin{align*}
  s_{a1} &= User(u) \land Purpose(p_1) \land intendedPurpose(u, p_1) \land equal(p_1, “ET”) \\
  &\land Location(l) \land hasLocation(u, l) \land equal(l, “Hospital”).
\end{align*}
\]  

(6.5)

6.1.3 Composite Situation

The process of inferring a new composite situation (complex situation) from the one or more already defined atomic situations is referred to as reasoning about situation. One of the main advantages of our situation model is its reasoning capability; that is, once facts about the world are stated, other facts can be inferred using an inference engine through
6.1. Defining Situation for CAAC

user-defined reasoning rules.

**Definition 6.3.** (Composite Situation, \(S_c\)). Given a collection of atomic situations of the same purpose, a composite situation can be defined by performing logical composition (AND, OR or NOT) on one or more already defined atomic situations (i.e. each of the current atomic situations can be a composite situation or an atomic situation). For example,

\[
\begin{align*}
\text{s}_{c1} &= s_1 \land s_2 \\
\text{s}_{c2} &= s_3 \lor s_4 \\
\text{s}_{c3} &= \neg s_5
\end{align*}
\]  

(6.6)

where \(s_1, s_2, s_3, s_4,\) and \(s_5\) are already defined atomic or composite situations (\(s_i \in S_a\) or \(s_i \in S_c\)), and \(s_{c1}, s_{c2},\) and \(s_{c3}\) are new composite situations.

In general,

\[
S_c = \{s_{c1}, s_{c2}, s_{c3}, \ldots, s_{ck}\}
\]  

(6.7)

where \(S_c\) is the set of composite situations and \(s_{c1}, s_{c2}, s_{c3},\ldots, s_{ck}\) are composite situations.

**Example 8.** Consider Policy #2 related to our application scenario (which is presented in Chapter 2): a user, by playing a registered nurse (RN) role, is granted the right to read/write a patient’s daily medical records (DMR) during her ward duty time (DT) or from the general ward (GW) where the patient is located for daily operation (DO) purpose. Whereas, the registered nurse can access the patient’s private medical records (PMR) with the consent of patient during her ward duty time and from the general ward where the patient is located for daily operation purpose. The daily operation (DO) purpose can be identified using the following rule (6.8). The rule specifies that a user by playing the ‘RN’ role can access a patient’s medical records for ‘DO’ purpose, when the patient’s health condition is normal.

\[
\begin{align*}
\text{Purpose}(p_2) \land \text{User}(u) \land \text{Role}(r) \land \text{Owner}(o) \land \text{Resource}(res) \\
\land \text{hasRole}(u, r) \land \text{equal}(r, "RN") \land \text{isOwnedBy}(res, o) \\
\land \text{healthStatus}(o, "normal") \rightarrow \text{equal}(p_2, "DO")
\end{align*}
\]  

(6.8)
Two atomic situations (rules 6.9 and 6.10) regarding the mentioned policy are represented as follows,

\[ s_{a2} = User(u) \land Purpose(p_2) \land intendedPurpose(u, p_2) \land equal(p_2, "DO") \]
\[ \land Location(l) \land hasLocation(u, l) \land equal(l, "GW") \]
\[ \land Owner(o) \land isColocatedWith(u, o). \] (6.9)

\[ s_{a3} = User(u) \land Purpose(p_2) \land intendedPurpose(u, p_2) \land equal(p_2, "DO") \]
\[ \land Time(t) \land hasRequestTime(u, t) \land equal(t, "DT"). \] (6.10)

An example policy associated with the situation ‘\( s_{a2} \)’ (rule (6.9)) can be read as, a user by playing a registered nurse (RN) role, who is located with a patient in the general ward (GW) of the hospital, can access the patient’s daily medical records (DMR) for daily operation (DO) purpose, when the patient’s health condition is normal.

An example policy associated with the situation ‘\( s_{a3} \)’ (rule (6.10)) can be read as, a user by playing the ‘RN’ role can access the patient’s ‘DMR’ during her ward duty time (DT) for ‘DO’ purpose, when the patient’s health condition is normal.

A composite situation ‘\( s_{c1} \)’ (\( s_{c1} \in S_c \)) with these two atomic situations (\( s_{a2} \) and \( s_{a3} \)) can be identified using the following logical conjunction,

\[ s_{c1} = s_{a2} \land s_{a3}. \] (6.11)

An example policy associated with the situation ‘\( s_{c1} \)’ (rule (6.11)) can be read as, a user by playing the ‘RN’ role, who is co-located with a patient in the general ward (GW) of the hospital, can access the patient’s private medical records (PMR) during her ward ‘DT’ for ‘DO’ purpose, when the patient’s health condition is normal.

### 6.2 Ontology-Based Situation Model

The identification of the relevant information to represent the purpose-oriented situations is already discussed in the previous section. In this section, we introduce an ontology-based...
6.2. Ontology-Based Situation Model

A situation model by using the semantic technologies for representing and inferring relevant situations. The main goals of our situation ontology are to specify the concept of purpose-oriented situation using an ontology-based modeling language (e.g., OWL), and to dynamically capture the relevant situations at runtime.

Our situation ontology is capable of representing core concepts, domain-specific concepts, and supporting reasoning about situations according to ontology-based reasoning rules. Domain-specific concepts are the specializations of core concepts for specific domains. This separation between the core concepts and the domain-specific concepts encourages the reuse of core concepts, and provides a flexible basis for specifying domain-specific concepts.

- **Core situation concepts** represent the general situation concepts and the relationships between these concepts that are specifically identified from the access control perspective.

- **Domain-specific situation concepts** represent the domain-specific situation concepts and the relationships between these concepts based on the core concepts.

- **Reasoning rules** express the user-defined rules to infer the purpose and atomic situations based on the currently available context information, and to derive composite situations that are only implicitly present but can be inferred from available atomic situations.

In the literature, there are many languages that have been developed for specifying computer-processable semantics. Ontology-based modeling technique has been proven as a suitable logical language for modeling dynamic contexts/situations (e.g., [20], [80]). The ontology-based modeling approach to achieve situation-awareness (e.g., [66], [97]) is not only beneficial from the representational viewpoint but also beneficial from the reasoning viewpoint; that is, once facts about the world have been stated in terms of the ontology, other facts can be inferred using an inference engine through user-defined reasoning rules.

To model the situation ontology, we adopt the OWL language as an ontology lan-
language to represent the situations, which has been the most practical choice for most ontological applications because of its considered trade-off between computational complexity of reasoning and expressiveness [80]. In order to support the process of inferring new composite situations, we need to define a set of reasoning rules that are associated with the existing or already defined situations. In addition, several of the reasoning rules require mathematical computation, which is not supported by the OWL language. Towards this end, the expressivity of OWL can be extended by adding SWRL rules to an ontology. We express the user-defined reasoning rules using the OWL and SWRL languages which provide the ability to identify the purposes and to reason about new composite situations.
6.2. Ontology-Based Situation Model

6.2.1 Core Concepts

A graphical representation of the core situation ontology is shown in Figure 6.1. We model our ontology based on the 3Ws: **who** (user/role) wants to access **what** (resource/service) and **when** (relevant states and purpose). The ontology facilitates software engineers to analyze and specify purpose-oriented situation information of service invocation for access control in a situation-aware manner. The ontology is divided into two layers. The bottom layer (*Layer 1*) shows the core concepts/elements for defining the context information specific to access control, which already discussed in the last chapter (see Chapter 5). The top layer (*Layer 2*) shows the core concepts for specifying the relevant situations (atomic and composite situations) by using the relevant information from *Layer 1*. The ontology models the following core concepts.

**Definition 6.4.** *(Situation Definition in OWL)*.

```xml
<owl:ObjectProperty rdf:ID="consistsOf">
     <rdfs:domain rdf:resource="#Situation" />
     <rdfs:range>
         <owl:Class>
             <owl:unionOf rdf:parseType="Collection">
                 <owl:Class rdf:about="#Purpose" />
                 <owl:Class rdf:about="#State" />
             </owl:unionOf>
         </owl:Class>
     </rdfs:range>
 </owl:ObjectProperty>
```

The top layer has the following situation modelling concepts, which are organized into a Situation hierarchy, namely *State*, *Purpose*, *AtomicSituation*, and *CompositeSituation* classes. A *Situation* consists of the relevant *States* and the *Purpose* of user’s access request. A *Purpose* is a user’s intention in accessing the resources or services; and it can be identified based on the currently available context information. A *State* can be composed of the relevant context information. Definition 6.4 above specifies the *Situation*
definition in OWL. It shows that the Situation class has an object property consistsOf, which is used to link the Situation and the union of Purpose and State classes.

A Situation can be either a simple situation (an AtomicSituation) or a complex situation (a CompositeSituation). A CompositeSituation can be composed by one or more already defined atomic/composite situations using logical operators (by performing logical AND, OR or NOT compositions). The AtomicSituation and CompositeSituation classes are the subclasses of the Situation class. A built-in property subClassOf is used to relate the Situation class and its subclasses (see Definition 6.5).

**Definition 6.5.** (Situation Class and Its Two Subclasses).

```
<owl:Class rdf:ID="Situation">
    <owl:Class rdf:ID="AtomicSituation">
        <rdfs:subClassOf rdf:resource="#Situation"/>
    </owl:Class>
    <owl:Class rdf:ID="CompositeSituation">
        <rdfs:subClassOf rdf:resource="#Situation"/>
    </owl:Class>
</owl:Class>
```

How a new composite situation is inferred based on the atomic or composite situations by using the ontology-based reasoning rule, is discussed in Subsection 6.2.3.

**Definition 6.6.** (hasSubPurpose Object Property Definition).

```
<owl:ObjectProperty rdf:ID="hasSubPurpose">
    <rdfs:domain rdf:resource="#Purpose"/>
    <rdfs:range rdf:resource="#Purpose"/>
</owl:ObjectProperty>
```

To achieve fine-grained control over access to services, the different purposes at various granularity levels (purpose hierarchy) are need to be identified. Towards this end, we consider an object property hasSubPurpose to model the purpose hierarchy. Definition 6.6
6.2. Ontology-Based Situation Model

above shows that the class Purpose has an object property hasSubPurpose. The details of the different purpose at various granularity levels are presented in Subsection 6.2.3.

Table 6.1: An Example Atomic Situation: RNFromGWForDO

```xml
<AtomicSituation rdf:ID="s_{a2}_{RNFromGWForDO}"
    <consistsOf rdf:resource="#Purpose_{p2}_{DO}"/>
    <consistsOf rdf:resource="#State_Location_GW"/>
    <consistsOf rdf:resource="#State_ColocatedRelationship_{Yes}"/>
</AtomicSituation>
<Purpose rdf:ID="Purpose_{p2}_{DO}"
    <purposeName rdf:datatype="&xsd;string">DailyOperation</purposeName>
</Purpose>
<State rdf:ID="State_Location_GW"
    <locationAddress rdf:datatype="&xsd;string">GeneralWard</locationAddress>
</State>
<State rdf:ID="State_ColocatedRelationship_{Yes}"
    <isColocatedWith rdf:datatype="&xsd;boolean">true</isColocatedWith>
</State>
```

6.2.2 Modelling and Inferring Atomic Situation

Let us consider two atomic situations that are associated with an access control policy for the registered nurse (Policy #2 in our application scenario mentioned in Chapter 2). We have already formalized these two atomic situation in Example 8 (rules (6.9) and (6.10)).

The first atomic situation is based on the following constraints: the intendedPurpose of the requester or user, the locationAddress from which the user has initiated the request and the co-located relationship (co-located or not) between user and owner. An example policy associated with this atomic situation ‘s_{a2}’ (s_{a2} \in S_{a}) specifies a registered nurse (RN) can access a patient’s daily medical records from the general ward (GW) for daily operation (DO) purpose. The specification of this atomic situation (named s_{a2}_{RNFromGWForDO}) in OWL is shown in above Table 6.1.
The second atomic situation is based on the following constraints: the intended
Purpose and requestTime of the user. An example policy associated with this atomic
situation ‘s\textsubscript{a3}’ (s\textsubscript{a3} \in S\textsubscript{a}) specifies a registered nurse (RN) can access a patient’s daily
medical records during her ward duty time (DT) for daily operation (DO) purpose. The
specification of this atomic situation (named s\textsubscript{a3}_\text{RNAAtDTForDO}) is shown in Table 6.2.

How the currently available context information (locationAddress, isColocatedWith
and requestTime) is captured which is relevant to the above identified atomic situations,
is discussed in the previous chapter (see Chapter 5). The details of the identification of
Purpose using currently available context information are presented in the next subsection.

6.2.3 Inferring Purpose and Composite Situation

The core situation ontology (shown in Figure 6.1) serves as an entry-point for the domain
ontologies. The domain-specific concepts extend the core ontology’s corresponding general
concepts. It is important for the application developers, providing a way to include domain-
specific concepts into the core ontology. Figure 6.2 shows an excerpt of the representation

<table>
<thead>
<tr>
<th>Table 6.2: An Example Atomic Situation: RNAAtDTForDO</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;AtomicSituation rdf:ID=&quot;s\textsubscript{a3}_\text{RNAAtDTForDO}&quot;&gt;</code></td>
</tr>
<tr>
<td><code>&lt;consistsOf rdf:resource=&quot;#Purpose\_p2\_DO&quot;/&gt;</code></td>
</tr>
<tr>
<td><code>&lt;consistsOf rdf:resource=&quot;#State\_Time\_DT&quot;/&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/AtomicSituation&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Purpose rdf:ID=&quot;Purpose\_p2\_DO&quot;&gt;</code></td>
</tr>
<tr>
<td><code>&lt;purposeName rdf:datatype=&quot;&amp;xsd;string&quot;&gt;DailyOperation&lt;/purposeName&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/Purpose&gt;</code></td>
</tr>
<tr>
<td><code>&lt;State rdf:ID=&quot;State\_Time\_DT&quot;&gt;</code></td>
</tr>
<tr>
<td><code>&lt;requestTime rdf:datatype=&quot;&amp;xsd;string&quot;&gt;dutyTime&lt;/requestTime&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/State&gt;</code></td>
</tr>
</tbody>
</table>
6.2. Ontology-Based Situation Model

![Purpose Ontology Diagram]

Figure 6.2: An Excerpt of Purpose Ontology

of the Purpose ontology for the healthcare domain (e.g. treatment purpose, daily operation purpose, and research purpose) to exchange patients’ medical records.

The different purposes at various granularity levels of a user’s service access request are individually identifiable, so as to achieve fine-grained control over access to services. As such, the Purpose class contains an important data type property (xsd:int type) named granularityLevel, which indicates the granularity level (see Figure 6.2). By doing so, we can provide different levels of purpose granularity. For example, an Emergency Doctor can access a patient’s emergency medical records for the Treatment purpose, at granularityLevel 0 (highest level), which means she also can access for the specific purposes at the lower granularity levels. A General Practitioner can access some of a patient’s medical records (e.g., daily medical records) for the NormalTreatment purpose. However, she can not access a patient’s emergency medical records for EmergencyTreatment purpose.

Various types of ontology-based inferences can be performed for the identification of different purposes and reasoning about situations. Our situation ontology is extended with a set of user-defined SWRL rules to infer purpose or user’s intention in accessing the services. A purpose is identified based on the available context information. An example SWRL rule shown in Table 6.3 identifies a user’s intended Purpose is DailyOperation(DO), based on the relevant context information from the context ontology, which is proposed in Chapter 5.

127
Chapter 6. Situation Model for CAAC

Table 6.3: An Example SWRL Rules Set that Captures the Purpose is DO

\[
\text{Purpose}(?\text{purpose}) \land \text{Role}(?\text{role}) \land \text{roleIdentity}(?\text{role}, "RegisteredNurse_1") \land \\
\text{User}(?\text{user}) \land \text{hasRole}(?\text{user}, ?\text{role}) \land \text{Resource}(?\text{resource}) \land \text{Owner}(?\text{owner}) \land \\
\text{isOwnedBy}(?\text{resource}, ?\text{owner}) \land \text{healthStatus}(?\text{owner}, "normal") \rightarrow \text{intendedPurpose}(?\text{user}, ?\text{purpose}) \land \text{purposeName}(?\text{purpose}, "DailyOperation")
\]

Table 6.4: Definition of Different Situations

<table>
<thead>
<tr>
<th>Situation Name</th>
<th>Situation Definition (high-level description)</th>
</tr>
</thead>
</table>
| An emergency doctor from the hospital for emergency treatment (EDFromHospitalForET) | User_Role(EmergencyDoctor) \land \\
Location_ED(Hospital) \land \\
Purpose(EmergencyTreatment) |
| A general practitioner (who is treating physician of a patient) from the emergency room for emergency treatment (GPFromERForET) | User_Role(GeneralPractitioner) \land \\
Location_GP(EmergencyRoom) \land \\
Interpersonal_Relationship(treatingPhysician) \land \\
Purpose(EmergencyTreatment) |
| A general practitioner from the hospital for normal treatment (GPFromHospitalForNT) | User_Role(GeneralPractitioner) \land \\
Location_GP(Hospital) \land \\
Purpose(NormalTreatment) |
| A registered nurse from the general ward (where the patient is present) for daily operation (RNFromGWForDO) | User_Role(RegisteredNurse) \land \\
Location_RN(GeneralWard) \land \\
Located_Relationship(Yes) \land \\
Purpose(DailyOperation) |
| A registered nurse at ward duty time for daily operation (RNAtDTForDO) | User_Role(RegisteredNurse) \land \\
RequestTime_RN(DutyTime) \land \\
Purpose(DailyOperation) |
| A guest researcher from the hospital for research (GRFromHospitalForR) | User_Role(GuestResearcher) \land \\
Location_GR(Hospital) \land \\
Purpose(Research) |
6.2. Ontology-Based Situation Model

By using the core concepts, for the patient’s medical records management (PMRM) application (which is discussed in Chapter 2), we specify different atomic situations based on the data/information from the domain (e.g., healthcare domain). Some of these ‘situations’ and their associated ‘context information’ and ‘purpose’ using situation reasoning rules are shown in Table 6.4.

Our situation ontology can be extended with ontology-based reasoning rules to infer new composite situations. The atomic situations can be explicitly identified by the situation ontology and using the captured context information from the context ontology. These identified atomic situations are used to infer new composite situations by using ontology-based reasoning rules.

Table 6.5: An Example Composite Situation: RNFromGWaDTForDO

```
<CompositeSituation rdf:ID="s_c1_RNFromGWaDTForDO">  
  <supportedSituation rdf:resource="#s_a2_RNFomGWForDO"/>  
  <supportedSituation rdf:resource="#s_a3_RNAatDTForDO"/>  
</CompositeSituation>

<Situation>
  <owl:intersectionOf rdf:parseType="Collection">
    <AtomicSituation rdf:about="#s_a2_RNFomGWForDO"/>
    <AtomicSituation rdf:about="#s_a3_RNAatDTForDO"/>
  </owl:intersectionOf>
  <rdfs:subClassOf>
    <Situation rdf:about="#CompositeSituation"/>
  </rdfs:subClassOf>
</Situation>
```

A set of reasoning rules are specified for implicit knowledge reasoning, which reasons about the implicit knowledge conveyed by the specification. For example, the reasoning rule specified in Example 8 (see rule (6.11)) is written in OWL/XML that is used to reason about a new composite situation \((s_{c1} = s_{a2} \cap s_{a3}, s_{c1} \subseteq S_c)\). The specification of these two
atomic situations ($s_{a2}$ and $s_{a3}$) in OWL are shown in Table 6.1 and Table 6.2. An example policy associated with this composite situation $s_{c1}$ specifies a registered nurse (RN) can access a patient’s private medical records (PMR) during her ward duty time (DT) from the general ward (GW) where the patient is located, for daily operation (DO) purpose. The specification of this composite situation (named $s_{c1\_RNFromGWatDTForDO}$) is shown in Table 6.5.

6.3 Summary

In this chapter, we have introduced a formalization of the basic components of the situation model for access control and proposed an ontology-based situation model for our context-aware access control (CAAC) framework. It includes a situation ontology specific to access control for specifying the relevant situations (atomic situations), and a reasoning technique for inferring composite situations based on ontology-based reasoning rules. The situation ontology can capture the relevant elementary information from the environments (the relevant states and the user’s intention or purpose) in order to identify relevant situations. The proposed purpose-oriented situation model significantly differs from the existing situation models in that it considers the relevant purpose-oriented situations rather than conventional situations (e.g., user’s state). By introducing the concept of purpose-oriented situation, the purpose-oriented control over access to information resources or software services has been achieved.

In the next Chapter, we present the policy model (see Chapter 7) for access control. Using the context model (which is presented in Chapter 5) and situation model (which is presented in this Chapter), the policy model supports access control to information resources or software services based on the relevant context/situation information.
For controlling access to information resources or software services, in this chapter we introduce an ontology-based policy model for our context-aware access control (CAAC) framework. Using the context and situation models proposed in the previous chapters (Chapter 5 and Chapter 6), the policy model provides dynamic assignments of user-role and role-permission capabilities. The model uses the relevant context and situation information that reflects the dynamically changing conditions of the environments, in order to specify the context-aware access control policies: the context-aware user-role and role-permission assignment policies. The first set of policies specifies that users can play the roles when a set of conditions are satisfied. The second set of policies specifies that users having roles are allowed to carry out an operation on the resource when a set of conditions are satisfied. Overall, these policy sets let users to access resources or services when certain (dynamically changing) contextual conditions are satisfied.

The structure of this chapter is as follows. Section 7.1 defines the CAAC policy concepts for context-aware access control. In Section 7.2, we present an ontology-based policy model, including the specification of context-aware user-role and role-permission assignment policies (see Section 7.2.1), and the example policies from the healthcare domain (see Section 7.2.2). Finally, Section 7.3 summarizes the chapter.

### 7.1 Defining CAAC Policy

A recent study [75] shows that the basic Role-based Access Control (RBAC) model has become the widely used access control model and the most attractive solution for providing security features. RBAC typically evaluates access permission through roles
assigned to users and permissions assigned to roles (a permission is an approval of a particular mode of access to the information resources or services) [84].

Based on the formalization of the RBAC policy model, we present a formal definition of our policy model. Our policy model for CAAC applications that extends RBAC with relevant dynamic information (context/situation information, which is defined in the previous chapters). Our goal in this research is to extend the user-role and role-permission assignment policies in RBAC [84] by incorporating dynamic context/situation information as policy constraints.

Our CAAC policy model consists of two sets of policies: the context-aware user-role assignment (CAURA) policy and context-aware role-permission assignment (CARPA) policy. The policy model uses the (dynamically changing) contextual conditions in order to enable context-aware user-role and role-permission assignments.

In the following, we present a formalization of the basic concepts of our CAAC policy model: the CAURA policy and CARPA policy.

7.1.1 CAURA Policy

We formulate our CAURA policy ontology based on the 3Ws: who (user) can play what (role) and when (relevant contextual conditions).

Our policy model extends the concept of basic user-role assignments (URA) in RBAC ($URA \subseteq U \times R$) [84], by introducing the concept of context-aware user-role assignments.

**Definition 7.1. (CAURA Policy.)** The CAURA policy for context-aware access control specifies the dynamic assignments of users to roles when a set of contextual conditions are satisfied. (see Formula (7.1)):

$$CAURA = \{(u_1, r_1, exp_1), (u_2, r_2, exp_2), ..., (u_i, r_j, exp_k)\} \subseteq U \times R \times Exp$$ (7.1)
7.1. Defining CAAC Policy

- **User (U):** $U$ represents a set of users, $U = \{u_1, u_2, ..., u_i\}$. The users or service requesters are human-beings, whose access requests are being controlled.

- **Roles (R):** $R$ represents a set of roles that reflect user’s job functions within the organization (e.g., healthcare domain), $R = \{r_1, r_2, ..., r_j\}$.

- **Expressions (Exp):** $Exp$ represents a set of contextual conditions based on the dynamically changing context or situation information, $Exp = \{exp_1, exp_2, ..., exp_k\}$.

As we discussed in the conceptual CAAC framework chapter (Chapter 4), a contextual expression $exp$ ($exp \in Exp$) is formed by using a set of contextual conditions. It can be a simple context, complex context, atomic situation or composite situation.

\[
exp = c_s | c_c | s_a | s_c
\]  

(7.2)

Where,

- $c_s$ denotes a simple context information (a context attribute of an entity based on a single context information source).

- $c_c$ denotes a complex context information (based on the various context information sources).

- $s_a$ denotes an atomic situation.

- $s_c$ denotes a composite situation.

**Example 9.** Considering our application scenario (Scene #2) presented in Chapter 2, the user-role assignment policy associated with this scene can be read as: a user Mary can play the registered nurse (RN) role when she is located with the patient in the general ward of the hospital, when the patient’s health condition is normal. Based on our CAURA policy, (for $User(u_1) \land Role(r_1) \land Expression(exp_1)$; $(u_1, r_1, exp_1) \in CAURA$), the rule shown in Table 7.1 expresses the policy, i.e., a user Mary can play the registered nurse (RN) role, if a situation $s_{a2}$ is satisfied. The atomic situation $s_{a2}$ is already specified in the previous chapter (see Example 6.10 in Chapter 6).
Table 7.1: An Example Context-Aware User-Role Assignment Policy

\[
\begin{align*}
&\text{If} \\
&\text{CAURAPolicy}(ur_1) \land \text{User}(u_1) \land \text{hasUser}(ur_1,u_1) \land \text{Role}(r_1) \land \\
&\text{equal}(r_1, "RN") \land \text{hasRole}(ur_1,r_1) \land \text{Situation}(s_{a2}) \land \text{hasSituation}(ur_1,s_{a2}) \\
&\text{Then} \\
&\text{canPlay}(u_1,r_1)
\end{align*}
\]

7.1.2 CARPA Policy

We formulate our CARPA policy ontology based on the 3Ws: who (role) wants to access what (resource) and when (relevant contextual conditions).

Our policy model extends the concept of basic role-permission assignments (RPA) in RBAC \((RPA \subseteq R \times P)\) [84], by introducing the concept of context-aware role-permission assignments.

**Definition 7.2.** (CARPA Policy.) The CARPA policy for context-aware access control specifies the dynamic assignments of permissions (service access permissions) to roles when a set of contextual conditions are satisfied. (see Formula (7.3)):

\[
\text{CARPA} = \{(r_1,p_1,exp_1),(r_2,p_2,exp_2),\ldots,(r_j,p_l,exp_k)\} \subseteq R \times P \times Exp \tag{7.3}
\]

- **Roles (R):** \(R\) represents a set of roles that reflect user’s job functions or job titles within the organization, \(R = \{r_1,r_2,\ldots,r_j\}\).

- **Permissions (P):** \(P\) represents a set of permissions, \(P = \{p_1,p_2,\ldots,p_l\}\). The permissions (service access permissions) are the sets of operations on the resources, \(\text{service} = (\text{res}, \text{op})\), where \(\text{res}\) denotes a resource and \(\text{op}\) denotes an operation on the resource \(\text{res}\).

- **Expressions (Exp):** \(Exp\) represents a set of contextual conditions based on the dynamic information, \(Exp = \{exp_1,exp_2,\ldots,exp_k\}\).

**Example 10.** Considering our application scenario (Scene #1) presented in Chapter 2, the role-permission assignment policy associated with this scene can be read as: the emergency doctors can access (write operation) a patient’s emergency medical records (EMR) in

---

134
the hospital for emergency treatment purpose, when the patient is in a critical health condition. Based on our CARPA policy, (for Role($r_2$) ∧ Permission($p_2$) ∧ Expression($exp_2$); ($r_2, p_2, exp_2$) ∈ CARPA), the rule shown in Table 7.2 expresses the policy, i.e., the emergency doctors (ED) can have the permission (writeEMR() service access permission), if a situation $s_{a1}$ is satisfied. The situation $s_{a1}$ is already specified in the previous chapter (see Examples 6 and 7 in Chapter 6).

7.2 Ontology-Based Policy Model

The identification of the relevant concepts to specify the context-aware user-role and role-permission assignment policies was discussed earlier. In this section, we present an ontology-based policy model for context-aware access control (CAAC), supporting policy specification (integrating relevant dynamic information into the access control policies).

The main goal of our policy ontology is to specify two sets of context-aware access control policies by incorporating the dynamic information (context/situation information). Our policy ontology is a hierarchical model that can be extended by a set of user-defined rules using application specific concepts/elements (e.g., healthcare domain concepts). An ontology representing these concepts can fulfill the following requirements:

- Representing all the policy concepts or elements (general and domain-specific) and the relationships between these elements.

- Specifying the CAAC policies (CAURA and CARPA policies) as user-defined rules based on the general and domain-specific concepts.
To simplify the management of access control policies, various policy languages have been proposed in the literature. Our goal in this research is to provide a way in which context-aware access control policies can be specified, which incorporate dynamic context or situation information. To be of practical use, it must be expressive enough to specify the policies in an easy and natural way. In particular, it needs to use the information from the context and situation models that is proposed in earlier chapters (see Chapter 5 and Chapter 6), in order to specify under which conditions the requested resource or service is accessible. To do so, we use the same ontology language OWL as the CAAC policy language.

7.2.1 Policy Specification

The policy ontology specifies two sets of context-aware access control (CAAC) policies: context-aware user-role and role-permission assignment policies. The basic concept of CAAC is that users are dynamically assigned to roles by satisfying the relevant conditions, service access permissions are dynamically assigned to roles by satisfying the relevant conditions and users acquire service access permissions by having corresponding roles.

Figure 7.1 shows the top-level conceptual view of our policy ontology. The ontology defines the following concepts under the hierarchy of CAACPolicy, namely CAURAPolicy and CARPAPolicy. The CAURAPolicy models context-aware user-role assignments and CARPAPolicy models context-aware role-permission assignments.
CAURA Policy Ontology

The CAURA policy ontology, representing context-aware user-role assignment policies, has been designed by answering the following questions.

- Who is requesting resource/service access (requester or user)?
- What role does the user play (role)?
- What is the dynamic contextual information that is relevant for this user-role assignment (context/situation)?

![CAURA Policy Ontology Diagram]

Figure 7.2: The CAURA Policy Ontology

The CAURA policy ontology, as depicted in Figure 7.2, has the following concepts which are organized into a hierarchy CAURAPolicy, namely User, Role and ContextualCondition. The ContextualCondition class uses the concepts (ContextInfo and Situation) from the context and situation models introduced in the previous chapters. The CAURA ontology also uses the concepts, User and Role, in order to capture the relevant context information at runtime. The reused classes are shown in shaded ellipses (see Figure 7.2).

The following code in OWL (see Definition 7.3) shows the class CAURAPolicy has an object property hasUser, which is used to link the classes CAURAPolicy and User.
Definition 7.3. (‘hasUser’ Object Property Definition).

```xml
<owl:Class rdf:ID="CAURAPolicy"/>
<owl:Class rdf:ID="User"/>
<owl:ObjectProperty rdf:ID="hasUser">
  <rdfs:domain rdf:resource="#CAURAPolicy"/>
  <rdfs:range rdf:resource="#User"/>
</owl:ObjectProperty>
```

Similar to Definition 7.3, we define two other object properties hasRole and hasCondition. The property hasRole is used to link the classes CAURAPolicy and Role (see Definition 7.4), and the property hasCondition links the classes CAURAPolicy and ContextualCondition (see Definition 7.5).

Definition 7.4. (‘hasRole’ Object Property Definition).

```xml
<owl:ObjectProperty rdf:ID="hasRole">
  <rdfs:domain rdf:resource="#CAURAPolicy"/>
  <rdfs:range rdf:resource="#Role"/>
</owl:ObjectProperty>
```

Definition 7.5. (‘hasCondition’ Object Property Definition).

```xml
<owl:ObjectProperty rdf:ID="hasCondition">
  <rdfs:domain rdf:resource="#CAURAPolicy"/>
  <rdfs:range rdf:resource="#ContextualCondition"/>
</owl:ObjectProperty>
```

The following OWL codes show that the class User has a data type property userIdentity, which is xsd:string type (see Definition 7.6), and the class Role has a xsd:string type property, named roleIdentity (see Definition 7.7).
7.2. Ontology-Based Policy Model

**Definition 7.6.** (‘userIdentity’ Data Type Property Definition).

```xml
<owl:DatatypeProperty rdf:ID="userIdentity">
    <rdfs:domain rdf:resource="#User"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

**Definition 7.7.** (‘roleIdentity’ Data Type Property Definition).

```xml
<owl:DatatypeProperty rdf:ID="roleIdentity">
    <rdfs:domain rdf:resource="#Role"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

A CAURA policy captures the who/what/when dimensions which can be read as follows: a CAURAPolicy species that a user (userIdentity) can play a Role (roleIdentity) by satisfying the relevant access conditions.

**CARPA Policy Ontology**

The CARPA policy ontology, representing context-aware role-permission assignment policies, has been designed by answering the following questions.

- Who is requesting access by playing what role (role)?
- What type of object is being requested (resource or service)?
- What is the dynamic contextual information that is relevant for this role-permission assignment (context/situation)?

A graphical representation of the ontology is shown in Figure 7.3. The ontology has the following concepts, which are organized into a hierarchy CARPA Policy, namely Role, Permission, Resource, Operation, Access Decision and Contextual Condition. The Contextual Condition class uses the concepts (ContextInfo and Situation) from the context and situation models introduced in the previous chapters. The CAURA ontology also uses
the concepts, Role and Resource, in order to capture the relevant context information at runtime. The reused classes are shown in shaded ellipses (see Figure 7.3).

**Definition 7.8.** (*Cardinality Constraint Definition of a Property ‘decision’*).

```xml
<owl:Class rdf:ID="AccessDecision"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#decision"/>
  <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction>
</owl:Class>
```

We specify that a CARPA policy has exactly one access decision value ("Granted" or "Denied"). To specify this cardinality constraint, in our CARPA policy ontology (see Figure 7.3), we consider a class `AccessDecision`, and a data type property `decision` for the `AccessDecision` class: each `AccessDecision` class instance has exactly one `decision` attribute.
7.2. Ontology-Based Policy Model

value, which means the value of the decision attribute may be “Granted” or “Denied”. It is not possible to specify the value of the decision attribute is both “Granted” and “Denied” for an access policy. The following OWL code (see Definition 7.8) specifies the cardinality of the class AccessDecision on the property decision.

The following OWL code shows that the class AccessDecision has an object property hasDecision, which links the classes CARPAPolicy and AccessDecision (see Definition 7.9).

**Definition 7.9.** (‘hasDecision’ Object Property Definition).

```owl
<owl:Class rdf:ID="CARPAPolicy"/>
<owl:Class rdf:ID="AccessDecision"/>
<owl:ObjectProperty rdf:ID="hasDecision">
  <rdfs:domain rdf:resource="#CARPAPolicy"/>
  <rdfs:range rdf:resource="#AccessDecision"/>
</owl:ObjectProperty>
```

**Definition 7.10.** (‘hasResource’ Object Property Definition).

```owl
<owl:Class rdf:ID="Permission"/>
<owl:Class rdf:ID="Resource"/>
<owl:ObjectProperty rdf:ID="hasResource">
  <rdfs:domain rdf:resource="#Permission"/>
  <rdfs:range rdf:resource="#Resource"/>
</owl:ObjectProperty>
```

The class Permission links to the classes Resource and Operation using two object properties hasResource and hasOperation. The hasResource property is used to link the classes Permission and Resource (see Definition 7.10), and the hasOperation property links the classes Permission and Operation (see Definition 7.11).
Definition 7.11. (‘hasOperation’ Object Property Definition).

\[
\text{<owl:Class rdf:ID="Operation"/>}
\]
\[
\text{<owl:ObjectProperty rdf:ID="hasOperation">}
\]
\[
\text{<rdfs:domain rdf:resource="#Permission"/>}
\]
\[
\text{<rdfs:range rdf:resource="#Operation"/>}
\]
\[
\text{</owl:ObjectProperty>}
\]

The Operation class has a data type property action (xsd:string type) in order to capture the operation on the resource. The following OWL code shows the property definition (see definition 7.12).

Definition 7.12. (‘action’ Data Type Property Definition).

\[
\text{<owl:DatatypeProperty rdf:ID="action">}
\]
\[
\text{<rdfs:domain rdf:resource="#Operation"/>}
\]
\[
\text{<rdfs:range rdf:resource="#xsd:string"/>}
\]
\[
\text{</owl:DatatypeProperty>}
\]

A CARPA policy captures the who/what/when dimensions which can be read as follows: a CARPA policy species that a user who is playing a Role has AccessDecision (“Granted” or “Denied”) to which parts (resourceIdentity) of a Resource (at which levels of granularity) for a specific action (“Read” or “Write” Operation) or range of actions under which access conditions.

7.2.2 Example Policies

In this section, we show some policy examples based on the healthcare application scenario presented in Chapter 2.

Revisiting the Application Scenario - Scene #1

Example 11. Let us consider an access control policy for the emergency doctors (Policy #1 in the application scenario presented in Chapter 2): a user Jane can play the emergency
7.2. Ontology-Based Policy Model

doctors (ED) role in the hospital (in order to access the patient’s medical records when the patient is in a critical health condition).

Table 7.3: An Example CAURAPolicy for the Emergency Doctor Role

<table>
<thead>
<tr>
<th>Line</th>
<th>CAURAPolicy RDF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>&lt;CAURAPolicy rdf:ID=&quot;ur2&quot;&gt;</code></td>
<td>Core policy concepts defined.</td>
</tr>
<tr>
<td>2</td>
<td><code>&lt;hasUser rdf:resource=&quot;#User_EmergencyDoctor_DB&quot;/&gt;</code></td>
<td>User specification.</td>
</tr>
<tr>
<td>3</td>
<td><code>&lt;hasRole rdf:resource=&quot;#Role_EmergencyDoctor&quot;/&gt;</code></td>
<td>Role specification.</td>
</tr>
<tr>
<td>4</td>
<td><code>&lt;hasCondition rdf:resource=&quot;#ContextualCondition_Situation&quot;/&gt;</code></td>
<td>Contextual condition.</td>
</tr>
<tr>
<td>6</td>
<td><code>&lt;/CAURAPolicy&gt;</code></td>
<td>End of policy definition.</td>
</tr>
<tr>
<td>7</td>
<td><code>&lt;User rdf:ID=&quot;User_EmergencyDoctor_DB&quot;&gt;</code></td>
<td>User specification.</td>
</tr>
<tr>
<td>8</td>
<td><code>&lt;userIdentity rdf:datatype=&quot;&amp;xsd;string&quot;&gt;Jane00X&lt;/userIdentity&gt;</code></td>
<td>User identity.</td>
</tr>
<tr>
<td>9</td>
<td><code>&lt;/User&gt;</code></td>
<td>End of user specification.</td>
</tr>
<tr>
<td>10</td>
<td><code>&lt;Role rdf:ID=&quot;Role_EmergencyDoctor&quot;&gt;</code></td>
<td>Role specification.</td>
</tr>
<tr>
<td>11</td>
<td><code>&lt;roleIdentity rdf:datatype=&quot;&amp;xsd;string&quot;&gt;ED00X&lt;/roleIdentity&gt;</code></td>
<td>Role identity.</td>
</tr>
<tr>
<td>12</td>
<td><code>&lt;/Role&gt;</code></td>
<td>End of role specification.</td>
</tr>
<tr>
<td>13</td>
<td><code>&lt;ContextualCondition rdf:ID=&quot;ContextualCondition_Situation&quot;&gt;</code></td>
<td>Contextual condition.</td>
</tr>
<tr>
<td>14</td>
<td><code>&lt;uses rdf:resource=&quot;#AtomicSituation_s_a1_EDFromHospitalForET&quot;/&gt;</code></td>
<td>Atomic situation.</td>
</tr>
<tr>
<td>15</td>
<td><code>&lt;/ContextualCondition&gt;</code></td>
<td>End of contextual condition.</td>
</tr>
</tbody>
</table>

In this policy, the access decision is based on the following policy constraints: who the user is (user’s identity), what role the user can play (role’s identity), and when (the location of the user and the health status of the patient). The CAURAPolicy for the emergency doctors in OWL is shown in Table 7.3. The core policy concepts are specified in Line# 1 to 6, the user specification is shown in Line# 7 to 9, the role specification (emergency doctor) is shown in Line# 10 to 12, and the contextual condition (an atomic situation $s_{a1}$, emergency doctors in the hospital for emergency treatment purpose when the patient’s health condition is critical) is specified in Line# 13 to 15. The specification of this situation ($s_{a1}_{EDFromHospitalForET}$) is shown in the previous chapter (see Chapter 6).

**Example 12.** Consider the same access control policy for the emergency doctors (Policy#1 in the application scenario presented in Chapter 2): the emergency doctors (ED) can access
Table 7.4: An Example CARPA Policy for the Emergency Doctor

```
1 <CARPAPolicy rdf:ID="rp2">
2   <hasRole rdf:resource="#Role_EmergencyDoctor"/>
3   <hasPermission rdf:resource="#Permission_EMR_Read_Write"/>
4   <hasDecision rdf:resource="#AccessDecision_Granted"/>
5   <hasCondition rdf:resource="#ContextualCondition_Situation"/>
6 </CARPAPolicy>

7 <Role rdf:ID="Role_EmergencyDoctor">
8   <roleIdentity rdf:datatype="&xsd;string">ED00X</roleIdentity>
9 </Role>

10 <Permission rdf:ID="Permission_EMR_Read_Write">
11    <hasResource rdf:resource="#Resource_EMR"/>
12    <hasOperation rdf:resource="#Operation_Read_Write"/>
13 </Permission>

14 <Resource rdf:ID="Resource_EMR">
15    <resourceIdentity rdf:datatype="&xsd;int">1</resourceIdentity>
16    <granularityLevel rdf:datatype="&xsd;int">0</granularityLevel>
17 </Resource>

18 <Operation rdf:ID="Operation_Read_Write">
19    <action rdf:datatype="&xsd;string">Read</action>
20    <action rdf:datatype="&xsd;string">Write</action>
21 </Operation>

22 <AccessDecision rdf:ID="AccessDecision_Granted">
23    <decision rdf:datatype="&xsd;string">Granted</decision>
24 </AccessDecision>

25 <ContextualCondition rdf:ID="ContextualCondition_Situation">
26    <uses rdf:resource="#AtomicSituation_sa1_EDFromHospitalForET"/>
27 </ContextualCondition>
```

(Read/Write operation) the patient’s emergency medical records (EMR) in the hospital for emergency treatment (ET) purpose, when the patient is in a critical health condition.
7.2. Ontology-Based Policy Model

In this policy, the access decision is based on the following policy constraints: **who** the user is (user’s role), **what** resource is being requested (resource’s identity and granularity level), and **when** the user sends the request (the location of the user, the health status of the patient, and the purpose or user intention in accessing the resource). The CARPA policy for the emergency doctors in OWL is shown in Table 7.4. The core policy concepts are specified in Line# 1 to 6, the role specification (emergency doctor) is shown in Line# 7 to 9, the permission specification (emergency medical records on read/write operation) is shown in Line# 10 to 21, and the access decision (granted decision) is specified in Line# 22 to 24. The contextual condition (an atomic situation $s_{a1}$, emergency doctors in the hospital for emergency treatment purpose when the patient’s health condition is critical) is specified in Line# 25 to 27. The specification of this atomic situation ($s_{a1}_{-}EDFromHospitalForET$) is shown in the previous chapter (see Chapter 6).

**Revisiting the Application Scenario - Scene #2**

**Example 13.** Consider the policy presented in Chapter 2 (Policy#2): a user Mary can play the registered nurse (RN) role (in order to access a patient’s private medical records), during her ward duty time (DT) from the general ward (GW) of the hospital, where the patient is located.

In this policy, the access decision is based on the following policy constraints: **who** the user is (user’s identity), **what** role she can play (role’s identity), and **when** (the locations of the user and patient and the request time of the user). The CAURA policy for the registered nurses in OWL is shown in Table 7.5. The core policy concepts are specified in Line# 1 to 6, the user specification is shown in Line# 7 to 9, the role specification (registered nurse) is shown in Line# 10 to 12, and the contextual condition (a composite situation $s_{c1}$, (registered nurses during ward duty time from the general ward of the hospital for daily operation purpose when the patient’s health condition is normal) is specified in Line# 13 to 15. The specification of this composite situation $s_{c1}_{-}RNFromGWAtDTForDO$ is shown in the previous chapter (see Table 6.5 in Chapter 6).
Example 14. Consider the same access control policy for the registered nurse (Policy#2 in the application scenario presented in Chapter 2): the registered nurses can access a patient’s private medical records (PMR) during her ward duty time (DT) from the general ward (GW) of the hospital where the patient is located, for daily operation (DO) purpose and when the patient health condition is normal.

In this policy, the access decision is based on the following policy constraints: who the user is (user’s role), what resource is being requested (resource’s identity and granularity level), and when the user sends the request (the co-location relationship between user and resource owner, the health status of the patient, the locations of the user and patient, the request time of the user, and the purpose or user intention in accessing the resource). The CARPA policy for the registered nurses in OWL is shown in Table 7.6. The policy states that the registered nurses can access the patient’s private medical records during her ward shift time for daily operation purpose if they both are co-located. The specification of the composite situation associated with this policy, named \( s_{c1\_RNFromGWAtDTForDO} \) (registered nurses during ward duty time from the general ward of the hospital for daily
Table 7.6: An Example CARPA Policy for the Registered Nurse

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>&lt;CARPAPolicy rdf:ID=&quot;rp1&quot;</code></td>
</tr>
<tr>
<td>2</td>
<td><code>&lt;hasRole rdf:resource=&quot;#Role_RegisteredNurse&quot;</code></td>
</tr>
<tr>
<td>3</td>
<td><code>&lt;hasPermission rdf:resource=&quot;#Permission_PMR_Read_Write&quot;</code></td>
</tr>
<tr>
<td>4</td>
<td><code>&lt;hasCondition rdf:resource=&quot;#ContextualCondition_Situation&quot;</code></td>
</tr>
<tr>
<td>5</td>
<td><code>&lt;hasDecision rdf:resource=&quot;#AccessDecision_Granted&quot;</code></td>
</tr>
<tr>
<td>6</td>
<td><code>&lt;/CARPAPolicy&gt;</code></td>
</tr>
<tr>
<td>7</td>
<td><code>&lt;Role rdf:ID=&quot;Role_RegisteredNurse&quot;</code></td>
</tr>
<tr>
<td>8</td>
<td><code>&lt;roleIdentity rdf:datatype=&quot;&amp;xsd:string&quot;&gt;RN00X&lt;/roleIdentity&gt;</code></td>
</tr>
<tr>
<td>9</td>
<td><code>&lt;/Role&gt;</code></td>
</tr>
<tr>
<td>10</td>
<td><code>&lt;Permission rdf:ID=&quot;Permission_PMR_Read_Write&quot;</code></td>
</tr>
<tr>
<td>11</td>
<td><code>&lt;hasResource rdf:resource=&quot;#Resource_PMR&quot;</code></td>
</tr>
<tr>
<td>12</td>
<td><code>&lt;hasOperation rdf:resource=&quot;#Operation_Read_Write&quot;</code></td>
</tr>
<tr>
<td>13</td>
<td><code>&lt;/Permission&gt;</code></td>
</tr>
<tr>
<td>14</td>
<td><code>&lt;Resource rdf:ID=&quot;Resource_PMR&quot;</code></td>
</tr>
<tr>
<td>15</td>
<td><code>&lt;resourceIdentity rdf:datatype=&quot;&amp;xsd:int&quot;&gt;4&lt;/resourceIdentity&gt;</code></td>
</tr>
<tr>
<td>16</td>
<td><code>&lt;granularityLevel rdf:datatype=&quot;&amp;xsd:int&quot;&gt;1&lt;/granularityLevel&gt;</code></td>
</tr>
<tr>
<td>17</td>
<td><code>&lt;/Resource&gt;</code></td>
</tr>
<tr>
<td>18</td>
<td><code>&lt;Operation rdf:ID=&quot;Operation_Read_Write&quot;</code></td>
</tr>
<tr>
<td>19</td>
<td><code>&lt;action rdf:datatype=&quot;&amp;xsd:string&quot;&gt;Read&lt;/action&gt;</code></td>
</tr>
<tr>
<td>20</td>
<td><code>&lt;action rdf:datatype=&quot;&amp;xsd:string&quot;&gt;Write&lt;/action&gt;</code></td>
</tr>
<tr>
<td>21</td>
<td><code>&lt;/Operation&gt;</code></td>
</tr>
<tr>
<td>22</td>
<td><code>&lt;AccessDecision rdf:ID=&quot;AccessDecision_Granted&quot;</code></td>
</tr>
<tr>
<td>23</td>
<td><code>&lt;decision rdf:datatype=&quot;&amp;xsd:string&quot;&gt;Granted&lt;/decision&gt;</code></td>
</tr>
<tr>
<td>24</td>
<td><code>&lt;/AccessDecision&gt;</code></td>
</tr>
<tr>
<td>25</td>
<td><code>&lt;ContextualCondition rdf:ID=&quot;ContextualCondition_Situation&quot;</code></td>
</tr>
<tr>
<td>26</td>
<td><code>&lt;uses rdf:resource=&quot;&amp;#CompositeSituation_s01_RNFromGWATDTForDO&quot;</code></td>
</tr>
<tr>
<td>27</td>
<td><code>&lt;/ContextualCondition&gt;</code></td>
</tr>
</tbody>
</table>

operation purpose when the patient’s health condition is normal), is shown in Table 6.5 in Chapter 6.
Discussion

Using our CAURA and CARPA policy models, users acquire service access permissions by having corresponding roles. For example, in the above-mentioned scenario (Scene #2), Mary can access a patient Bob’s private medical records by playing the registered nurse role. We can observe that if Mary is located in the general ward during her ward duty time, she is authorized to access the private medical records of the patient Bob, who is hosted in that ward in his normal health condition. Conversely, when context changes, she is not allowed to access such records. That is, when Mary leaves the ward or before the beginning of her ward duty time, or after the end of her duty time, she is not allowed to play the registered nurse role (Mary’s membership in the nurse role is revoked).

7.3 Summary

In this Chapter, we have introduced a formalization of the basic concepts of the policy model for context-aware access control (CAAC) and proposed an ontology-based policy model for our CAAC framework. It includes two sets of context-aware access control policies. The first set specifies the context-aware user-role assignment policies and the second set specifies the context-aware role-permission assignment policies. Our policy model significantly differs from the existing policy models in that it considers the dynamic associations of user to role and role to permission (service access permission), for providing appropriate service access permission to user by having appropriate role. The model uses the relevant dynamic information from the context and situation models. The example policies from the healthcare domain are specified by using our policy models.

In the following chapters, we present the implementation of the CAAC framework (see Chapter 8), to demonstrate the practical applicability of our framework; and evaluate the proposed CAAC framework (see Chapter 9), to show the viability of the framework.
Part III
Implementation of the CAAC Framework

Chapters 4, 5, 6, and 7 presented the theoretical foundations of the context-aware access control approach, including an ontology-based context model, situation model, and policy model for the construction of our context-aware access control (CAAC) framework. In this chapter, the development of a software prototype of the CAAC framework built upon these foundations is considered. The design of this architecture is inspired by the basic layers in developing the context-aware systems: context gathering, context management and application layers. In order to illustrate the use of the CAAC framework, we develop a context-aware access control application in the domain of patient medical records management using the software prototype.

The structure of this chapter is as follows. Section 8.1 presents the design and implementation of the CAAC framework, including a software architecture of the CAAC framework (see Section 8.1.1) and the data flow sequences of the prototype components (see Section 8.1.2). Section 8.2 shows the technology used to implement the prototype components. Section 8.3 presents a context-aware access control application in the healthcare domain, to illustrate the use and practical applicability of our CAAC framework. Finally, Section 8.4 summarizes the chapter.

8.1 Design and Implementation of the CAAC Framework

In this section, we present a prototype implementation of the CAAC framework for developing context-aware access control applications.
The advancement of computing technology necessitates more advanced access control frameworks to be built with context-awareness capabilities and adaptability to their changing contexts or situations. Today, building context-aware access control applications is a complex task due to the lack of an adequate infrastructure support in dynamic and context-aware environments. Towards this goal, in this thesis we have presented a context-aware access control (CAAC) framework, including an ontology-based context model, situation model and policy model for CAAC. Based on the context, situation and policy models, we design a software prototype for building and rapid prototyping of context-aware access control applications. The prototype framework provides an efficient infrastructure support for building context-aware access control applications in dynamic and context-aware environments.

8.1.1 System Architecture of the Prototype Framework

The overall system architecture of our prototype framework is shown in Figure 8.1, which depicts the functional components of the architecture. The architecture consists of three layers: Sensor, Middleware, and Application. The CAAC Middleware Layer has the following main components: Context Manager, Situation Manager, and Access Control Manager. The Middleware Layer captures raw context data from various data sources in the Sensor Layer, infers the high-level contextual information (context and situation), and supports the building of context-aware access control applications by the software engineers, such as the patient medical record management (PMRM) system, in the Application Layer.

The functional components related to sensor and application layers are application-specific, which are the outside of our research scope. In this thesis, our main focus is the CAAC middleware layer and its associated components.

In our prototype, the Context Manager provides functionalities to manipulate the context ontology (capturing low-level context information and inferring high-level context information). The Context Manager consists of a number of Context Providers and a Context Reasoner. The Context Providers capture the raw context data from the sensors,
8.1. Design and Implementation of the CAAC Framework

Figure 8.1: The Overall Architecture of the CAAC Prototype Framework
extract the low-level context information, and provide this low-level context information to the Context Ontology Knowledge Base (KB). In this process, the Context Providers convert this information to OWL representation (according to the context ontology) so that it can be shared and reused by other software components. The Context Reasoner consists of the Context Inference Engine and the Context Rule Base. It deduces the high-level context information from the low-level context information using the defined rules, and provides this high-level context information back to the Context Ontology KB.

Similar to context manager, the Situation Manager consists of the Situation Inference Engine and the Situation Rule Base. The Situation Manager identifies the relevant situations, by inferring the states of the relevant context entities, the states of the relevant relationships between entities and the purpose from the available context information, using the defined rules.

The Access Control Manager consists of the Policy Manager and the Query Manager. The CAAC Policy Decision Point is implemented as part of the Policy Manager and the CAAC Policy Enforcement Point is implemented as part of the Query Manager, inspired by the XACML architecture [73]. The Policy Manager also has the Policy Ontology KB, and the Query Manager has the Access Query. The CAAC Policy Decision Point is used to allow the CAAC application developers to add, edit and delete context-aware access control policies, based on the Policy Ontology KB. The CAAC Policy Enforcement Point queries the CAAC Policy Decision Point for the relevant policies, and then checks the user’s request and makes the access control decisions using the relevant context/situation information in the Context/Situation Ontology KB. In addition, the Query Manager allows application developers to add, edit and delete the access queries, in order to check ‘who can access what resources’.

8.1.2 Data Flow Sequences of the Prototype Components

Using our developed prototype, the context-aware access control decisions are performed according to the following data flow sequences:
8.2 Technology Used to Implement Prototype Components

1. The Users (access requesters) send the access requests to access the Resources which get processed by our developed CAAC Policy Enforcement Point.

2. Once receiving the request for resource access, the CAAC Policy Enforcement Point queries the applicable access control policies and the relevant contextual information (context/situation) to the CAAC Policy Decision Point.

3. Based on the selected policies, the CAAC Policy Decision Point makes the access control decision by runtime assessing the relevant context and situation information from the Context Manager and Situation Manager respectively.

4. The decision made by CAAC Policy Decision Point is passed to CAAC Policy Enforcement Point. If it is admissible, the access request is granted, otherwise, it is denied. Being granted to access the resources means that the Users are able to access (read, write or both) the Resources. If the decision is positive or "granted decision", the CAAC Policy Enforcement Point returns the requested Resource. If the decision is negative, a "denied" response is sent back to the Users.

As the primary focus of our prototype is on the middleware layer, and not on the sensor and application layers, the core functionalities of the CAAC middleware layer are implemented. The sensor layer provides the raw contextual data of the user, resource, and interaction environment, which impacts decision making in access control. The application layer receives access requests from the users and passes to the middleware layer for dynamic access control decision making.

8.2 Technology Used to Implement Prototype Components

In this section, we describe the technology used to implement our CAAC middleware layer of the prototype.

We have developed our prototype in the Java 2 Platform Standard Edition (J2SE) [3] using widely supported open source tools. Figure 8.2 shows the technology used to implement the software components shown in the prototype architecture (as shown in Figure 8.1). It includes the following main parts:
1. Ontology KB.

2. Rule Base.

3. Context Manager.

4. Situation Manager.

5. Access Control Manager.

6. Communication Among Software Components.

### 8.2.1 Ontology KB

We have used the Protégé-OWL API [5] to implement the ontology KB for representing and capturing the general and domain-specific concepts, and the relationships among these concepts. In the previous chapters (Chapters 5, 6, and 7), we have already presented the context, situation and policy ontologies, in order to represent the context information,
8.2. Technology Used to Implement Prototype Components

purpose-oriented situation information, and context-aware access control policies. Figure 8.3 shows a snapshot of the context ontology development, where the concepts and the relationships among concepts are described.

![Image of Context Ontology](image)

Figure 8.3: A Snapshot of the Context Ontology

The tree on the left shows the class hierarchy diagram, including the Context Entity and the Context Info hierarchies. The access control related general context entities and context information representing these entities are defined in the Context Entity and Context Info hierarchies respectively. Using these general concepts, we have developed the domain-specific concepts, which are already presented in the context modelling chapter (chapter 5). Figure 8.4 shows a snapshot of the domain-specific role ontology. The tree on the left shows the role hierarchy, including the InDomainUser and the OutDomainUser hierarchies.

157
Chapter 8. Implementation of the CAAC Framework

The *InDomainUser* roles are modeled, relying on the Australian Standard Classification of Occupations (ASCO) of the Health Professionals [1]. For example, the *NursingProfessional*, *MedicalPractitioner*, and *OtherProfessional* roles are the subclasses of *InDomainUser*.

![Figure 8.4: A Snapshot of the Domain-Specific Role Ontology](image)

Similar to the context ontology development, we have used the Protégé-OWL API [5] to implement the situation and policy ontology KB.

### 8.2.2 Rule Base

To derive high-level implicit information from the available information at runtime, we have used the user-defined reasoning rules (OWL and SWRL rules). We have used the
8.2. Technology Used to Implement Prototype Components

Protégé-OWL API [5] to codify the user-defined OWL rules. To infer high-level complex information, like *interpersonal relationships between user and owner and the purpose or user’s intention in accessing the services*, we have used the SWRL rule language [9] and integrated the SWRLTab [7] plug-in to the Protégé-OWL, in order to codify the user-defined SWRL rules. The SWRL rules can directly use OWL knowledge from the ontology.

The SWRLTab editor in Protégé-OWL checks syntax and provides built-in functions for mathematical computations (e.g., add, subtract, power), comparisons (e.g., equal, not equal, greater than, less than or equal), etc. [8]. A reasoning rule is specified in Table 8.1 using SWRL rule language, to derive *interpersonal relationship* between user and resource owner. This rule has used the required context information, the user’s *personal profile* information and the patient’s *social profile* information using our context model. For the application scenario presented in Chapter 2, based on our context ontology, this rule can determine that Jane and Bob has a "nonTreatingPhysician" relationship, because Jane is not assigned as a treating physician of patient Bob.

### Table 8.1: An Example Reasoning Rule

<table>
<thead>
<tr>
<th>No</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule #1</td>
<td>User(?u) ∧ Role(?rol) ∧ hasRole(?u, ?rol) ∧ swrlb:equal(?rol, &quot;GeneralPractitioner&quot;) ∧ Resource(?r) ∧ hasReqRes(?u, ?r) ∧ Owner(?o) ∧ isOwnedBy(?r, ?o) ∧ Relationship(?re) ∧ hasRelationship(?u, ?re) ∧ hasRelationship(?o, ?re) ∧ Person-Centric(?rel) ∧ has(?re, ?rel) ∧ PersonalProfile(?pp) ∧ has(?u, ?pp) ∧ userIdentity(?pp, ?uID) ∧ roleIdentity(?pp, ?rolID) ∧ SocialProfile(?sp) ∧ has(?o, ?sp) ∧ connectedPeopleIdentity(?sp, ?cpID) ∧ connectedPeopleRoleIdentity(?sp, ?cpRID) ∧ swrlb:notEqual(?cpID, ?uID) ∧ swrlb:notEqual(?cpRID, ?rolID) → interRelationship(?rel, &quot;non-TreatingPhysician&quot;)</td>
</tr>
</tbody>
</table>
Chapter 8. Implementation of the CAAC Framework

8.2.3 Context Manager

The Context Manager consists of the Context Providers and the Context Reasoner. We have implemented a number of Context Providers in Java which receive raw context data from the data sources (sensors). For example, an environmental Context Provider (TimeSensor Java class) receives raw context data (the system time is 15:12:32) and generates the low-level context information (the requestTime is 15:12:32) based on the context ontology concepts. Protégé-OWL [5] is not capable of executing SWRL rules. In order to execute SWRL rules, we have implemented the context inference engine, as part of the Context Reasoner. We have used the Jess Rule Engine [4] and Java to implement the Context Reasoner which uses the user-defined reasoning rules and context ontology instances and generates the high-level context information. We have selected the Jess rule engine because of its rule management capability.
8.2. Technology Used to Implement Prototype Components

Figure 8.6 shows two Java functions, the first function executes SWRL rules using Jess rule engine, to infer high-level context information using the basic information from the context ontology, and the second function transfers inferred information in the context ontology.

Figure 8.6: A Part of the Implementation of Our Inference Engine: A Java Code Snapshot in Eclipse

8.2.4 Situation Manager

Similar to Context Manager, we have used both the Jess rule engine [4] and Java to implement the Situation Manager. It uses the user-defined OWL and SWRL rules and ontology instances from the context and situation ontology KB and identifies the purpose-oriented situation information.
8.2.5 Access Control Manager

We have implemented the CAAC Policy Enforcement Point and the CAAC Policy Decision Point in Java, as part of the Access Control Manager. Upon receiving an access request from the user, the CAAC Policy Enforcement Point forwards it to the CAAC Policy Decision Point. The CAAC Policy Decision Point is responsible for the evaluation of access requests. It checks the user’s access request and the relevant context-aware access control policies in the Policy Ontology KB. Then, depending on whether all the necessary contextual information (context and situation information) and the applicable access control policies are satisfied, the CAAC Policy Decision Point determines whether the access request is ‘granted’ or ‘denied’. Finally, the CAAC Policy Enforcement Point enforces the decision of the CAAC Policy Decision Point. If the decision is ‘granted’, i.e., if access to the requested resource is authorized, the CAAC Policy Enforcement Point returns the requested resource to the user, otherwise, a ‘denied’ response is sent to the user. Moreover, in order to check who can access what resources at runtime, we have used the SQWRL [76] as a query language. Specifically, in order to codify the access queries, we have integrated the SQWRLQueryTab [6] plug-in to the Protégé-OWL (as part of the Query Manager).

Figure 8.7 shows a Java function, which executes access query and checks context-aware access control decision.

8.2.6 Communication Among Software Components

We have used Java to realize the communication among various functional components of our prototype framework. The main advantage of Java is its support for interoperability between heterogeneous platforms [46]. The components of our CAAC prototype are designed as independent service components so that they can interact with each other. The implementation as a whole provides an efficient infrastructure support for building access control applications in context-aware environments.
8.3. Application: Patient Medical Records Management (PMRM) System

In this section, we present a context-aware access control application developed for the healthcare domain, in order to illustrate the use of our CAAC framework.

The application, Patient Medical Record Management (PMRM), demonstrates the practical applicability of our framework. This PMRM application is an example of how to realize context-aware access control decisions. The main goal of this application is to retrieve (read) and modify (write) different medical records of patients based on the dynamic contextual information. The application allows different users (e.g., healthcare domain users) to invoke different operations to access specific patient records (i.e., the patients’ medical records at different granularity levels) in a context-aware manner.
Chapter 8. Implementation of the CAAC Framework

In the PMRM application, the healthcare system is equipped with various networked sensor devices such as location sensors, temperature sensors, patient’s body sensor network (BSN), and so on. In our application, we have developed a set of sensors as different context sources by simulating the sensors using Java programs and relational databases. Some of these sensors are SystemTime (which provides current time), Location (which provides location address), User_Pass (a relational table containing usr_id and password), User_Role (a relational table containing usr_id and usr_role_id), Patient_Profile (a relational table containing patient_id, patient_name and connected_people_id), patient Health_Profile (which provides patient_id, heart_rate and body_temperature). Furthermore, using the ontology development capability/tool and the general concepts from the context and situation ontology, we have developed the domain-specific ontologies (in OWL) for the PMRM application (as presented in Chapters 5 and 6). To dynamically infer the implicit information based on the basic available information that has been explicitly captured along with their semantics in the ontologies, we have specified the user-defined rules (OWL and SWRL rules) as shown in Chapters 5 and 6. Moreover, we have developed the access control policy ontology in order to specify the domain-specific context-aware access control policies (presented in Chapter 7) for the PMRM application. Further details of this PMRM application, can be found in the case study of the next chapter (Chapter 9).

8.4 Summary

In this chapter, we have presented a software prototype of the CAAC framework that includes sensor, middleware and application layers. First, the roles of the three layers of the prototype architecture are summarized, then, the implementation details of the software components are described. The prototype architecture offers a new solution in developing context-aware access control applications. We have also developed a context-aware access control application in the healthcare domain using the software prototype. The application allows different healthcare users to access specific patient medical records in a context-aware manner. The prototype and application development serves as an important testbed for the key research contributions of this thesis, and forms the basis for the evaluation of the CAAC framework.
8.4. Summary

In the next chapter, we present the evaluation of the CAAC framework (see Chapter 9) in order to demonstrate the effectiveness of our framework, where we present the case study, performance evaluation and comparative assessments.
Chapter 8 presented a software prototype of our CAAC framework and a context-aware access control application for the healthcare domain developed using the software prototype. In this chapter, we first demonstrate the use of the prototype framework in making context-aware access control decisions through a healthcare case study. Then, we perform a range of experiments, in order to assess the performance of the prototype framework. In addition to the case study and performance evaluation, we present a comparative analysis of our CAAC framework and the related access control frameworks that offers context-aware access control.

The structure of this chapter is as follows. Section 9.1 discusses the patient medical records management application, and its associated contexts, situations and access control policies. Section 9.2 presents a healthcare case study, in order to evaluate the underlying conceptual foundation of our CAAC framework. The experiments that quantify the performance of our framework are discussed in Section 9.3. We then compare our CAAC framework with existing context-aware access control efforts in Section 9.4. Finally, Section 9.5 summarizes the chapter.

9.1 CAAC Application: Patient Medical Records Management

In this section, we discuss the patient medical records management (PMRM) application (presented in previous Chapter 9), covering the CAAC features: contexts, situations, and related access control policies. The PMRM application allows different healthcare users to access appropriate patient medical records in a context-aware manner.
Chapter 9. Evaluation of the CAAC Framework

Our prototype framework determines the access permission on a service request by evaluating the specified CAAC policies, capturing the policy constraints and relevant contexts and situations that are currently in effect. As discussed in Chapter 8, we have implemented the general *CAAC Policy Enforcement Point* and *CAAC Policy Decision Point* in Java. The policy evaluation for this case study (to make context-aware access control decisions) is performed using this prototype.

During the evaluation phase, access queries are used to check the user’s *Access Requests* (i.e., who can access what resources). An access query is formulated using the query language SQWRL [76], which is based on OWL and SWRL.

**Definition 9.1. (AccessRequest)** An *AccessRequest* is defined as a tuple, \(<\text{pass}, \text{service}>\), where ‘pass’ is the user’s access pass (identity, password), and ‘service’ is the (resource, operation) pair.

When an *AccessRequest* comes, the user is first identified based on his provided pass. Then, the policy decision point identifies the applicable context and situation information using the context and situation ontologies. It also identifies the applicable access control policies using the policy ontology. Then, the formulated access query is used to check the user’s request to access the resources using both the policy constraints and currently available context and situation information.

Overall, when an access request comes, our CAAC framework provides the following modeling features for dynamic access control decision making:

- **Context**: our framework models the contextual conditions using its associated context model.
  
  - Modelling and capturing the relevant low-level context information from the raw data.
  
  - Modelling and inferring the relevant high-level context information from the low-level information.
9.2. Case Study: Patient Medical Records Management

- Specifying the contextual conditions using the currently available context information.

- **Situation:** our framework models the contextual conditions using its associated situation model.
  - Modelling and identifying the states of the relevant context entities.
  - Modelling and identifying the states of the relevant relationships between entities.
  - Modelling and inferring the purpose or user’s intention in accessing the services.
  - Specifying the contextual conditions using the inferred purpose and other relevant states.

- **Policy:** our framework models the context-aware access control policies using its associated policy model.
  - Specifying context-aware user-role assignment policies for dynamic associations of users to roles.
  - Specifying context-aware role-permission assignment policies for dynamic associations of roles to permissions.

- **Decision:** our framework provides context-aware access control decision.
  - Matching the current context/situation (contextual conditions) against the constraints of the policies.
  - Sending granted or denied decision to the users.

9.2 Case Study: Patient Medical Records Management

In evaluating the effectiveness of our CAAC framework, we in this section present a case study in the healthcare domain using our prototype framework and patient medical records management application. We have carried out the case study from two different aspects.

- Incorporating the context information in the access control policies.
- Incorporating the situation information in the access control policies.
9.2.1 Incorporating Context Information into CAAC Policy

In this section, we provide two test cases (emergency and normal cases from our application scenario) that highlight specific access control requirements of the PMRM application, by incorporating the context information in the access control policies.

**Test Case I: Emergency Case**

Consider our application scenario presented in Chapter 2 (for Scene #1), when Jane wants to access the service writeEMR(Bob) or readEMR(Bob) (i.e., the write or read access of the emergency medical records of patient Bob), a service AccessRequest is submitted to the CAAC Policy Enforcement Point (which is a part of the Access Control Manager) for evaluation. Jane’s service AccessRequests are shown as follows:

\[
<\text{pass} = (\text{Jane001}, \text{******}), \text{service} = (\text{EMR(Bob)}, \text{write})> \\
<\text{pass} = (\text{Jane001}, \text{******}), \text{service} = (\text{EMR(Bob)}, \text{read})>
\]

The CAAC Policy Enforcement Point forwards the request to the CAAC Policy Decision Point, which checks the relevant context information in the context ontology KB (introduced in Chapter 5) and the applicable access control policies in the policy ontology KB (introduced in Chapter 7).

Our context ontology (introduced in Chapter 5) models and represents the relevant context entities in the scenario (Scene #1). Figure 9.1 shows the entities/instances that are involved in Scene #1. The layer (L3) in Figure 9.1 shows a partial definition of access control-specific core concepts from our context ontology: User, Role, Resource, Owner, Relationship, and Place (classes). The layer (L2) shows a partial definition of domain-specific entities/concepts for the healthcare domain. For example, the core classes Role and Resource of the healthcare domain are instantiated into subclasses emergency doctor (ED) and electronic health records (EHR) respectively. The layer (L1) shows some example instances based on the PMRM application. For example, the instances Jane and Bob are represented as InDomainUser and Patient, the emergency medical records (EMR) is an instance of EHR.
The context ontology captures the relevant low-level context information (from the context providers) (the location address, the user identity and role identity, the connected people identity and connected people role identity, the body temperature and heart rate of the patient, and the resource identity and granularity level). The context ontology also captures the relevant high-level implicit contexts (the health status of patient Bob and the interpersonal relationship between Jane and Bob) inferred using our developed Context Reasoner based on the basic information in the context ontology and the user-defined reasoning rules. A set of user-defined reasoning rules already specified (in Chapter 5) for reasoning about the implicit knowledge conveyed by the specification. Table 9.1 depicts the applicable context information (low-level context and high-level context) characterizing the captured context entities.

The policy ontology captures the relevant policy constraints applicable to the service request, i.e., (resource, operation) pair. The relevant contexts and the policy constraints, captured at the time of access request, are provided to the CAA/C Policy Decision Point as part of the request processing. The current contexts are then matched against the constraints of the policy, as part of making the access control decision.
Table 9.1: Context Information Characterizing the Context Entities in Scene #1

<table>
<thead>
<tr>
<th>Low-Level Context</th>
<th>High-Level Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>locationAddress, userIdentity, roleIdentity, resourceIdentity, granularityLevel, bodyTemperature, heartRate, connectedPeopleIdentity, connectedPeopleRoleIdentity</td>
<td>healthStatus, interRelationship</td>
</tr>
</tbody>
</table>

Based on this information, the **CAAC Policy Decision Point** determines whether the request is “granted” or “denied” for the submitted access request, and returns the decision to the **CAAC Policy Enforcement Point**. Finally, the **CAAC Policy Enforcement Point** enforces the decision. If the decision is “granted”, the requested resource is sent to the user; otherwise, a “denied” response is sent to the user.

Table 9.2: An Example Access Query

```
CAACPolicy(?caac_Jane_EMREMR(Bob)) \ User(?Jane) \ Role(?role) \ ContextualCondition(?context) \ Permission(?EMR(Bob)) \ AccessDecision(?decision) 
\rightarrow sqwrl:select(?user, ?role, ?context, ?permission, ?decision)
```

Table 9.3: Access Query Results

<table>
<thead>
<tr>
<th>?user</th>
<th>?role</th>
<th>?context</th>
<th>?permission</th>
<th>?decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>ED</td>
<td>ContextualCondition1 _ED</td>
<td>EMR(Bob)_read</td>
<td>granted</td>
</tr>
<tr>
<td>Jane</td>
<td>ED</td>
<td>ContextualCondition1 _ED</td>
<td>EMR(Bob)_write</td>
<td>granted</td>
</tr>
</tbody>
</table>

For the application example (Scene #1 in the motivating scenario), the defined access query (see an access query in Table 9.2) is used to check whether the access request is *granted* or *denied*. The entries in the access query results (Lines #1 and #2 of Table 9.3) satisfy Jane’s *AccessRequests* mentioned above (i.e., Jane’s service access permissions (“EMR_write” and “EMR_read”) are granted). The policy specified in Chapter 7 shown that *if* Jane and Bob both are *located* in the “emergency room” of the hospital *and*
9.2. Case Study: Patient Medical Records Management

Bob’s current health status is “critical” and the interpersonal relationship between Jane and Bob is “treating physician” or “non-treating physician”, then Jane is authorized to access the service writeEMR(Bob) or readEMR(Bob), because the available contexts and policy constraints indicate that the access conditions are satisfied. Note that, a contextual condition ContextualCondition1_ED (shown in Table 9.3) is modeled/captured in our context ontology, based on the available contextual information (mentioned in Table 9.1). Figure 9.2 presents a screenshot of our prototype implementation that shows the access control decision for Jane’s access request (the write access to the emergency medical records of patient Bob). The above case study for the test scenario shows that our framework is able to successfully make context-aware access control decisions.

Figure 9.2: A Screenshot of Our Prototype
Chapter 9. Evaluation of the CAAC Framework

In this scenario (Scene #1), Jane is not a treating physician of patient Bob, but she is allowed to access Bob’s EMR (at granularity level 0), i.e., including all other sub-components of the EMR (at the lower granularity levels), in such a critical situation. On the other hand, when the context changes (e.g., Jane leaves the emergency room, or Bob’s health condition changes from critical to normal), the system will not grant Jane the access to the requested service.

Test Case II: Normal Case

For Scene #2 in our application scenario, where a registered nurse Mary wants to access Bob’s daily medical records (DMR) and his past medical history (PMH), an access request is submitted to the CAAC Policy Enforcement Point for evaluation.

Mary’s service AccessRequests are shown as follows:

\[
\begin{align*}
\text{pass} = & (\text{Mary001, ******}, \text{service} = (\text{DMR(Bob), read})) \\
\text{pass} = & (\text{Mary001, ******}, \text{service} = (\text{DMR(Bob), write})) \\
\text{pass} = & (\text{Mary001, ******}, \text{service} = (\text{PMH(Bob), read})) \\
\text{pass} = & (\text{Mary001, ******}, \text{service} = (\text{PMH(Bob), write}))
\end{align*}
\]

Our context ontology models and represents the relevant context entities in the scenario (Scene #2). Figure 9.3 shows the entities/instances are involved in Scene #2. The layer (L3) shows the core classes: User, Role, Resource, Owner, Relationship and Place. The layer (L2) shows the domain-specific concepts. For example, the core classes Role and Resource of the healthcare domain are instantiated into subclasses registered nurse (RN) and electronic health records (EHR) respectively. The layer (L1) shows some example instances based on the PMRM application: for example the instances Mary and Nurse_1 are represented as InDomainUser and registered nurse (RN), the daily medical records (DMR) and past medical history (PMH) are the instances of EHR.

The context ontology captures the relevant low-level and high-level context information. Table 9.4 depicts the relevant context information characterizing the captured context entities in Scene #2. Two contextual conditions ContextualCondition1_RN (shown in
9.2. Case Study: Patient Medical Records Management

![Context Ontology Diagram](image)

Figure 9.3: An Excerpt of Context Ontology (Context Entities in Scene #2)

Table 9.6) and ContextualCondition2_RN (shown in Table 9.8) are modeled/captured in our context ontology, based on this applicable contextual information.

Table 9.4: Context Information Characterizing the Context Entities in Scene #2

<table>
<thead>
<tr>
<th>Low-Level Context</th>
<th>High-Level Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>locationAddress, startTime, endTime, userIdentity, roleIdentity, resourceIdentity, granularityLevel</td>
<td>requestTime, isColocatedWith</td>
</tr>
</tbody>
</table>

The relevant context information captured in the context ontology and the relevant policy specified in the policy ontology are provided to the CAAC Policy Decision Point as part of the access request processing. Finally, the CAAC Policy Enforcement Point returns the access control decision, i.e., Mary’s access request is “granted”, because it satisfies the policies which are stored in the policy knowledge-base. We can observe that if Mary is located in the general ward during her ward shift time, she is authorized to access the
Chapter 9. Evaluation of the CAAC Framework

DMR (at granularity level 1) of patient Bob, who is hosted in that ward in his normal health condition.

For the application scenario (Scene #2), the access query (see an access query in Table 9.5) is used to check whether the access request is "granted" or "denied". Lines #1 and #2 of Table 9.6 satisfy Mary’s AccessRequests mentioned above (i.e., Mary’s service access permissions (“DMR_write” and “DMR_read”) are granted).

Table 9.5: An Example Access Query

| CAACPpolicy(?caac_Mary_DMR(Bob)) ∧ User(?Mary) ∧ Role(?role) ∧ ContextualCondition(?context) ∧ Permission(?DMR(Bob)) ∧ AccessDecision(?decision) → sqwrl:select(?user, ?role, ?context, ?permission, ?decision) |

Table 9.6: Access Query Results

<table>
<thead>
<tr>
<th>?user</th>
<th>?role</th>
<th>?context</th>
<th>?permission</th>
<th>?decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>RN</td>
<td>ContextualCondition1_RN</td>
<td>DMR(Bob)_read</td>
<td>granted</td>
</tr>
<tr>
<td>Mary</td>
<td>RN</td>
<td>ContextualCondition1_RN</td>
<td>DMR(Bob)_write</td>
<td>granted</td>
</tr>
</tbody>
</table>

Mary is also authorized to access Bob’s past medical history (PMH) when Jane (who is a general practitioner) is present with her (see an access query in Table 9.7 and the query results are shown in Lines #1 and #2 of Table 9.8); conversely, a registered nurse alone has no right to access a patient’s past medical history. However, the system will not grant Mary access to Bob’s EMR at granularity level 0 as there is no such policy.

Table 9.7: An Example Access Query

| CAACPpolicy(?caac_Mary_PMH(Bob)) ∧ User(?Mary) ∧ Role(?role) ∧ ContextualCondition(?context) ∧ Permission(?PMH(Bob)) ∧ AccessDecision(?decision) → sqwrl:select(?user, ?role, ?context, ?permission, ?decision) |

176
9.2. Case Study: Patient Medical Records Management

Table 9.8: Access Query Results

<table>
<thead>
<tr>
<th>?user</th>
<th>?role</th>
<th>?context</th>
<th>?permission</th>
<th>?decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>RN</td>
<td>ContextualCondition2_RN</td>
<td>PMH(Bob)_read</td>
<td>granted</td>
</tr>
<tr>
<td>Mary</td>
<td>RN</td>
<td>ContextualCondition2_RN</td>
<td>PMH(Bob)_write</td>
<td>granted</td>
</tr>
</tbody>
</table>

9.2.2 Incorporating Situation Information into CAAC Policy

In the previous section, we have shown that our prototype framework makes access control decisions by incorporating the context information into access control process. Our framework also makes the access control decisions with purpose-oriented situation information. In this section, we provide a test case from our application scenario that highlights specific access control requirements of the PMRM application, by incorporating the situation information into access control process.

Test Case I: Emergency Case

Consider the same application scenario (Scene #1), where Jane, by playing an emergency doctor role, wants to access the requested service writeEMR(Bob) (i.e., the write access to the emergency medical records of patient Bob). Our context ontology (introduced in Chapter 5) captures and reasons about the relevant low-level and high-level context information (interpersonal relationship, location address, health status, etc.). Using the context ontology, our situation ontology (introduced in Chapter 6) identifies the relevant situation based on the captured context information and situation specification rules. For the PMRM application we have specified different types of situations. A relevant ‘situation’ (regarding Scene #1) and its associated ‘context information’ and ‘purpose’ using situation reasoning rules are shown in Table 9.9.

In order to identify the relevant situations, our situation model first identifies the user's intention or purpose of resource access based on the relevant context information, and then identifies the relevant states using currently applicable context information from the context model. This inferred purpose and the states based on the context information are used to infer the situation.
Chapter 9. Evaluation of the CAAC Framework

Table 9.9: Definition of Situation

<table>
<thead>
<tr>
<th>Situation</th>
<th>Situation Definition (High-Level Description)</th>
</tr>
</thead>
<tbody>
<tr>
<td>An emergency doctor from the hospital for emergency treatment (ED-FromHospitalForET)</td>
<td>User_Role(EmergencyDoctor(ED)) \∧ \ Location_ED(Hospital) \∧ \ Relationship(ED, Patient)(nonTreatingPhysician) \∧ \ Purpose(EmergencyTreatment(ET))</td>
</tr>
</tbody>
</table>

The relevant situation information (identified in the situation ontology) and policy (specified in the policy ontology) are provided to the CAAC Policy Decision Point as part of the request processing. Based on this information, the CAAC Policy Decision Point determines whether the request is “granted” or “denied” for the submitted access request, and returns the decision to the CAAC Policy Enforcement Point. Finally, the CAAC Policy Enforcement Point enforces the decision and returns the decision, i.e., Jane’s service access request is granted (see an access query in Table 9.10 and result in Table 9.11), because the situation ontology identifies relevant situation, and satisfies a policy which is stored in the policy knowledge base (policy ontology).

Table 9.10: An Example Access Query

\[
\text{CAACPolicy(?caac\_Jane\_EMR(Bob)) } \∧ \text{ User(?Jane) } \∧ \text{ Role(?role) } \∧ \text{ ContextualCondition(?situation) } \∧ \text{ Permission(?EMR(Bob)) } \∧ \text{ AccessDecision(?decision)} \rightarrow \text{ sqwrl:select(?user, ?role, ?situation, ?permission, ?decision)}
\]

Table 9.11: Access Query Result

<table>
<thead>
<tr>
<th>?user</th>
<th>?role</th>
<th>?situation</th>
<th>?permission</th>
<th>?decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>ED</td>
<td>EDFromHospitalForET</td>
<td>EMR(Bob)_write</td>
<td>granted</td>
</tr>
</tbody>
</table>

In general, when a new access request comes or a context/situation changes, the CAAC Policy Enforcement Point sends automated request to the CAAC Policy Decision Point for
9.3. Performance Evaluation

the relevant policies and (re-)evaluates the context-aware access control decisions according
to the relevant context and situation information. The above case study explores the use
of our CAAC framework and its theoretical underpinnings for the development of the
context-aware access control applications. It yields a variety of access control decisions in
a context-aware manner.

9.3 Performance Evaluation

In addition to the case study, we assess the effectiveness of our CAAC framework, where
we measure the query response time to provide resource access permissions to users. The
main purpose of our experimental investigation is to quantify the performance overhead
of our CAAC framework. We have conducted two sets of experiments with our prototype
framework as applied to the PMRM application.

- Experimental Setting I: incorporating the context information in the access control
  policies.
- Experimental Setting II: incorporating the situation information in the access control
  policies.

9.3.1 Experimental Setting I: Incorporating Context Information into
CAAC Policy

With the goal of evaluating the runtime performance of the various stages of our prototype
framework, in this first setting we incorporate the relevant context information in the
access control policies. Towards this end, we have conducted four experimental tests on
a Windows XP Professional operating system running on Intel(R) Core(TM) Duo T2450
@ 2.00 GHz processor with 1GB of memory. Using our prototype framework and PMRM
application, we have measured the end-to-end response time (T) of the different aspects of
access request processing, i.e., the time taken from the arrival of the user's service access
request to the end of its execution (the sum of context information processing time and
access control decision making time).
We separate the time delay for loading the context ontology and policy ontology from the access request processing time. The ontology loading occurs when the system runs the first time and when the access policy changes. In general, the most important parameter here is the access request processing time, which is of more interest or significance than the ontology loading time (context and policy ontologies) on an ongoing basis. The following are the time breakdowns of the various processing stages.

- Total response time \((T)\) - time taken
  - for processing context information - time taken
    * to capture the raw context facts from different data sources (fetching raw data from the sensors using context providers),
    * to extract the low-level context information from the raw data,
    * to deliver and save the low-level information to the context ontology knowledge-base (KB),
    * to select the appropriate user-defined rules from the SWRL rule-base and data/information from the context KB for deriving high-level context information,
    * to infer the high-level context information by executing the selected SWRL rules (using context reasoner), and
    * to deliver and save this high-level information to the context KB.
  - for making access control decision - time taken
    * to select the appropriate access control policies from the policy ontology, and
    * to execute/evaluate the relevant policies in order to make access control decision.

Experimental Results and Analysis

The objective of these four experimental tests is to conduct a quantitative feasibility study to measure the end-to-end response time \((T)\) with respect to the increasing number of
context information types or access control policy rules (overall, increasing the ontology KB size from small to big).

**Test 1.** The first test focuses on measuring the response time ($T$) of our prototype in the light of increasing the number of context types (e.g., the identity of the user, the granularity level of the requested resource, the interpersonal relationship between the user and the owner, the health status of the owner, and the personal status of the user). We have selected 10 fixed number of policy rules, which are attached to 10 different health professionals roles: GeneralPractitioner, RegisteredNurse, etc. The number of context information types ($\#CT$) is varied from 3 to 10. In step $\#CT = 3$ in Figure 9.4, we first codify 10 access control policy rules in which 3 context information types act as the policy constraints. In such a manner the number of context information types is varied in increment of one. There are 10 execution runs for each set of the context types, to measure the end-to-end response time. The average value of the 10 execution runs is used for the analysis.
The test results in Figure 9.4 show that the average response time ($T$) in changing the number of context types varies from 2.7 to 3.5 seconds approximately. We can see that the response time seems to be linear, when the ontology KB (the context and policy ontologies) size is small (44.2 to 53.4 Kilobytes).

**Test 2.** In this test, we have evaluated the response time when the number of access control policies increased. First, we have selected 10 policy rules with respect to 10 different health professional roles. We have varied the number of policies up to 50 with 50 different roles. Each of these variations is executed 10 times for each of following cases: 3 CT (context types), 5 CT, and 10 CT. The average value of the 10 execution runs is used for the analysis (see Figure 9.5).

The test results in Figure 9.5 show that the average response time ($T$) increases when the number of context types and policies increases. For example, it varies between 3.5 and 4.4 seconds for 10 types of context information in every policy. Also, similar to test 1, in Figure 9.5 we can see that when the ontology KB size is small (44.2 to 293 Kilobytes) the response time seems to be linear.

---

Figure 9.5: Average Response Time over Different Numbers of Context-Aware Policies
9.3. Performance Evaluation

**Figure 9.6: Average Response Time over Various Size of the Ontology KB**

**Test 3.** In this test, we have again evaluated the end-to-end response time (T) over various sizes of the ontology KB. We have varied the number of policies up to 500 with respect to 138 different health professionals [1] (i.e., 138 roles). In order to build the ontology KB of increasing sizes, we have specified 2000 rules. To measure the response time, we have run each experiment 10 times and the average value of the 10 execution runs is used for the analysis (see Figure 9.6).

The test results in Figure 9.6 show the average response time over various sizes of ontology KB (the context and policy ontologies). The computational overhead increases at a linear rate, up to 1250 kilobytes KB size and gets higher after that (note that the scale is not linear in the horizontal axis). As the size of the KB increases beyond 1250 kilobyte the response time increases dramatically. This is due to the large number of policy rules (larger KB size), fully utilizing the memory capacity of the computer. At the point of the KB size being 1250 kilobytes (where we have specified 500 policies for the 138 health professional roles), it takes approximately 10 seconds to process the request. We consider this as acceptable for this type of context-aware access control application.
Figure 9.7: Ontology Loading Time over Different Numbers of Context-Aware Policies

**Test 4.** This test focuses on measuring the ontology loading time (the context and policy ontologies) of our prototype when the size of the KB increases.

The test results in Figure 9.7 show that the ontology loading time in increasing the size of the KB varies from 10.02 to 32.31 seconds approximately. Note that the scale is not linear in the horizontal axis. This results do not have impact on the end-to-end response time ($T$) as ontology loading is only done at the start of the system and when access policy changes.

In general, the end-to-end response time increases with the increasing number of context types and policy rules, but not dramatically until the KB size is greater than 1250 kilobytes (including 138 health professional roles and 500 policy rules).

### 9.3.2 Experimental Setting II: Incorporating Situation Information into CAAC Policy

We have conducted a second set of experiments in the same setup: on a Windows XP Professional operating system running on Intel(R) Core(TM) Duo T2450 @ 2.00 GHz
9.3. Performance Evaluation

processor with 1GB of memory. In this second setting, we incorporate the relevant situation information in the access control policies. Towards this end, we have conducted two experimental tests.

The main purpose of this experimental investigation is to quantify the performance overhead of our prototype including situations. Our main measures in this experimental setting include:

- situation identification time, and
- policy evaluation time.

The first measure indicates how long it took to identify/infer a relevant situation (by capturing the currently available context information and identifying the purpose using this information). The second measure indicates how long it took to determine a user’s access permission on a requested resource and service (by incorporating the identified situation into the access control process and making context-aware access control decision). We calculate the average end-to-end response time \( T \), time from the arrival of the user’s service access request (query) to the end of its execution, including the time for identifying/inferring relevant situation and time for evaluating relevant policy. The following are the time breakdowns of the various processing stages.

- End-to-end response time \( T \) - time taken
  - for inferring situation information - time taken
    * to capture the low-level context information from the raw data,
    * to infer the high-level context information from the low-level information,
    * to identify the relevant states using the relevant context information,
    * to identify the purpose or user’s intention in accessing the services,
    * to select the appropriate user-defined rules from the ontology/rule-base, and
    * to infer the relevant situation information executing the defined rules.
  - for evaluating relevant access control policy - time taken
Figure 9.8: Average Response Time Over Different Numbers of Access Control Policies

* to select the appropriate access control policies from the policy ontology, and
* to execute/evaluate the relevant policies in order to make situation-aware access control decision.

Experimental Results and Analysis

With the goal of evaluating the runtime performance of our prototype, the main finding was that the time for making context-aware access control decision (based on the situation identification time and policy evaluation time) is acceptable, as they impose just a small overhead.

Test 1. This test focuses on measuring the response time of our prototype in the light of increasing number of access control policies. First, we have selected 20 policy rules in which 5 situation types (ST) act as the policy constraints (e.g., the policy rules for emergency
9.3. Performance Evaluation

Test-2 Results (up to 2000 policy rules)

Figure 9.9: Average Response Time Over Different Size of the Ontology KB

doctor to access patients medical records for emergency treatment purpose shown in Tables 7.3 and 7.4 in Chapter 7). We have varied the number of policy rules up to 100 with 15 different types of situation variations. Each of these variations is executed 10 times for each of following cases: 5 ST (situation types), 10 ST, and 15 ST. For each setting, the average value of the 10 execution runs is used for the analysis (see test results in Figure 9.8). The test results show that the average response time increases when the number of situation types and policy rules increases. For example, the response time varies between 4.1 and 5.2 seconds for 15 types of situation information and for the variation of 20 to 100 policy rules. We can see that the average response time seems to be linear. Overall, we consider that the performance is acceptable.

Test 2. In this test, we have again evaluated the total response time (situation identification and policy evaluation time) over various sizes of the knowledge base. We have varied the number of policies up to 500 with respect to 138 different health professionals [1] (i.e., 138 roles). To build the ontology KB of increasing sizes, we have specified 2000 policies. In order to measure the response time, we have run the experiment 10 times and the
average value of the 10 execution runs is used for the analysis (see test results in Figure 9.9). As the size of the ontology KB increases beyond 1618 kilobyte, the computational overhead increases dramatically (note that the scale is not linear in the horizontal axis). This is due to the large number of policy rules (larger KB size), fully utilizing the memory capacity of the computer. At the point of the KB being 1618 kilobyte (500 policy rules for the 138 health professionals), it takes approximately 10 seconds to process the request. We can see that changes to the number of access control policies do not have much impact on the response time, when the ontology knowledge base (KB) size is small. Overall, the runtime performance is acceptable for a reasonable sized KB (up to 1618 Kilobytes size).

Similar to previous set of experimental tests (where we incorporated context information in the CAAC policies), in this set of experimental tests we incorporate situation information in the CAAC policies. The main measure in this set of experiments included (i) situation identification and (ii) policy evaluation. Though both sets of experiments follow similar nature, the situation inference is all about identifying and inferring the relevant states of the context entities and the purpose of resource access. As such, in this experimental setting, the ontology KB size is a little bit bigger than previous setting and consequently the computational overhead increases a little bit higher than previous setting (see Figures 9.6 and 9.9).

9.3.3 Overall Discussion

We have conducted two sets of experimental tests and measured the computational overhead, in order to evaluate the runtime performance of the various stages of our prototype framework. Towards this goal, we have incorporated the relevant context and situation information in the access control policies.

In general, in these experiments, the response time increases with the following variations: in the light of increasing the number of context types and situations types in the access control policies and also with the increasing number of access control policy rules. However, the response time does not increase dramatically when the KB size is small (44.2 to 293 kilobytes, shown in Figure 9.5 and 92 to 687 Kilobytes size, shown
9.4. Comparative Assessment

in Figure 9.8). The computational overhead increases at a linear rate with different variations (of increasing context types, situation types and policy rules). However, it increases dramatically when the size of the KB becomes big (beyond 1250 kilobytes size, shown in Figure 9.6 and beyond 1618 kilobytes size, shown in Figure 9.9).

Although the runtime performance seems a little expensive, this is acceptable for a reasonable sized knowledge base and application. It can be improved by employing better software tools and hardwares. Overall, the experimental results have shown that our access control framework has good response time granting user’s access request to resources and services in a context-aware manner.

9.4 Comparative Assessment

In addition to the case study and runtime performance evaluation, we have also conducted a comparative assessment of our CAAC framework and the related access control frameworks analysed in Chapter 3. First we present the key features of our CAAC framework and then we analyse the existing frameworks in terms of these features.

9.4.1 Key Features of Our CAAC Framework

Our context-aware access control framework provides the following key features.

- **Access control-specific context entities and information.** Our CAAC framework provides a Context Model (CM) to represent and capture access control-specific context entities and context information characterizing these entities.
  - Capturing the broad categories of access control-specific context entities (CE): user, role, resource, owner, relationship, and environmental entities.
  - Capturing and reason about context information characterizing the entities (EC): status, profile, spatial, temporal, and historical information.
  - Capturing and reasoning about context information characterizing the relationship between entities (RC): different types of relationship information.
Chapter 9. Evaluation of the CAAC Framework

- **Purpose-oriented situation information.** Our CAAC framework provides a Situation Model (SM) to identify relevant atomic situations and reason about composite situations. The situation model uses the context information from the context model in order to identify the elementary information.
  
  - Identifying and inferring the states of the context entities (ES).
  - Identifying and inferring the states of the relationships between entities (RS).
  - Identifying and inferring the purpose or user's intention in accessing the resources (UI).

- **Context-Aware User-Role Assignment.** Our policy model supports the specification of Context-Aware User-Role Assignment (CAURA) for dynamic associations of users to roles.

- **Context-Aware Role-Permission Assignment.** Our policy model supports the specification of Context-Aware Role-Permission Assignment (CARPA) for dynamic associations of roles to permissions (resource access permissions).

### 9.4.2 Comparative Analysis

We compare our framework with existing access control frameworks based on the key features specified in the previous section. We conduct a comparative assessment of our framework in terms of these features.

In general, we distinguish three different categories of access control frameworks for this analysis. Firstly, the access control frameworks that incorporate different types of context information into role-based access control models. Secondly, the access control frameworks that integrate context information into attribute-based access control models. Thirdly, the access control frameworks that incorporate situation information into access control models.
Table 9.12: Comparative Analysis between Our Context-Aware Access Control Framework and the Related Access Control Frameworks

<table>
<thead>
<tr>
<th>Access Control Frameworks</th>
<th>CM</th>
<th>SM</th>
<th>CAURA</th>
<th>CARPA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CE</td>
<td>EC</td>
<td>RC</td>
<td>ES</td>
</tr>
<tr>
<td><strong>Frameworks to Incorporate Context Information into Role-Based Access Control</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRBAC: Temporal RBAC [18]</td>
<td>–</td>
<td>Δ</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Generalized TRBAC [55]</td>
<td>–</td>
<td>Δ</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>GEO-RBAC: Spatially Aware RBAC [19]</td>
<td>–</td>
<td>Δ</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LoT-RBAC: Location and Time-Based RBAC [25]</td>
<td>–</td>
<td>Δ</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CA-RBAC: Context-Aware RBAC [68]</td>
<td>Δ</td>
<td>Δ</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Semantics-Based Access Control [48]</td>
<td>Δ</td>
<td>Δ</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Context-Aware RBAC [87]</td>
<td>Δ</td>
<td>Δ</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Frameworks to Incorporate Context Information into Attribute-Based Access Control</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UbiCOSM: Context-Based Middleware [28]</td>
<td>Δ</td>
<td>Δ</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CSAC: Context-Sensitive Access Control [54]</td>
<td>–</td>
<td>Δ</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Semantic Context-Aware Access Control [90]</td>
<td>Δ</td>
<td>Δ</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Context-Aware Access Control [29]</td>
<td>Δ</td>
<td>Δ</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Frameworks to Incorporate Situation Information into Access Control</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA-RBAC: Situation-Aware RBAC [65]</td>
<td>Δ</td>
<td>Δ</td>
<td>–</td>
<td>Δ</td>
</tr>
<tr>
<td>SA-AC: Situation-Aware Access Control [99]</td>
<td>Δ</td>
<td>Δ</td>
<td>–</td>
<td>Δ</td>
</tr>
<tr>
<td>SAW: Situation-Awareness Model [97]</td>
<td>Δ</td>
<td>Δ</td>
<td>–</td>
<td>Δ</td>
</tr>
<tr>
<td><strong>Frameworks to Incorporate Context and Situation Information into Access Control</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAAC: Our Framework [59, 62, 60, 61, 63]</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>
Table 9.12 shows a comparative analysis of our context-aware access control framework with existing access control frameworks. With respect to our CAAC framework features specified in Section 9.4.1, we use a √ if a related framework provides similar and/or comparable support for a certain aspect, a △ if a related framework provides partial support for a particular aspect, and a – if a related framework provides no support for a particular aspect.

Several existing access control approaches have extended the basic role-based access control model (RBAC) [84] by incorporating some specific types of contextual information into access control processes: temporal information (e.g., [18, 55]), spatial information (e.g., [19]), and both time and location (e.g., [25]). A number of recent approaches (e.g., [68, 48, 87]) have also adopted the RBAC model for context-specific access control. These approaches usually focus on the integration of several further context information other than location and time, such as user attributes. Different from these approaches, our CAAC framework considers a wide range of access control-specific context information characterizing the relevant entities. In addition, our framework also considers the context information characterizing the relationships between entities. Moreover, these extended RBAC approaches incorporate the spatial and temporal information to specify the user-role and role-permission assignment policies. Different from them, our framework considers a broader range of contextual dimensions for the specification of context-aware user-role assignment and role-permission assignment policies.

On the other hand, some access control approaches have integrated different types of context information and extended the attribute-based access control (ABAC) model [92]. Only certain types of contextual aspects, such as environment (location and time), resource and user attributes are considered in these approaches (e.g., [28, 54, 90, 29]). These approaches incorporate this contextual information in the access control policies as attributes. Different from these approaches, our CAAC framework considers relationship dimension and context inference. Besides, the attribute-based approaches, which grant accesses to resources and services based on the attributes of entities (e.g., user's identity - the attribute possessed by the user/requester) rather than direct role-based support,
9.4. Comparative Assessment

do not have adequate functionality to specify context-aware user-role assignment and role-permission assignment policies. Whereas, our CAAC framework provides this role-based support.

Some other access control approaches describe situation as the specific combinations of certain contextual information (e.g., [97]). Kim and Lim propose the situation-aware RBAC (SA-RBAC) approach, which extends the basic RBAC model and dynamically grants roles (or permissions) to users based on the situation information of the user [65]. Yau and Liu have presented a situation-aware access control (SA-AC) based privacy-preserving service matchmaking approach [99]. The SA-AC approach incorporates situation-aware constraints into RBAC model that can affect the access control decisions, such that the states of service providers, requesters and environments. In order to specify situation, these approaches consider the states of the relevant entities. Different from these approaches, our CAAC framework also considers the states of the relevant relationships between entities for situation specification. None of these approaches consider the purpose or user’s intention in accessing resources and services, whereas our framework considers this.

In general, the existing context-specific approaches to access control have only considered specific types of context information, each of them having different origins, pursuing different goals and often, by nature, being highly domain-specific. The situation-specific access control approaches consider the specific combinations of certain contextual information as constraints in the access control policies. However, they do not consider the purpose or user’s intention in accessing the resources, which is another important aspect. To the best of our knowledge, this thesis presents the first attempt to dynamically capture user’s intention in accessing resources at runtime through purpose-oriented situations. Overall, access control by considering a wide range of dynamic context and purpose-oriented situation information with reasoning capabilities is largely missing in the literature. As such, in this thesis we have introduced a context-aware access control framework for software services [59, 62, 60, 61, 63]. It includes (i) a general and extensible context model specific to access control to capture and reason about contextual informa-
Chapter 9. Evaluation of the CAAC Framework

tion, (ii) a situation model to identify and infer purpose-oriented situation information, and (iii) a context-aware policy model incorporating context and situation information into access control process, in order to specify context-aware user-role assignment and role-permission assignment policies.

9.5 Summary

In this chapter, we have evaluated our CAAC framework. First, we have presented a case study in the healthcare domain in the area of patient medical records management (PMRM). The case study has shown that our framework is effective and controls access permissions to information resources in a context-aware manner.

We have also conducted a range of experiments in a simulated environment using the PMRM application. The experimental results have shown that the prototype implementation of our framework offers satisfactory system performance.

Finally, we have compared our CAAC framework with existing access control frameworks. The comparative analysis has shown that our framework offers a range of new capabilities (such as purpose-oriented situation) for context-aware access control compared with other related access control frameworks.
The aim of this thesis is to address the challenges associated with the engineering of context-aware access control applications, by developing a new access control framework for dynamically changing environments. Towards this end, in this thesis, we have first started with an application scenario from the healthcare domain that illustrates the need for the incorporation of dynamic contextual information in the access control process. Then, we have analyzed the existing works that are related to our research. On the basis of analyzing the limitations of existing related research and the general requirements of developing context-aware access control applications, we have introduced a novel framework for context-aware access control. The framework incorporates context-awareness in the access control process and controls access permissions to information resources and software services in a context-aware manner.

In this chapter, we first summarize the major contributions made in this thesis, and then discuss some interesting research directions which may be explored further in the future.

10.1 Contributions

In this thesis, we have introduced a new context-aware access control framework for software services. It includes a general and extensible context model to represent and reason about access control-specific contexts, a situation model to identify purpose-oriented situations, and an access control policy model incorporating context and situation information
in the access control process. This thesis makes a number of research contributions regarding the design and implementation of this novel *Context-Aware Access Control Framework (CAAC)* for software services.

- **Context-Aware Access Control Approach**

  For our framework, we have introduced a novel context-aware access control approach, which adopts and extends the basic role-based access control (RBAC) approach with dynamically changing contextual information. The approach has introduced the dynamic context-awareness capabilities into both user-role and role-permission assignments. We have clarified the notion of context and purpose-oriented situation and laid the foundation for the CAAC framework by proposing a conceptual context-aware access control approach. Our access control approach significantly differs from the existing access control approaches in that it integrates the dynamic context and situation information in the access control process and considers the context-aware user-role assignment and context-aware role-permission assignment policies.

- **Access Control-Specific Context Model**

  For capturing the different types of context information relevant to access control, we have introduced an ontology-based context model for our CAAC framework. Ontology-based approach has been evaluated as the most widely used modelling technique for context/situation modelling because of its considered trade-off between computational complexity of reasoning and expressiveness. We define context as any relevant information representing the context entities and the relationship between context entities. In our context model, first we have captured the broad categories of context entities capable of providing context information and identified categories of basic context information specific to access control. Then, the context model has been extended with user-defined rules to reason about high-level implicit context information that is not directly available but can be derived from available (basic) context information. Our context model significantly differs from the existing context models in that it considers the relevant access control-specific entities (user,
role, resource, owner, relationship, and environmental entities), represents and
models a wide range of important dynamic context information characterizing these
entities within a single framework, and allows context information representation
and derivation at different levels of abstraction by inferring high-level implicit
context (i.e., more abstract and richer context information can be derived by using
our context model).

- **Purpose-Oriented Situation Model**

  We have introduced an ontology-based situation model to represent the purpose-
  oriented dynamic situations. We have defined situation as the correlation of user's
  intention in accessing resources and some selective context information concerning
  that intention, and categorized two types of situations, atomic situations and
  composite situations. In order to identify relevant situations, our situation model
  uses the relevant elementary information from the context model (in order to
  identify the relevant states and the user's intention in accessing the services). One
  of the main advantages of our situation model is its purpose-oriented control over
  access to information resources. It also has the reasoning capability, i.e., once the
  basic or atomic situations are captured, the composite situations can be inferred
  using an inference engine through user-defined reasoning rules. Our situation
  model significantly differs from the existing situation models in that it considers
  the states of the entities, the states of the relationships between entities, and the
  purpose-oriented information, whereas the existing situation models consider only
  the states of the relevant context entities.

- **Context-Aware Policy Model**

  We have introduced an ontology-based policy model for our CAAC framework. The
  policy model supports the specification of context-aware user-role and role-permission
  assignment policies for context-aware access control applications. It uses the dy-
  namic information from the context and situation models, and allows the users to
  access information resources of different granularities depending upon the dynam-
ically changing context and situation information. Our policy model significantly
differs from the existing policy models in that it considers the context-aware access
control policies in terms of dynamic context-aware user-role assignment and role-
permission assignment. Whereas, the existing access control policy models do not
provide adequate functionality to incorporate the context-awareness capabilities into
both user-role assignment and role-permission assignment policies.

In addition to the above four major contributions, we have also implemented and evaluated
our context-aware access control framework.

- **Implementation of Context-Aware Access Control Framework**

In order to demonstrate the practical applicability of our framework, we have
presented a software prototype of our CAAC framework. We have also developed
a context-aware access control application in the healthcare domain in the area of
patient medical records management (PMRM) using the software prototype. We
have clearly demonstrated how our prototype framework can be used to build the
PMRM application. The application controls different healthcare professionals'
access to patient medical records of different granularities in a context-aware manner.
Overall, our prototype framework offers a new solution in developing context-aware
access control applications.

- **Evaluation of Context-Aware Access Control Framework**

In order to evaluate the underlying conceptual foundation of our CAAC framework,
we have carried out a case study in the healthcare domain. The case study has
shown that our framework is effective and controls access permissions to information
resources in a context-aware manner. To assess the effectiveness of the framework, we
have performed a range of experiments. The experimental results have shown that the
prototype implementation of our framework offers satisfactory system performance.
In addition to the case study and performance evaluation, we have compared our
CAAC framework with existing access control frameworks. The comparative analysis
has shown that our CAAC framework offers a range of new capabilities for context-
aware access control.
10.2. Future Work

In summary, our CAAC framework addresses the key requirements/issues of developing context-aware access control applications, i.e., understanding and identifying the context and situation information and associated context entities relevant to access control; inferring richer and more complex context and situation information according to user-defined rules; and defining and enforcing access control policies that use the context and situation information, to provide context-aware resource access permission to users. In this thesis, we have presented how these key issues can be addressed by our novel CAAC framework that includes techniques for modelling and reasoning about context and situation, incorporation of this information into the access control process, and a software prototype for supporting the development of context-aware access control applications. These research contributions improve the access control in dynamically changing context-aware environments.

10.2 Future Work

Much work can be done to further enhance the CAAC framework we have presented in this thesis. The following are several interesting research directions that might be explored in the future.

- **Relationship Context Information with Different Granularities and Different Strengths**

  In today's open and dynamic environments, the relationship context information can bring new benefits to the access control mechanism/process. In order for the access control paradigms to provide relationship-specific access to information resources and software services in these environments, they need to capture the relevant relationship context information. A relationship-aware access control policy can be seen as the particular services can be invoked based on (i) the types of the relationship, (ii) the granularity levels of the relationship, and (iii) the relationship strengths. For example, a doctor's request to access a patient's medical records may be possible, but only when having the appropriate relationship with patient. Such service access request can be granted if there is a relevant relationship (type)
with certain granularity levels (‘Doctor-Patient’ relationship, at granularity level 0; ‘TreatingDoctor-Patient’ relationship or ‘EmergencyDoctor-Patient’ relationship, at granularity level 1) and strengths (‘strong’ or ‘weak’ relationship).

These new challenges require a new relationship-aware access control framework, in order to capture relationship context information and consequently make access control decisions. In this thesis, we have considered the relationship concept (the different types of relationship) in our context model without considering the relationship information at different granularities and different strengths. The relationship context information is highly dynamic, and granting a user access without taking the relationship information into account can compromise security as the user’s access privileges not only depend on "what the relevant relationship between resource requester and resource owner is" but also on "what the granularity levels and strengths of that relationship are".

Other than the "person-centric" relationship (what we have considered in this research), our framework can be extended to model and capture a broader range of relationships. Towards this end, we will explore other types of relationships in the similar manner. For example, (i) the "relationships between different resources" can be identified from the other data and (ii) a fine-grained "location-centric" relationship can be identified by using the spatial relations between different entities, i.e., nearby or co-located persons.

Overall, the rapid advancement of social computing technologies (e.g., [23], [88]) has led to the world to a new paradigm of access control from general context-aware to relationship-aware. Thus, we intend to study the different relationship context information with different granularities and different strengths in the future.

- Trustworthiness of Contextual Information in CAAC

The trustworthiness of the sensor values (from the context sources), i.e., the authenticity of contextual information, is another key issue we wish to investigate in
10.2. Future Work

the future. The authenticity of contextual information can play an important role in improving context-aware access control decision making and in ensuring the correct behavior of context-aware access control applications.

Contextual information is highly dynamic, and often inappropriate or unavailable as a result of noise or sensor failures [49]. As a result, context-aware systems should be developed with an understanding of the problems inherent in gathering reliable contextual information.

Existing research works have attempted to characterize imperfect context information of a physical nature such as location and time (e.g., [49]). The work presented in this thesis has considered not only the physical context information, but also the (social) relationship context information and the purpose-oriented information or user’s intention. However, the trustworthiness of social context information and purpose-oriented situation information yet needs to be addressed.

- **Prototype Framework for Mobile Platforms**

The users nowadays are becoming more and more dependent on mobile devices, and the use of contextual information in mobile devices such as cellphones, PDAs or laptops has been receiving increasing attention in pervasive and mobile computing research. As future work, we plan to extend our prototype framework to support mobile platforms. Although the experimental evaluation in this thesis demonstrates the feasibility of our framework in the desktop platform, it may not sufficient for mobile platforms. Leveraging the scalability of our framework in mobile platforms will be another important issue to be addressed.

- **Real Sensors**

In our in-lab experimental tests, we have used simulated sensors instead of real ones (such as patients’ wearable sensors, and location sensors). With real sensors, our processing stages will be the same except for the process of capturing the low-level
contextual information from the simulated sensors. For example, in order to identify a location with finer-granularity (such as general ward and emergency room) with real sensors, we intend to use existing technologies like the active badge system [94] in our future work. In the active badge system, healthcare professionals can wear badges that transmit signals providing information about their location to a centralized location system.

- **Transferring of Access Rights among Different Users**

In our CAAC prototype framework, we have considered the users’ access rights (on the resources) for dynamic access control decision making. Our framework uses the users’ provided data/information (\(<\text{identity}, \text{password}\>) pairs), in order to authenticate the users’ access rights. In practice, the access rights accompany access rules. As future work, we plan to consider the transferring of access rights among different users. Transferring an access right (to access resource or service) is to transfer it from a user to another (e.g., from a senior doctor to junior doctor). It can be carried out by developing a new policy manager (as part of our CAAC prototype framework), creating a new set of access control rules and inspecting available contextual information and other necessary data.
Bibliography


Bibliography


206


Bibliography


[77] S.-H. Park, Y.-J. Han, and T.-M. Chung. Context-role based access control for
and 45.)

[78] S. Preda, F. Cuppens, N. Cuppens-Boulahia, J. García-Alfaro, and L. Toutain. Dy-
namic deployment of context-aware access control policies for constrained security
42, 45 and 62.)

pallis, and G. A. Papadopoulos. A comprehensive context modeling framework for
pervasive computing systems. In Proceedings of the 8th IFIP WG 6.1 international
conference on Distributed applications and interoperable systems, pages 281–295,
2008. (Cited on pages 7, 53 and 54.)

[80] D. Riboni and C. Bettini. Owl 2 modeling and reasoning with complex human
55, 85, 87, 121 and 122.)

middleware infrastructure to enable active spaces. IEEE Pervasive Computing, 1:74–
83, 2002. (Cited on page 46.)


[83] R. Sandhu, D. Ferraiolo, and R. Kuhn. The nist model for role-based access control:
Towards a unified standard. In Proceedings of the Fifth ACM Workshop on Role-
based Access Control, RBAC ’00, pages 47–63, New York, NY, USA, 2000. ACM.
(Cited on page 33.)

control models. IEEE Computer, 29:38–47, 1996. (Cited on pages 4, 6, 8, 11, 20, 31,
33, 37, 41, 42, 51, 64, 73, 80, 101, 132, 134 and 192.)


