
Modeling Security Requirements for Context-Aware Systems using UML

Ph.D Thesis

Saad Matlaq Almutairi

This thesis is submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Software Technology Research Laboratory
De Montfort University
Leicester - United Kingdom

2013

Dedication

To my parents

For all the prayers, unconditional love and faith in me

To my wife

For her endless love, support and believing in me

Thank you for being there even during the hardest of times

Abstract

Modeling in general is “an abstract representation of a specification, design or system from a particular point of view”. System modeling is “a technique to express, visualise, analyse and transform the architecture of a system”. The Unified Modeling Language (UML) is “a language for specifying, visualising, constructing, and documenting the artefacts of a software-intensive system as well as for business modeling and other non-software systems”. UML consists of different types of diagrams such as Use Case diagram, Activity diagram, State diagram and Class diagram. Each type of these diagrams concerns a different aspects of the system development process.

Context-Aware Systems (CASs) are primarily associated with Pervasive/Ubiquitous Computing, which has become most prominent since the advent of smart phones and the inclusion of mobility features in computing devices. CASs can sense different aspects of their environment and use the dynamic Context Information (CI) to adapt their behavior accordingly. Hence, various precises of CI, such as User context, Physical context, Computer context and Time context, play a major role in controlling CAS behaviour and functions.

Security is considered one of major challenges in CAS specially because such systems often gather sensitive user information; this information may compromise the security of the system if disclosed to unauthorised users. Thus, the design of a CAS must consider system security as a major requirement. Although security is

traditionally considered as a non-functional requirement and is delayed to a later stage of the system development lifecycle, this thesis insists that security must be considered as early as possible because of its high importance. This is also in line with the “secure by design” concept.

Therefore, in this thesis the UML diagrams Use Case diagram, Activity diagram and State diagram will be enhanced in order to enable them to model a CAS and then capture its security requirements at the earliest possible stage of the software development process.

The contribution to knowledge that this thesis makes is at least threefold, as outlined below:

- Enhancing Use Case diagram notations to express dynamic CAS functional behaviour by showing the influences of CI changes. These extended notations are then used to capture the CAS security requirements.
- Enhancing Activity diagram notations in order to demonstrate and clarify the extended Use Case diagram by developing general diagram elements for CASs. This helps to show the data flow during the execution of a CAS function, and then present the security requirements.
- Enhancing State diagram notations to depict dynamism and security of a CAS also at this level, and to ultimately support the enhancement on Use Case and Activity diagrams.

These extended UML diagrams will be evaluated by applying them to a real-world Case Study to show their practical applicability. The case study is about an infostation-based mobile learning environment. This environment of Mobile Learning (M-learning) is deployed across a university boundary and provides a variety of services such ‘download lecture’ and ‘do exam’ to mobile users.

In conclusion, this research proposes and demonstrates an applicable approach to capture and model security requirements for CASs using innovative extensions of existing types of UML diagrams: Use Case, Activity and State.

Declaration

I declare that the work described in this thesis is original work undertaken by me for the degree of Doctor of Philosophy, at the software Technology Research Laboratory (STRL), at De Montfort University, United Kingdom.

No part of the material described in this thesis has been submitted for any award of any other degree or qualification in this or any other university or college of advanced education.

This thesis is written and produced using L^AT_EX.

Saad M Almutairi

Publications

1. S. Almutairi, A. Abu-Samaha, G. Bella and F. Chen. An enhanced Use Case diagram to model Context Aware System. *Science and Information Conference (SAI)*, London, UK, 7-9 October 2013.
2. S. Almutairi, H. Aldabbas and A. Abu-Samaha. Review on the Security Related Issues in Context Aware System. *International Journal of Wireless and Mobile Networks (IJWMN)* Vol. 4, No. 3, June 2012.
3. S. Almutairi, A. Abu-Samaha and G. Bella. Specifying Security Requirement for Context Aware System using UML. *Seventh International Conference on Digital Information Management (ICDIM)* 978-1-4673-2430-4/12/ in Macau, 2012 IEEE.
4. W. Alkhaldi, S. Almutairi, A. Almutairi and K. Aldrawiesh. Toward Development Context Aware Advertisement System (Case Study). *In proceedings of the IEEE 2011 International Conference on Computer Applications and Network Security (ICCANS 2011)*, Maldives, IEEE Press, CFP1182M-PRT/ISBN: 978-1-4244-9764-5, May 27th-29th 2011.

Acknowledgements

First and foremost, my truthful thankfulness goes to the most merciful **ALLAH** for all the things he blessed me with throughout my whole life, without those blessings, I would not be able to accomplish this work at all.

Most importantly, I would like to thank my mentor, supervisor and the one behind this project, **Dr Giampaolo Bella** for his endless support, encouragement and guidance. This thesis would not have been possible to achieve without him. I am so fortunate and happy to complete my Ph.d under his supervision.

Also, many thanks and gratitude goes to my supervisor **Prof- Hussein Zedan**, the director of the STRL, For his critical comments, technical suggestions and professional guidance. I admit that without his guidance this work would not be finished.

A very special thanks goes to **Dr Ala Abu-Samaha**, who gave me much time and effort to finish this work. I can only say it is the day to see one of your students become a doctor.

I want to express my deepest thanks to the one who all the good words would not be enough to describe, without, I would not be the man I am today: to my beloved father. What he has done for me, from big sacrifices to simple advice, is beyond the limits. He was the one to push me forward, to guide me, and to plant

every goodness in me. I owe it all to my father. I hope one day I can be a man just the way you are.

Also, I would like to thank my mother for her endless support, prayers and guidance since the day I was born. Thank you for being the perfect mother a person can wish for, thank you Mom for everything.

I would like to express my deepest love and gratitude for my wife, who stood by me in all these difficult times and has offered me her constant support, patience, encouragement, unconditional love and life.

I also like to acknowledge my daughters *Wajd and Taif*: looking into their eyes and playing with them gave me the happiness I needed through the bitter times.

I would like to thank my office mates *Dr. Abdullah Alamarshed and Dr. Fayez Alazmi*, for the enjoyable time and scientific atmosphere we had in the office.

I would like to thank my dearest friends and family whom life has scattered among different continents, thanks to everyone who showed his/her support throughout the years.

Last but not least, I would not forget to thank every member of the STRL for providing the academic and home-like environment and the support whenever needed, especially *Mrs. Lindsey Trent*.

Contents

Dedication	I
Abstract	II
Declaration	V
Publications	VI
Acknowledgements	VII
Table of Contents	XIV
List of Figures	XIX
List of Tables	XXI
List of Abbreviations	XXII
1 Introduction	1
1.1 Overview	2
1.2 Motivation	3
1.3 Research Methodology	4
1.4 Contribution	5
1.5 Thesis Scope and Research Questions	6

1.6	Measures of Success	7
1.7	Thesis Structure	8
1.8	Chapter Summary	9
2	Literature review	10
2.1	Modeling	11
2.1.1	UML	12
2.1.1.1	Use Case diagram	15
2.1.1.2	Activity diagram	16
2.1.1.3	State diagram	19
2.2	Key Concepts	20
2.2.1	Security	21
2.2.2	Security Requirements	22
2.2.3	Context and Context Aware System	23
2.2.3.1	Context	23
2.2.3.2	Context Aware System (CAS)	25
2.2.3.3	Context Aware System Lifecycle (CASLC)	26
2.2.3.4	Context Aware System Security	28
2.3	Framework supporting CAS security and their limitations	31
2.4	UML variants supporting CAS security and their limitations	35
2.5	Chapter Summary	39
3	Enhancing the Use Case diagram to model CASs and gather their security requirements	41
3.1	Introduction	42
3.2	Existing Use Case diagram elements	42
3.3	Enhancements	44
3.3.1	Extending the Use Case elements to model CASs	48

3.3.1.1	Adjusting the existing Use Case diagram elements . . .	49
3.3.1.2	Defining new Use Case diagram elements	50
3.3.2	Gathering security requirements for CASs using the extended Use Case	54
3.3.2.1	Authentication	55
3.3.2.2	Authorisation	58
3.3.2.3	Confidentiality	63
3.4	Chapter Summary	67
4	Enhancing the Activity diagram to support the extended Use Case diagram for CASs and gather their security requirements	68
4.1	Introduction	69
4.2	Existing an Activity diagram elements	69
4.3	Enhancements	70
4.3.1	Extending the Activity diagram to support the extended Use Case diagram	76
4.3.2	Gathering security requirements for CASs using the extended Activity diagram	82
4.3.2.1	Authentication	84
4.3.2.2	Authorisation	91
4.3.2.3	Confidentiality	96
4.3.2.4	Integrity	101
4.4	Chapter Summary	107
5	Enhancing the State diagram to support the extended Use Case diagram for CASs and gather their security requirements	108
5.1	Introduction	109
5.2	Existing State diagram elements	109

5.3	Enhancements	110
5.3.1	Extending the State diagram to support the extended Use Case diagram	114
5.3.2	Gathering security requirements for a CASs using the ex- tended State diagram	119
5.3.2.1	Authentication	120
5.3.2.2	Authorisation	123
5.3.2.3	Confidentiality	127
5.3.2.4	Integrity	131
5.4	Chapter Summary	135
6	A Real-World Case Study: An Infostation-based M-Learning Sys- tem	136
6.1	Introduction	137
6.2	M-learning definition	137
6.3	M-learning Infrastructure	138
6.4	Description of M-learning system	139
6.5	M-learning system users and functions	141
6.6	Capturing the key security requirements for M-learning system func- tions	143
6.6.1	Authentication	143
6.6.2	Authorisation	144
6.6.3	Confidentiality	146
6.6.4	Integrity	146
6.7	Modeling the M-learning system using UML	147
6.7.1	Use Case diagram for the system	147
6.7.2	Activity diagram for the Download Materials function	151

6.7.3	State diagram for the Download Materials function	152
6.8	Capturing the security requirements for M-learning functions using UML	154
6.8.1	Authentication	154
6.8.1.1	Using our Enhanced Use Case diagram	154
6.8.1.2	Using our Enhanced Activity diagram	156
6.8.1.3	Using our Enhanced State diagram	157
6.8.2	Authorisation	159
6.8.2.1	Using our Enhanced Use Case diagram	159
6.8.2.2	Using our Enhanced Activity diagram	161
6.8.2.3	Using our Enhanced State diagram	163
6.8.3	Confidentiality	164
6.8.3.1	Using our Enhanced Use Case diagram	164
6.8.3.2	Using our Enhanced Activity diagram	165
6.8.3.3	Using our Enhanced State diagram	167
6.8.4	Integrity	168
6.8.4.1	Using our Enhanced Activity diagram	168
6.8.4.2	Using our Enhanced State diagram	170
6.9	Chapter Summary	171
7	Conclusion and Future work	172
7.1	Research Summary	173
7.2	Statement of Evaluation	175
7.3	Research Questions Revisited	176
7.4	Contribution to Knowledge	177
7.5	Future Work	178
	Bibliography	195

A	Presenting the rest of UML diagrams	196
A.1	Modeling M-learning system functions	196
A.1.1	Use Case diagram	196
A.1.2	Activity diagram	200
A.1.3	State diagram	205
A.2	Capturing security requirement for M-learning system functions using UML	210
A.2.1	Authentication	210
A.2.2	Authorisation	210
A.2.2.1	Using our Enhanced Use Case diagram	210
A.2.2.2	Using our Enhanced Activity diagram	213
A.2.2.3	Using our Enhanced State diagram	219
A.2.3	Confidentiality	223
A.2.4	Integrity	224
A.2.4.1	Using our Enhanced Activity diagram	224
A.2.5	Using our enhanced State diagram	230

List of Figures

2.1	Use Case diagram elements	16
2.2	Activity diagram elements	18
2.3	State diagram elements	20
2.4	Context Aware System Lifecycle (CASLC)	27
3.1	Normal Use Case diagram	45
3.2	Extended Use Case diagram	52
3.3	Authentication process in CAS	56
3.4	Authentication Use Case diagram	57
3.5	Authorisation process in CAS	59
3.6	Authorisation Use Case diagram	61
3.7	Confidentiality process in CAS	64
3.8	Confidentiality Use Case diagram	65
4.1	CAS Activity diagram framework	72
4.2	CASLC Activity diagram	75
4.3	CASLC Activity diagram with proposed enhancements	79
4.4	Check books Use Case	80
4.5	Check books Activity diagram	81
4.6	Present Authentication by login action state	84
4.7	Expanded login action state	85

LIST OF FIGURES

4.8	Authentication with CASLC	86
4.9	Library system login Use Case	87
4.10	Librarian Authentication process	88
4.11	Determining the Authentication process during CAS	90
4.12	Basic concept of Authorisation mechanism	91
4.13	Authorisation mechanism with CAS	92
4.14	Authorisation within CAS Activity diagram	94
4.15	Determining the Authorisation process during CAS	95
4.16	Basic concept of Confidentiality mechanism	97
4.17	Confidentially with CASLC	98
4.18	Confidentiality within CAS Activity diagram	99
4.19	Determining the confidentiality process during CAS	100
4.20	Basic concept of Integrity mechanism	102
4.21	Integrity within CAS Activity diagram	104
4.22	Determining the Integrity process during CAS	106
5.1	General states for mobile device	111
5.2	Decomposed Active state	112
5.3	State diagram model with proposed enhancements	115
5.4	State diagram model with CASLC	116
5.5	Check book Use Case	117
5.6	Check book State diagram	118
5.7	Basic concept of Authentication mechanism	120
5.8	Authentication with CASLC	122
5.9	Basic concept of Authorisation in State diagram	124
5.10	Authorisation with CASLC	126
5.11	Basic concept of Confidentiality in State diagram	127

5.12 Confidentiality with CASLC	130
5.13 Basic concept of Integrity in State diagram	132
5.14 Integrity with CASLC	134
6.1 The three-tier architecture of the infostation-based network	139
6.2 University environment divided in accordance with deployed CIPs	140
6.3 Main Use Case diagram for M-learning	148
6.4 Download materials Use Case	150
6.5 Download materials Activity diagram	152
6.6 Download materials State diagram	153
6.7 Login Use Case	155
6.8 Authentication through Activity Diagram	157
6.9 Authentication through State diagram	158
6.10 Authorisation for upload exam using Use Case diagram	159
6.11 Authorisation for upload exam using Activity diagram	162
6.12 Authorisation for upload exam using State diagram	163
6.13 Confidentiality for upload exam using Use Case diagram	164
6.14 Confidentiality for Upload exam using Activity diagram	166
6.15 Confidentiality for upload exam using State diagram	167
6.16 Integrity for upload exam using Activity diagram	169
6.17 Integrity for upload exam using State diagram	170
A.1 View materials Use Case diagram	197
A.2 Do exam Use Case diagram	197
A.3 Make a private chat Use Case diagram (Student)	197
A.4 Make a private chat Use Case diagram (Lecturer)	198
A.5 Mark students exams Use Case diagram	198
A.6 Upload lectures Use Case diagram	199

A.7 View materials Activity diagram	200
A.8 Do exam Activity diagram	201
A.9 Make a private chat Activity diagram (Student)	202
A.10 Make a private chat Activity diagram (Lecturer)	203
A.11 Mark students exams Activity diagram	204
A.12 Upload lectures Activity diagram	205
A.13 View materials State diagram	206
A.14 Do exam State diagram	206
A.15 Make a private chat State diagram (Student)	207
A.16 Make a private chat State diagram (Lecturer)	207
A.17 Mark students exams State diagram	208
A.18 Upload lectures State diagram	209
A.19 View materials Authorisation Use Case	210
A.20 Make private chat Authorisation Use Case	211
A.21 Mark student exam Authorisation Use Case	211
A.22 Upload exam Authorisation Use Case	211
A.23 Make private chat Authorisation Use Case	212
A.24 Upload exam Authorisation Use Case	212
A.25 View materials Authorisation Activity diagram	213
A.26 Download materials Authorisation Activity diagram	214
A.27 Do exam Authorisation Activity diagram	215
A.28 Make a private chat Authorisation Activity diagram	216
A.29 Upload lectures Authorisation Activity diagram	217
A.30 Mark students exams Authorisation Activity diagram	218
A.31 Make a private chat Authorisation Activity diagram	219
A.32 View materials Authorisation State diagram	220
A.33 Do exam Authorisation State diagram	220

LIST OF FIGURES

A.34 Make private chat Authorisation State diagram	221
A.35 Upload lectures Authorisation State diagram	221
A.36 Upload exam Authorisation State diagram	222
A.37 Mark students exams Authorisation State diagram	222
A.38 View materials Integrity Activity diagram	224
A.39 Make private chat Integrity Activity diagram	225
A.40 Do exam Integrity Activity diagram	226
A.41 Upload exam Integrity Activity diagram	227
A.42 Mark students exam Integrity Activity diagram	228
A.43 Upload lectures Integrity Activity diagram	229
A.44 Make private chat Integrity Activity diagram	230
A.45 View materials Integrity State diagram	231
A.46 Do exam Integrity State diagram	231
A.47 Make private chat Integrity State diagram	232
A.48 Upload exam Integrity State diagram	232
A.49 Upload lectures Integrity State diagram	233
A.50 Mark student exam Integrity State diagram	234

List of Tables

2.1	Static and Dynamic diagrams of UML	14
2.2	Relationship between frameworks and security requirements	34
2.3	Summary of the above mentioned modeling methods based on the defined assessment criteria	38
3.1	Existing Use Case relationships	44
3.2	Adjusted notations	50
3.3	The proposed new elements to model CASs and to capture their se- curity requirements	51
3.4	Description of Use Case diagram	53
3.5	Typical VS Adjusted Use Case Diagram	54
3.6	Presenting security requirement using Use Case diagram	55
3.7	Description of Authentication Use Case diagram	58
3.8	The proposed Authorisation Use Cases	60
3.9	Description of Authorisation Use Case diagram	62
3.10	The proposed Confidentiality Use Case	65
3.11	Description of Confidentiality Use Case diagram	66
4.1	CAS swimlanes functions	73
4.2	CASLC swimlanes table	74

4.3	Proposed notations to enhance the Activity diagram to present the adjusted Use Case diagram	77
4.4	Icons used to present the security requirements in the extended Activity diagram	83
5.1	Description of Active state	113
5.2	Description of Authentication states	121
5.3	Description of Authorisation states	125
5.4	Description of confidentiality states	129
5.5	Description of Integrity states	133
6.1	Student functions	149
6.2	Lecturer functions	149
6.3	Download materials Use Case description	151
6.4	Authentication Use Case diagram description	156
6.5	Authorisation Use Case description	160
6.6	Confidentiality Use Case description	165

List of Abbreviations

UML	Unified Modeling Language
CAS	Context Aware System
OMG	Object Management Group
CI	Context Information
CASLC	Context Aware System Life Cycle
OCL	Object Constraint Language
CIS	Context Information Store
FR	Function Requirement
M-learning	Mobile Learning
CIP	Context Information Provider
SAY	Study Academic Year
CenStat	Central Infostation
PDA	Personal Digital Assistant
UAC	Uniform Access Control
RBAC	Role-Based Access Control
GRBAC	General Role Based Access Control

XML	Extensible Markup Language
D.B	Data Base
GRBAC	General Role Based Access Control
IRC	Internet Relay Chat
ACI	Authentication Context Information
BPMN	Business Process Modeling Notation
CCAA	Calculus of Context-Aware Ambient
FML	Formal Modeling Language
PIM	Platform Independent Model
BPSec	Business Process Security
CIM	Computation Independent Model
UCS	Use-Case Specification
CCA	Context-Aware Application
SML	Structural Modeling Language
SFML	Semi Formal Modeling Language
EML	Extensible Markup Language

Chapter 1

Introduction

Objectives:

- To give an introduction and the motivation of this research
 - To present the research methodology
 - To list the research questions
 - To present the measures of success
 - To give the thesis structure
-

1.1 Overview

Modeling in general is “an abstract representation of a specification, design or system from a particular point of view” [110]. System modeling is a technique for expressing, visualising, analysing and transforming the architecture of a system. Here, a system may consist of software components, hardware components (usually both), and the connections between those components, all of which can be illustrated in a skeleton model of the system [88]. The Unified Modeling Language (UML) is “a major tool for specifying, visualising, constructing and documenting the artefacts of a software-intensive system as well as for business modeling and other non-software systems” [19].

UML provides different types of diagrams such as Use Case diagram, Activity diagram and State diagram. Each types of these diagrams concerns different modeling aspect. This thesis demonstrates that they can be profitably enhanced to capture and model security requirements at early stage of software development.

Context Aware Systems (CASs) are primarily associated with Pervasive/Ubiquitous Computing, which became most prominent since the advent of smart phones and the inclusion of mobility features in computing devices. CASs are those which can sense different aspects of their environment and use the dynamic Context Information (CI) to adapt their behaviour accordingly. Dey et al [31]. provided a comprehensive definition of context as: “any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves”. For that, CASs rely on three key processes;

- sensing the context in the surrounding environment;
- reasoning about changes in context;

- react to those changes.

As the CI that constitute CAS are rapidly changing, it is challenging to continuously keep track of it while preserving the security and privacy of the system and its users.

1.2 Motivation

As noted in the previous section, one of the most challenging features of CAS is the system's ability to keep track of rapidly changing CI, such as location and time of day, as well as the presence of nearby people and devices. These constitute potential threats to the system security. Hence, while capturing security requirements in the initial stage of the CAS development process becomes necessity, it also turns out to be challenging. Traditionally, security requirements have been perceived as relatively static in nature, as access control decisions do not change with CI nor do they account for changing conditions in the system/application environment. However, the challenge of context awareness derives from the mobility of the user and the changing context of his/her environment.

We select Use Case, Activity and State diagrams as the most widely used diagrams in the early stages of software development through UML modeling. We will refer to these three diagrams as our three "target diagrams". Each target diagram can be used to capture and define system requirements specially at the initial phase of software development. It is important to highlight how limited the current form of the target notations are, and at the same time, the need to extend/enhance them to enable specifying the security requirements of a CAS. Firstly, current target diagrams are mainly capable of capturing and modeling the system's functional requirements; however, security requirements are known to be non-functional in nature, and hence have so far been excluded from the target diagrams and the early

phases of the software development process. Secondly, the current target diagrams are mainly capable of modeling traditional system functions and behaviour that are static, and hence the system design is not influenced by any external condition. This defies the basic nature of a CAS, which is adaptable to its environment in a constant manner.

As a result of these shortcomings, a new set of extensions/enhancements for all the three target diagrams are needed to enable them to fully model CASs and their security requirements.

1.3 Research Methodology

The research methodology used in this work is Constructive method, which is a customary scientific research technique. In constructive method, the novelty is established using an inventive architecture, prototype or procedure. A very high level expertise is required in the area of research to establish novelty, with the literature plays the pivotal role [116]. Consequently, the suggested approach is composed of five work packages; starting with review of the state-of-the art, then proceeding to the proposed extension on Use Case diagram. The third and fourth work package present how Activity and State diagrams demonstrate the extension that is done on Use Case diagram. The last work package deals with a comprehensive M-learning Case study, which is used to emphasize the practicality of the proposed extensions as discussed in work packages 2 to 4.

1. Work package 1: Research background

Literature review is an exploration of the existing work in the field of the research taken in this research and it mainly focuses on the research questions raised earlier. The literature review has been done mainly using books, journals, online resources, peer discussions, and paper reviews.

2. Work package 2: Extending Use Case diagram

Extending Use Case diagram notations to express dynamic CAS functional behaviour and inculcating the effect of CI on CAS's behaviour. These extended notations are then used to capture the CAS security requirements.

3. Work package 3: Extending Activity and State diagrams

Extending Activity and State diagrams notations in order to demonstrate and clarify the extended Use Case diagram by developing general diagram elements for CASs. Extending Activity diagram demonstrates the data flow during the execution of a CAS function. Consequently enhancing the State diagram notations to depict dynamism and security of a CAS, and to ultimately support the enhancement on Use Case and Activity diagrams.

4. Work package 4: Evaluation

In the final phase, a real-world case study is conducted using the proposed enhancements; this case study is about a M-learning system. This case study is employed in order to clarify how CAS functions can be modelled and its security requirements can be gathered using the target diagrams.

1.4 Contribution

This thesis aims to develop an approach using UML diagrams to capture and model the security requirements needed for CASs in an effective manner. Although many aspects of security requirements and CASs have been studied, the combination of the three aspects (UML, CASs and security requirements capturing) in one study distinguishes the present research from others. The key contributions of this thesis therefore are as follows:

- Enhancing the existing Use Case notations to model a CAS, and then using this extension to capture the security requirements in the initial stage of the software development process. This extension will show how the CI can manipulate a CAS functioning.
- Enhancing the Activity diagram notations to support our extended Use Case diagram; this guides builds a clearly structured framework for modeling a CAS and gathering its security requirements through an Activity diagram.
- Enhancing the State diagram notations to support our extended Use Case diagram; this demonstrates our new framework for modeling a CAS and gathering its security requirements through a State diagram.

1.5 Thesis Scope and Research Questions

The work in this thesis focuses on the following issues:

- UML firstly this research focuses on UML diagram types, namely Use Case diagram, Activity diagram and State diagram. It also provides their definitions as well as the limitations that hinder their capability to model CAS then gather its security requirements.
- CAS secondly this research targets a main technology as its main goal, namely a Context Aware System, and the general notion of context and lifecycle.
- Security requirement this research aims at capturing and modeling several types of security requirements, specifically Authentication, Authorisation, Confidentiality and Integrity. In order to secure a CAS, these types explored in detail and then innovatively incorporated in the modeling tools.

In the light of the above arguments regarding the scope of this thesis, the main research question is formulated as follows:

How to model CASs and gather its security requirements using UML?

However to answer this question appropriately requires, in turn, answering the following sub-questions:

Q1. Are the current form of Use Case notations applicable to modeling a CAS?

If the answer is NO, then:

Q2. Can the Use Case notations be extended to model a CAS?

If the answer is Yes, this leads to another question:

Q3. Is the extended Use Case diagram capable to capture all the security requirements, namely Authentication, Authorisation, Confidentiality and Integrity?

If the answer is Yes, this poses a new question:

Q4. Are the existing notations of both Activity and State diagram mature enough to present the extended Use Case diagram?

If the answer is No,

Q5. Can the current notations of Activity and State diagram be enhanced to do so?

If the answer is Yes, finally:

Q6. Can the proposed extensions for gathering and modeling the security requirements of CAS be practically applicable to real-world case studies?

1.6 Measures of Success

The success criteria for the work reported in this thesis are as follows:

- The research questions must be clearly answered.
- A study showing how the proposed UML diagrams extensions contribute to knowledge by advancing existing research in the area.

- A study illustrating the advantages of using UML diagrams as empowered through the present work must be provided.
- A large real-world case study to demonstrate that the enhanced UML diagramming techniques can effectively model a CAS and its security requirements.

1.7 Thesis Structure

The first chapter is the thesis introduction, and the remaining chapters are structured as follows:

- Chapter 2: this Chapter provides an overview of the literature that influenced the proposed research. The review will present background information on UML and its various diagram types, security and its requirements in general, security requirements for CAS in practical, and frameworks for modeling CAS; the Chapter provide in the same time definitions for context and CAS.
- Chapter 3: this Chapter investigate the Use Case notations in depth, extend the Use Case diagram to model a CAS, and then to capture its security requirements.
- Chapter 4: this Chapter studies Activity diagram notations, refining them and assessing their capability in terms of supporting the extension that was done on the Use Case diagram.
- Chapter 5: this Chapter presents the enhanced State diagram to support the extension that was done on the Use Case diagram.
- Chapter 6: this Chapter presents a case study in order to demonstrate the effectiveness of the UML extensions proposed in the previous Chapters.

- Chapter 7: this Chapter summarises the work discussed in the entire thesis, highlights the significance of the proposed contributions and outline directions for possible future work.

1.8 Chapter Summary

This chapter has highlighted the main aspects of this research, such as UML diagram types, CAS and the importance of capturing the security requirements when modeling a CAS. In addition this Chapter has introduced the motivation of the thesis; moreover, it has clearly detailed the contribution, the research questions, the measures of success and the structure of the thesis.

Chapter 2

Literature review

Objectives:

- To describe the concept of modeling
 - To introduce UML and its diagram types
 - To explain the security and security requirements
 - To introduce the context, context aware system and its lifecycle
 - To review the context aware system frameworks
 - To investigate the related work
-

2.1 Modeling

Modeling in general is “an abstract representation of a specification, design or system from a particular point of view” [110], [97]. System modeling is a technique to express, visualise, analyse and transform the architecture of a system. Here, a system may consist of software components, hardware components, or both and the connections between these components [88]. System modeling is intended to assist in developing and maintaining large systems with emphasis on the construction phase [120]. The idea is to encapsulate complex or changeable aspects of a design inside separate components with well-defined interfaces indicating how each component interacts with its environment [117].

System model can increase reliability and reduce development cost by making it easier to build systems, to reuse previous built components within new systems, to change systems to suit changing requirements such as functional enhancement and platform changes, and to ultimately better understand systems. In this way, a system model can support various goals, such as documenting the system, providing a notation for tools, such as consistency checkers, and can also be used in the design stage of system development [97].

There are several ways of expressing system modeling [35], [36], [49]. Here, we only mention the most used ones:

- Structural Modeling Language (SML): it rests on diagramming techniques with named symbols that represent concepts and lines that connect the symbols and represent relationships, such as (UML, Flowchart, Business Process Modeling Notation (BPMN)) [113]. SML is concerned with visually describing all the “things” in a system and how these relate to each other.
- Formal Modeling Language (FML): it is a set of strings of symbols that may

be constrained by rules that are specific to it, such as (Calculus of Context-Aware Ambient (CCAA) [106]). FMLs have a number of advantages over informal languages, such as their precise meaning and the possibility to derive properties through formal proofs [96].

- Semi formal Modeling Language (STML): it is language that may use standardized keywords accompanied by parameters or natural language terms and phrases to make computer-interpretable expressions, such as (Object Constraints Language (OCL)) [44].

This thesis chooses the Structural approach to modeling CASs and their security requirements. In particular, it will adopt UML to do so both because of its popularity and because of its flexibility.

2.1.1 UML

The Unified Modeling Language(UML) emerged in 1994 through the Object Management Group (OMG) [19]. The OMG claim that “UML is a language for specifying, visualising, constructing, and documenting the artefacts of a software-intensive system as well as for business modeling and other non-software systems”.

As a result, UML has become one of the most popular modeling languages in the field of software engineering [113], [67]. UML is mainly characterized as a graphical language based on rules for creating, designing and analysing the system development methods [95],[117]. UML is a very expressive language, addressing all the views needed to develop and then deploy software systems. UML is not restricted to modeling software, it can also be utilised for many other purposes, such as building models for system engineering, business processes and organizational structures [3].

UML represents a set of best engineering practices that have proven successful in the modeling of a wide range of complicated systems. UML generally offers a set

of modeling diagrams specialised in the different view of information, hence UML is very useful for both illustrating designs and understanding them. UML allows the separation of concerns by using several diagrams in order to focus on different aspects of a software system. Therefore, a number of different diagram types may be employed in order to describe different aspects of the system at various degrees of abstraction [61], [63].

Therefore, using the UML diagrams will provide a standardised way to write system blueprints, covering conceptual aspects such as business processes and system functions, as well as more concrete aspects [87].

The UML diagrams can be classified into two main types as follows [117],[91]:

- **Static.** This describes the static semantics of data and messages within system development. Structure diagrams also define the static architecture of a model, representing the physical and conceptual elements. Such diagrams are mainly concerned with modeling classes, objects, interfaces and physical components in addition to the dependencies and relationships among them [8].
- **Dynamic.** This entails modeling the dynamic aspects of the system. A dynamic model represents the interactions and the activities within the system and with the users and the environment [8]. A dynamic model can be also understood as a description of the behaviour of the system over time. Behaviour diagrams capture all the various interactions and instantaneous states within a model while it 'executes' over time [41].

The Table 2.1 summarises the types of UML diagrams:

UML diagram	Type
Class diagram	Static
Object diagram	Static
Component diagram	Static
Deployment diagram	Static
Package diagram	Static
Composite structure diagram	Static
Communication diagram	Dynamic
Timing diagram	Dynamic
Use Case diagram	Dynamic
Sequence diagram	Dynamic
Activity diagram	Dynamic
State machine diagram	Dynamic

Table 2.1: Static and Dynamic diagrams of UML

This thesis endeavours to explore some of the most widely used behavioral diagrams of UML Modeling notations, namely, Use Case diagram, Activity diagram and State diagram. The reason behind selecting these types over the others are that the selected ones are all classified as behavioral modeling types which help to define and model dynamic systems. In addition the Use case diagram is one of the means of gathering security requirement [29], and also the Activity diagram and State diagram are very beneficial to depict the system dynamic behaviour [80],[72].

Therefore, the key factor that distinguishes a CAS from other conventional systems is that CAS behaviour is inherently dynamic (based on changes in its environment), and therefore the selected UML diagrams will prove beneficial in defining CAS behaviour and then capturing its security requirements. Accordingly, the selected diagrams are discussed in the sequel of this section.

2.1.1.1 Use Case diagram

Use Case diagrams are used to describe the abstract view of the functionality offered by a system through specifying typical interactions with the system environment [75]. Moreover, a Use Case specifies a sequence of actions (including variants) that the system can perform, detailing how actors interact with the system [55]. A Use Case is the core element of a UML behavioural diagram, and is a methodology used in system analysis in order to identify, clarify and organize system requirements [95].

However, a Use Case was originally defined by Jacobson as, “a sequence of transactions in a system whose task is to yield a result of measurable value to an individual actor of the system” [57]. A Use Case has been classified as an effective method for gathering system requirements from the user perspective, particularly during the elicitation phase [38], [56], [6]. Hence a Use Case entails a group of different scenarios; each scenario depicts in detail a sequence of interactions between the system and its environment (users and other external systems and/or devices).

A Use Case diagram is a powerful tool because in addition to defining basic system functions, it also serves to test individual case scenarios in order to assure that those requirements have been met [113]. A Use Case diagram is typically used to clarify the high-level functions of the system as well as the system scope. Consequently, a Use Case diagram is usually the first diagram that should be constructed when a software engineer starts modeling a system. Thus, the main purpose of utilising a Use Case diagram is to help software developers to identify, visualise, organise and clarify system functional requirements. In addition, the Use Case construct can be used to define the behaviour of a system without revealing its internal structure.

Use case diagram consists of three main elements as depicted in Figure 2.1

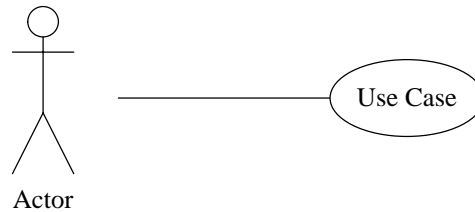


Figure 2.1: Use Case diagram elements

2.1.1.2 Activity diagram

An Activity diagram is defined by the OMG as a diagram derived from various techniques to illustrate workflows in a visual manner [117]. An Activity diagram specifies the control flow between several components within a system in order to depict the main dynamic aspects.

Activities eventually lead to some form of an action, which is composed of executable atomic computations that result in a change in the state of the system or in the return of a value. An Activity diagram can affectively model the dependency between the activities as well as the decision points that enable the branching of those activities (based on specified conditions). An Activity diagram can also model the synchronisation of activates and map them through multiple threads and to corresponding actors [95]. Typically an Activity diagram is used for modeling the logic captured in a specific Use Case diagram [24].

Thus, an Activity diagram is similar to a flowchart in behavior, which illustrates the data flow between the various activities of a program or a business process. Hence, an Activity can be described as a state of performing an action, either a real-world process such as typing a report, or executing a software function, such as

a method in a class [20]. Based on the above, Activity diagrams can be used independently of Use Case diagrams for other purposes, such as to model the business process of a system or to model the detailed logic of a business role [24].

Activity diagrams can also be used to visualise, construct, specify and document the behaviour of a group of objects [80]. In short, “an Activity diagram in its basic form is a simple and intuitive illustration of what happens in a workflow, what activities can be done in parallel, and whether there are alternative paths through the workflow” [59]. The rule for reading Activity diagrams is from top right (presented as the initial node) to bottom (presented as the end node).

Figure 2.2 shows an example of Activity diagram

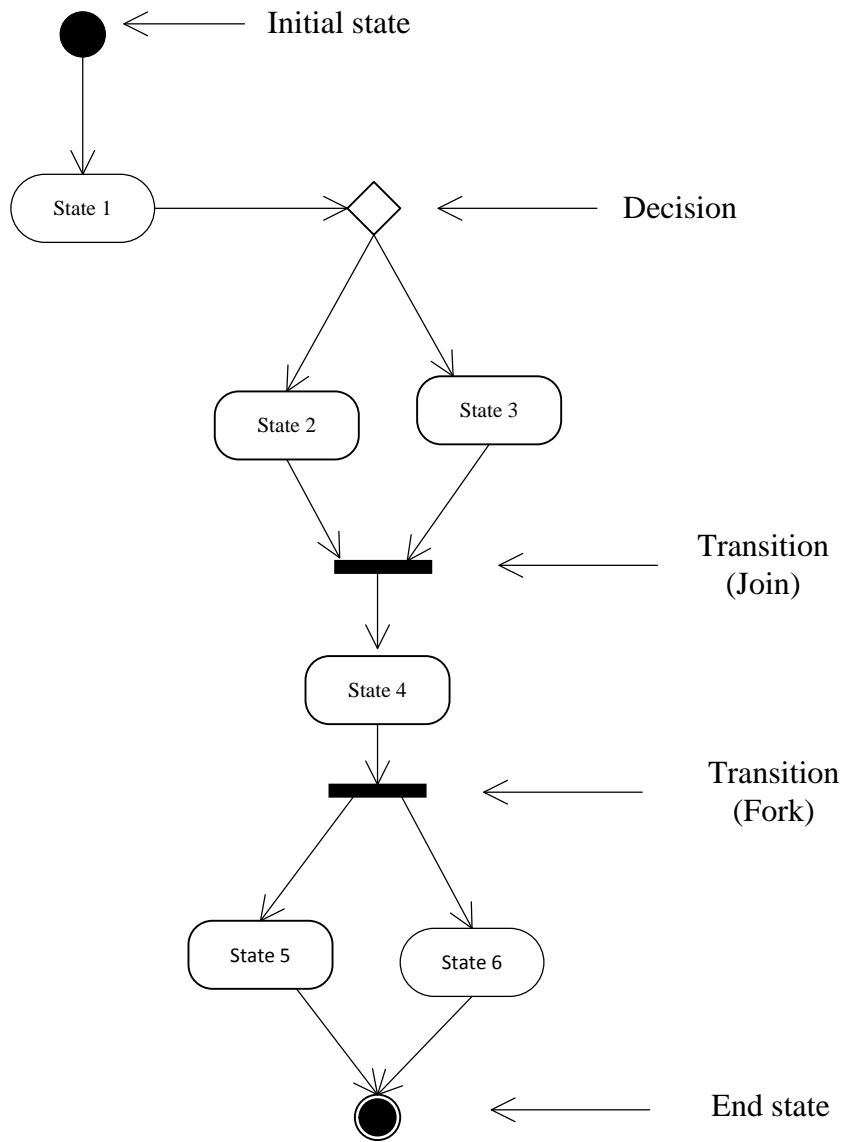


Figure 2.2: Activity diagram elements

2.1.1.3 State diagram

State diagrams express the dynamic behaviour of an individual object or component; hence events may cause a change in state or execution of actions [33]. A State diagram is considered one of the most important UML types for modeling the dynamic nature of a system [72].

A State diagram is mainly used to provide a very clear abstract description of the system behaviour, taking into account all the possible states of an object when an event occurs. This behaviour can be fully described and represented via a sequence of events that could happen in one or more states. A State diagram is very beneficial in modeling the interactions between a class and the system interface, and also in realising Use Cases [92]. It is also used to specify the sequence and time behaviour of the objects in any given class (State, Event and Transition) [47].

A State diagram illustrates the discrete behaviour of a part of the designed system through finite State transitions [115]. State diagrams can also be used to express the usage protocols of part of a system. In some cases, State diagrams are considered as directed graphs, as the graph nodes can represent states and the labels of the graph edges can represent actions. Moreover, a State diagram is similar to a flowchart in terms of notation; however, it differs from other types of UML diagram, such as Class diagram, Use Case diagram and Object diagram, in that a State diagram expresses single objects whereas the others describe groups of objects. Therefore, a State diagram is imperative for both system analyst and developer to fully express the behaviour of the objects in a system. It can be concluded that the main purposes of using a State diagram are as follows [72].

- To model the dynamic aspect of a system.
- To model the life time of a reactive system.
- To describe the different states of an object during its life time.

Figure 2.3 shows an example of State diagram

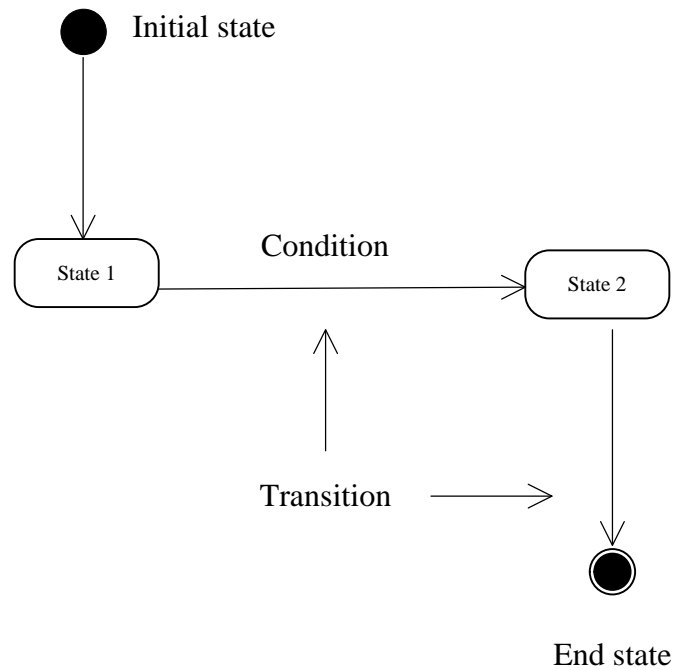


Figure 2.3: State diagram elements

2.2 Key Concepts

As explained in the thesis introduction, this study concerns many distinguishing aspects, in particular, security, security requirements, context and context-aware systems and thus it is necessary to highlight each concern in order to identify and define any weaknesses that could represent an obstacle to capturing and modeling the security requirements for a CAS using UML diagram types.

2.2.1 Security

Security in a generic sense means “freedom from risk or danger”; in the context of computer science, security is “the prevention of, or protection against access to information by unauthorised recipients, and intentional but unauthorised destruction or alteration of that information” [54].

Security is the one of the key elements that can hinder the growth and spread of software systems. Security is devoted to prevent any activity that may pose a threat to either the stakeholders or the system itself [118]. In addition, security is a system property that should remain dependable even following malicious activity, error or misfortune [109], [66].

In spite of the huge numbers of studies currently available on general security, they can broadly be classified into three main groups as follows:

- Group 1 concerns the modeling of malicious behaviour and vulnerabilities (for example, Misuse Case [107], Abuse Case [75]).
- Group 2 concerns the countermeasures and the security requirements, for instance Authentication, Authorisation, Confidentiality, Integrity, Non-repudiation and Availability [95], [61], [108].
- Group 3 concerns the protocols and the transformations of sensitive data [13], [17].

This thesis falls within the second group of security studies, with its focus on security requirements. They will be explored in depth and then utilised in order to define a secure environment for CASs.

2.2.2 Security Requirements

Security is the subset of quality requirements that describe certain qualities of system services [60], [111], [45], [108]. Security requirements specifically deal with determining how the system is to be protected against any kind of threat [45], [99], [58]. The role of security requirements is to provide information on the actual needs of a system or application with respect to security in order to accomplish its business goals [21]. Each security requirement is defined in terms of security policies, which state certain constraints on functionality. Thus, any inadequate understanding of the security requirements could lead to serious consequences, such as leak of sensitive information, or even system failure [118], [51]. Hence, such requirements must be gathered and maintained in an efficacious manner in an initial phase of the software development process along with other system functional requirements [108]. Security requirements are classified into the following groups [40], [37], [39]:

- Authentication requirement. This type of requirements seeks to verify the user identity (whether an end user, an external system or other integrated applications).
- Authorisation requirement. This type of requirements is defined as the process of granting permissions or access to authenticated users to benefit from system facilities. At the same time, authorisation denies access to system utilities in case of a fake or unauthorised user. In addition, authorisation helps users to restrict each other from accessing their perspective private information.
- Confidentiality requirement. This security requirement ensures that information is accessible only by authorised persons.
- Non-repudiation requirement. This security requirement specifies the extent to which the system shall maintain tamper-proof evidence recording all accesses

made to the system.

- Integrity requirement. This security requirement ensures that system data cannot be deliberately corrupted or modified by any unauthorised entity.
- Availability requirement. This security requirement prescribes that services and applications in the system should be accessible, when needed, even in the presence of faults or malicious attack.

2.2.3 Context and Context Aware System

To fully comprehend context-aware system, it is essential to define a fundamental related point, such as what context means in general, and then to classify the context types.

2.2.3.1 Context

Defining the term ‘context’ is challenging because it has a variety of meanings. According to the Oxford Dictionary, context is a circumstance in which something happens or in which something needs to be considered; it also refers to time and situation. According to the Merriam-Webster Collegiate Dictionary, context is “the interrelated conditions in which something exists or occurs”. However, many researchers are not convinced by such general concepts, and so they have attempted to produce a more applicable concept of context which is related to computing. In this vein some consider context to be the user’s environment and others the system’s environment.

However, more prominent definitions do consider context in relation to both the users or persons and to the systems or objects as entities that are found in a particular circumstance, such as Schmidt et al [102]. They defined context as

“knowledge about the user’s and IT device’s state, including surroundings, situation, and to a less extent, location”.

Schilit and Theimer [101] define context by providing examples associated with location, identity of nearby people and objects and changes to those objects. The definition of context in Ryan et al [98] also includes location of the user, identity and time. However Dey and Abowd provided a foundational and comprehensive definition of context that makes it easier to identify the various elements that context features. That definition is emphasised in the following box [30];

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

This definition is adopted coherently in this thesis as the main working definitions of context. In consequence, the context of a user can be seen as information about the user and their environment [1].

Most research studies have provided various categories of CI, the majority of which agreed collectively upon four major categories [78], [31], [46], [1], [105], [43]. These are as follows:

- Computing context: it covers areas and information related to system connectivity, network capabilities, Internet access and speed, and other computing related assets such as printers and workstations, CPU size and speed, memory resources, etc. Another important aspect of context-aware applications in terms of computing context is that, in order to avoid the use of multiple devices, users may at times prefer those devices that provide the facility to per-

form multiple functionalities at the same time. PDAs and smartphones are the most obvious example of this kind of devices, which allow for the integration of innumerable applications

- **User context:** it contains information relating to the user's application usage, which includes the user's personal information, preferences, current location, potential activity, etc. The choices users make in determining the context are related to their preferences. For instance, some users are willing to go to any specified location by public or private transport, whereas others are willing only to travel within a walking distance. For this reason, a component or facility for storing and efficiently using preference-related information is required.
- **Physical context:** it is sometimes referred to as the environmental context, and provides information regarding current location (public or private), intended destination, time, environmental conditions (light or dark, noisy or imposed silence), etc. For instance, the user may require access to information such as weather forecasts, route directions, on-going traffic situations or just current temperature updates, and therefore this requirement should be updated on a regular basis and made always available to users.
- **Time context:** it provides information about the time and the date on a daily, weekly or monthly basis.

As we shall see in the sequel of this thesis, CI plays a major role in controlling CAS behaviour [31], [46].

2.2.3.2 Context Aware System (CAS)

The idea of CAS was first theorised by Mark Weiser in his paper 'The Computer for the 21st Century' [114]. Weiser also provided us with predictions for context-

aware computing: “computers will come invisible to common awareness. People will simply use them unconsciously to accomplish everyday tasks”.

Weiser professes that computers will be location aware, will have the ability to capture and retrieve information, and will offer seamless interaction in supporting tasks. Context-aware computing as we understand it today was first presented by Schilit and Theimer [101]. They postulated that “CAS is about exploiting the changes in the environment in which mobile and distributed applications run, therefore, the system should have the ability to adapt to changes such as user location or connecting host over a period of time” [86]. Moreover, CASs should have a monitoring mechanism that can react to changes in the environment [106].

According to Sheng and Benatallah [104], the system becomes aware once it can sense information originating in a particular context and can profitably use it to provide services to users. However Dey and Abowd give a very influenced definition of CAS [30];

“A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task”.

It was stated above that, the context of the user can be seen as information about the user and their environment [100]. Therefore due to the mobility of both user and device, the context of the user is not steady but is changing irregularly.

2.2.3.3 Context Aware System Lifecycle (CASLC)

A Context Aware System Lifecycle (CASLC) can be mainly classified into three stages, namely Acquisition, Reasoning and Acting stages [84], [71], [82], [2], [43]. Each stage has distinctive behaviour and special duty. The various stages are pic-

tured in Figure 2.4, and are discussed below.

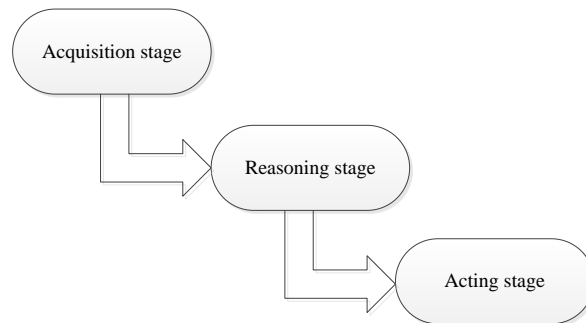


Figure 2.4: Context Aware System Lifecycle (CASLC)

- Acquisition stage: in this stage the system senses the surrounding environment and then captures a raw data or information about the physical world or some aspects of it. In this step, CAS collects CI from the provider of CI and then passes it to a CI repository for further reasoning.
- Reasoning stage: this is metaphorically called the thinking stage but, it is also known as context manager. In this stage, the data collected by the CI provider is processed and analysed in order to make it understandable and usable for the system to take an action upon it. Reasoning mechanisms enable applications to take advantage of the available CI. The reasoning can be performed upon the basis of a single piece of CI or of a collection of such information.
- Acting stage: this is also known in some studies as context consumer; this stage provides the physical world with the result of processing the gathered data.

These stages are fundamental to the study presented in this thesis because they will help to clarify the modeling of a CAS using the UML diagrams.

2.2.3.4 Context Aware System Security

Security is a vital issue in CAS [79], hence and needs to be considered in depth. Tracking security requirements of CASs relies on the explicit definition of the system environment [11]. This is a necessary prerequisite to allow the system to adapt to the new security settings following a change of context parameters such as location.

Context aware security is defined as “dynamic adaptation of the system security policy according to the context” [73], [79]. The reason for asserting that CASs need more security than other systems is related to the nature of CAS communication; it is wireless and so the data are radiated to anyone within a range, whereas ordinary systems use cables that circulate the data only to authorised persons [15]. Therefore, many researchers have developed and collectively agreed upon certain steps that must be undertaken in an effort to implement the most efficient and effective level of computer security that is possible for applications running in a CAS environment. The main purpose of these steps is to ensure that the level of risk is reduced as far as possible.

The security requirements will be discussed in order to determine the security steps that need to be taken in order to provide reliable security to CASs and applications. The sequel of this section describes the security requirements that this thesis identifies as the crucial once to minimise the risk related to CASs [79]. It is convenient to term them the ‘key security requirements’ for future references throughout this thesis [4], [69], [76], [112], [118], [53].

Authentication is the process of verifying the identity of the entity that is seeking to access the facilities of CASs and its applications. Authentication is essential for verifying the identity of each node in the system and its eligibility to access the network. The authentication function in a CAS is required only once; it occurs between the user device and the context-aware infrastructure

supporting the system. In practise, this means that nodes in CASs are required to verify the identities of all communicating entities in the network. This is needed at least to ensure that these nodes are communicating with the correct entity and that the user who is trying to access the CAS is eligible to do so.

Mutual authentication allows pairs of communicating nodes to get evidence on each other identity. Furthermore, in order to validate and run the Authentication process, it is not always required that the user or device be connected to the application; the authentication process could be done in terms of the user's context, e.g. location, Activity or nickname in an Internet Relay Chat (IRC) room.

Authorisation in practice often follows authentication; it manages user privileges by retrieving the permissions associated with the authenticated user within a given policy. Therefore, the authenticated user should have permission to access the required resource. In a CAS, the CI is utilised to manipulate the user privileges. Consequently, the authorisation behaviour must be dynamic in order to address the context change. For example, a university may control access to its resources, such as a library, depending on circumstances, and authorisation can also occur between two contextually aware devices, such as mobile computers.

In the first example, the context is known, and authorisation is easily implemented; the administrator of the infrastructure can therefore define the authorisation policies in terms of these known contexts. In the second example, it becomes more difficult, as users of contextually aware devices move around their environment; they may be involved in several contexts, and may wish to implement authorisation (based on these contexts) to the data they hold. These devices may 'swap' new authorisation policies depending on their

context and on the wishes of the associated users.

Confidentiality (sometimes called privacy in this area) is related to protecting or restricting the use of certain highly sensitive, private or secure information or data from being shared or being made available to anyone without permission. Each node in a CAS must secure both the data that is exchanged and the location information and other data that are stored on these nodes.

Although confidentiality has been vastly studied for traditional system, implementing this concept in a context-aware environment is a very challenging task. Most of the studies in this field have claimed that it is possible that users who wish to access and use the functionalities and facilities of CASs can be somehow drawn into sharing their contextual information.

In order to ensure that such users do not unwittingly share or provide access to their private information to other entities, certain methods and tools are needed to prevent sensitive information from being accessed by others, and the user's location can play a key role in this regard.

Integrity prescribes that the data transmitted between users in CASs should be received by the intended entities without having been tampered with or changed through any unauthorised modification. This requirement is essential, particularly in mission critical systems such as banking or aircraft control where data modification could cause considerable damage. Previous studies have shown that integrity of user's data can be ensured by guaranteeing that the data cannot be accessed and then altered by fake, illegal or invalid users. The integrity requirement in CASs is defined as a guarantee in which access to a user's resources is not allocated or assigned to any illegal or incorrect user.

Integrity instantiated to data is essential in CASs, especially when communication occurs between a CI provider and a mobile user, and also during

data exchange, whether the data are stored in the main server or in the user's device. Thus, transferring data is a fundamental issue in terms of integrity because it needs that data is received as it was sent to be guaranteed, particularly in a wireless environment. It therefore necessary to continuously trace and monitor the data flowing through the system.

2.3 Framework supporting CAS security and their limitations

Previous studies have proposed various frameworks for implementing efficient and effective security on user information in CASs. The main purpose of implementing these frameworks is to acquire and state various security-related requirements and models. However, each framework has developed its own means of implementing security requirements and models. For this reason, each separate framework is valuable in its own right and worth of consideration. The following subsections briefly describe the most distinctive frameworks to capture some set of security requirements within in the context-aware paradigm.

- **Confab Framework.** Users should be able to apply certain privacy and confidentiality controls, thereby protecting their private or personal information from illegal access by unauthorized users. Therefore, in an effort to fulfil this security requirement, Jason et al. [51] proposed the Confab framework. Specifically, this framework is designed to provide protection and reliable security for information that is relevant to the user's location in a ubiquitous system. Moreover, its working hierarchy is based on specific analysis relating to the basic privacy requirements of end-users and application developers [103]. For security purposes, the private information of the user is acquired, stored and

processed in the user's device, instead of being stored in some other device.

- Uniform Access Control (UAC). Covington et al. [26] came up with a uniform access control framework specifically for serving the purpose of environmental roles. Moreover, it has been declared and claimed as a further extension to the Role-Based.
- Uniform Access Control (UAC). Bardram et al [16] developed a uniform Access Control framework specifically for environmental roles and declared to be an extension to the Role-Based Access Control (RBAC) model. In the RBAC model, specific privileges or access rights for the user, in terms of accessing the system services, are linked with the environmental roles. More precisely, in this concept, a role can be a developer or a manager in a top-level domain. Moreover, a role is responsible for analysing and determining the security aspects relating to CASs and any applications in ubiquitous environments [74], [16].
- General Role Based Access Control (GRBAC). Ahamad et al. [74] highlighted the main difference between the RBAC and GRBAC models. The RBAC framework is a basic model that only covers the subject-oriented approach whereas GRBAC allows the definition of access control policies depending upon not only the subject but also on other essential factors such as object or environment.
- Gaia. A framework designed to assist in the building of 'smart space' applications [100], [14], [50] such as smart homes and conference rooms, consists of a framework for building distributed context-aware applications. Gaia's event manager service enables applications to be developed as loosely coupled components, and can provide basic fault tolerance by allowing failed event

producers to be automatically replaced. Gaia’s remaining services support various forms of context-awareness, which include the following ones.

- Presence service: it monitors the entities entering and leaving a smart space (including people as well as hardware and software components).
 - Space repository: it maintains descriptions of hardware and software components.
 - Context file system: it associates files with relevant CI and dynamically constructs virtual directory hierarchies according to the current context [93].
- Roman et al. [22] defined generic context-based software architectures for physical spaces such as Gaia. Moreover, they defined a physical space as a “geographic region with limited and well defined boundaries containing physical objects, heterogeneous networked devices and users performing a range of activities”. Depending upon this very physical space, active space providers assist the users of CASs and applications to directly connect and therefore interact with the physical space. Again, this framework is being used to cover the security requirement that is called Access Control. Its main purpose is therefore to define the giving permissions for authentic and real users to access or utilise the system facilities.
 - Kerberos. Kerberos is designed to achieve the purpose of fulfilling and implementing various security requirements in CASs and applications, such as Identification, Authentication and Access control. Kerberos, however, is a framework for which the center of attention concentrates on verification of the identity of the user who is requesting access to services and facilities of CASs. This verification can be done by making use of data relating to the user’s context which includes fingerprint, voice and face recognition, etc [22].

- Context Toolkit. provided by Dey [32], it introduces the concepts of ownership. The Context Toolkit makes it easy to add the use of context to existing non-context-aware applications and to evolve existing context aware applications. New components involved in this access control are the Mediated Widgets, Owner Permissions, a modified Base Object and Authenticators. Context Toolkit provides context widgets, and they can be used as reusable software components for accessing and interpreting context data while hiding details. Therefore, Context Toolkit can support various domains of context aware applications [68], [50].

FRAMEWORKS	SECURITY REQUIREMENTS
Confab	Authentication
UAC	Access Control
GRBAC	Access Control
Gaia	Authentication, Access Control
Kerberos	Authentication, Access Control, Privacy
Context Toolkit	Authentication, Access Control

Table 2.2: Relationship between frameworks and security requirements

Table 2.2 summarises the framework outlined above and the specific security requirements each of them can handle. It can be seen none of the existing framework allows for the representing of all the key security requirements selected above, namely Authentication, Authorisation, Confidentiality and Integrity.

2.4 UML variants supporting CAS security and their limitations

In spite of the several researchers' efforts in the field of context-aware applications aimed at introducing extensions of UML diagrams for various purposes, there has been no research using UML over CASs for the gathering and modeling of their security requirements.

In this section, we present a number of past and current researches into UML extensions whether utilised for specifying security aspects in general, or for modeling mobile computing aspects in wider context. Two main criteria were used to properly assess the strengths and weaknesses of previous research efforts; these criteria are:

- The ability of the proposed method in respect to model the context awareness and the mobility aspect of the system using UML notations and diagramming techniques.
- The ability of the proposed method to gather or specify security requirements of the system using UML diagram notations techniques.

Choi [23] defined context and context-aware service as both concrete and evaluative forms and introduced a requirement analysis process, context-aware Use Case diagrams, context-switch diagrams, as well as a dynamic service model based on those definitions. His work is outstanding, and the extended Use Case diagram looks also very intuitive. However, he did not show how CI can be presented. However, he did not clarify how CI can be presented. In addition he proposed more notations which make his approach to model a huge system very complicated.

Hannes [81] presented some ideas related to the context-driven Use Case creation process, and gave evidence for its advantages (compared with the goal-driven approach) using the example of the distronic function. He also noted the importance

of context in Use Case creation, and thus introduced a process to capture context situations. The main goal of such process is to bridge the gap between critical context situations and the user goal by specifying the situations in details. However, he did not depict CAS behaviour and the effect of CI on that behaviour.

Sheng et al. [104] presented a modeling language for the model-driven development of context aware web services based on UML, emphasizing how UML can be used to specify information related to the design of context aware services. Kang et al [64] extended their previous research [65] to the use of UML 2.0 Activity Diagram for modeling mobile agent applications, and discussed their computational models to capture agents.

In relation to security requirements using UML, Sindre et al. [107] presented a systematic approach to eliciting security requirements based on Use Case modeling, which represents threats or abuse scenarios the users do not want to happen and must be prevented or mitigated. Similarly McDermott and Fox [75] have adapted the UML Use Case diagram to capture potential attacks by unauthorised stakeholders, calling this extension Abuse Case diagram.

Jurjans [60] presented an extension (UMLsec) [61] that allows relevant security information to be expressed within the diagrams of a system specification during formalisation. Jurjans also developed an extended Use Case diagram for capturing security requirements; however, his approach was only designed to describe the situation to be achieved together with goals tree [60]. That approach is not well suited to this research as it cannot describe the behaviour of CAS.

Houmb and Islam [52] proposed a security requirements engineering methodology called SecReq. However, their work mainly aims at compensating the lack of security expertise in software development teams. Mariscal et al [85] proposed an approach to model security as a separate concern by augmenting UML with separate and new diagrams for role-based, discretionary and mandatory access control, and

providing visual access control diagramming techniques. Peralta et al [89] presented a technique to specify UML security stereotypes aiming to guide developers by annotating vulnerable model parts, and to support automatic security test case study generation.

Alghathbar et al [7] proposed a logic-based system (flowUML) to validate information flow policies at the requirement specification phase of UML based design. Alghathbar et al, introduced [8] an extension to the UML meta model with an access control policy constraint specification and enforcement module, business tasks and history log for method calls. The extension shows how access control requirements of an application can be modeled in the design phase.

By contrast, in a different study, [6] Alghathbar focuses on the representation of access control policies in the requirements phase of the lifecycle. In relation to this work, Alghathbar and Wijesekera introduced [5] AuthUMLs, which is a logic programming based framework that analyses static access control requirements in the requirements phase of the life cycle to produce a consistent, complete and conflict-free access control requirements.

Jurjens and Shabalin [62] developed a tool for the analysis of UML models against difficult system requirements. More precisely, they described a UML verification framework supporting the construction of automated requirement analysis tools for UML diagrams. The framework is connected to industrial CASE tools using Extensible Markup Language (EML) and allows convenient access to this data and to the human user.

Alfonso Rodrigue et al [90] presented an extension of UML 2.0 Activity diagrams which allowed security requirements to be specified in business processes. They denominated as Business Process Security (BPSec), is Model Driven Architecture compliant since it is possible to obtain a set of UML artifacts Platform Independent Model (PIM) used in software development from a secure business process model

specification (Computation Independent Model (CIM)).

Modeling Method	Models Context Awareness and Mobility	Models Security Requirements
Choi	✓	×
Hannes	✓	×
Sheng et al	✓	×
Kang et al	×	✓
Sinder et al	×	✓
McDermott and Fox	×	✓
Jurjens	×	✓
Houmb and Islam	×	✓
Pavlich et al	×	✓
Rodriguez et al	×	✓
Karine et al	×	✓
Alghathber et al	×	✓
Alghathber et al	×	✓
Alghathber	×	✓
Alghathbar and Wijesekera	×	✓
Jan Jurjens and Pasha Shabalin	×	✓
Alfonso Rodrigue et al	×	✓

Table 2.3: Summary of the above mentioned modeling methods based on the defined assessment criteria

Table 2.3 summarises our short accounts existing UML works that have variously CASs and some security requirements. It can be seen that none of them is specifically tailored to gather security requirements for CASs.

This thesis aims to develop a practical and comprehensive approach to both

model CASs and identify their security requirements using Use Case, Activity and State diagrams.

2.5 Chapter Summary

This chapter has addressed the critical aspects underlying the present. It has also shed light on UML by presenting an overview of it was presented, following which its diagram types (either behavioral or structural) were described. Then, this chapter highlighted system issues by grouping them into two: those related to security in general, and then those related to security requirements gathering.

This chapter also examined context, giving the main working definitions, and subsequently examined the notion of CASs, depicting it in detail and describing its security requirements. It then identified the key security requirements of CASs, namely Authentication, Authorisation, Confidentiality and Integrity. Finally, the relevant related works that have been conducted using extended versions of UML diagrams were assessed in order to identify the gap that this thesis intends to cover.

In short, this gap is the capability of treating all key security requirements of CAS.

UML is universal modelling language , which has been utilised and extended for many purposes. In both areas CAS and security requirements UML has been enhanced.

However there is no a single study devoted to model security requirement for CAS using UML. As result this thesis was aimed to enhance UML diagram types namely (Use Case, Activity and State), which have enabled them to model a security requirements at the early stage of developing CASs. These diagrams notations were enhanced which became applicable to model the behaviour of CASs by showing the effectiveness of CI in such behaviour, then latterly to model the key security

requirements.

Furthermore, in regard to the chosen case study that has been presented in this thesis is probably correct. The case study is modelled based on our proposed enhancements on Use case diagram, Activity diagram and State diagram in order to represent some scenarios.

Chapter 3

Enhancing the Use Case diagram to model CASs and gather their security requirements

Objectives:

- To describe the existing Use Case elements
 - To reveal the weaknesses that hinder the current Use Case diagram notations in modeling CASs
 - To extend the Use Case diagram notations to model a CASs
 - To gather security requirements using the extended Use Case notations
-

3.1 Introduction

Use Case diagram is a vital instrument for both modeling system functions and gathering security requirements; however, the existing Use Case notations are not adequate to describe and model the behaviour of a CAS, which is dynamic and should fulfill the key security requirements identified in the pervious Chapter: authentication, authorisation, confidentiality and integrity. As a result of this, the Use Case diagram elements need to be extended to address the behaviour of a CAS as well as to enable gathering its security requirements. Therefore, this Chapter aims to show how to extend and adjust the existing elements of Use Case diagram to address those needs.

3.2 Existing Use Case diagram elements

A Use Case diagram consists of three main elements that must always be considered in order to complete the system blueprint [75]. These are enumerated here.

- Actor: an Actor is any stakeholder, whether a person, an organization, a device or an external system, that interacts directly with the system [41]. Moreover, Actors thus define the roles that users can play, and they can be used to model any other element that needs to exchange information with that system. An Actor exists outside a system, and is not actually part of the system itself. Actors in Use Case diagrams are graphically represented as stick men, denoted underneath by their names [10].
- Use Case: a Use Case presents the essential system functions to be performed by the actors; it also describes how a user and a system interact in order to accomplish some defined goal [25]. Each Use Case constitutes a complete course of events initiated by an actor, and specifies the interactions that take

place between that Actor and the system. The Use Case itself is divided into two (similar) types: main Use Case, and dependent Use Case. A Use Case is usually represented as an oval, and the Use Case name either appears inside or outside that oval.

- Relationships: there are several types of relationships that may exist between an actor and a use case(s) or a use case and another use case(s) [94]. Table 3.1 provides a summary of these relationships.


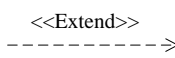
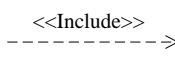

Relation name	Function	Notation
Association	The communication path between an actor and a use case where the direction of the arrow specifies who starts communication with whom.	
Dependency	The communication path between two or more use cases to invoke an additional behaviour based on a condition. Two stereotypes of this relationship exist in a use case diagram; extend and include.	 
Generalization	A relationship between two or more entities (actors) where the higher class entity provides a generalisation of the lower class entities.	

Table 3.1: Existing Use Case relationships

3.3 Enhancements

Use Case diagram is mainly used to define the functional requirements for system from user perspective, which is static in nature. In contrast a CASs functional

behaviour differs from other traditional systems as they are rapidly changeable based on context [28], [23].

It is not straightforward to communicate about, analyse, design, and implement systems. For example, customers and developers have communication problems in the requirement phase, and they may face this question: What context does the system function being developed need? [23].

The current form of Use Case diagram notations are mainly capable of only modeling traditional system functions, which are static, and hence the system design is not influenced by any external condition. In other words, all the system functions presented by existing Use Case diagram notations are not subject to any constraints, and so they are always available and ready to be performed when the user desires.

To support this, the example shown in Figure 3.1 depicts a student in an M-learning system invoking *<< Download materials >>* or *<< Do exam >>*. A normal Use Case diagram is represented with immutable functions that are not affected by any changes in context. This diagram is not subject to any other conditions.

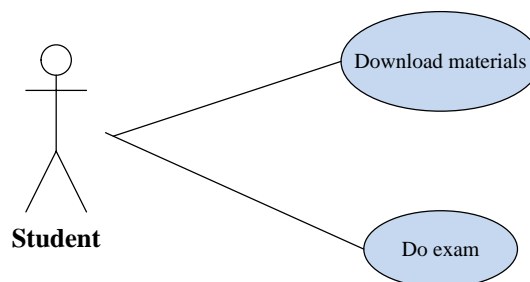


Figure 3.1: Normal Use Case diagram

However, this defies the basic nature of a CAS, which is adaptable to its envi-

ronment in a constant manner. Specifically, a function in a CAS may or may not be available and performed, as it is based on CI parameters such as time or place. The reason for this is that the function in CAS is subject to such CI, and thus once this context is fulfilled or satisfied, the function will be made accessible; otherwise it will be prohibited. Therefore, CI plays a critical role in determining the CAS functions, whose modeling necessitates the inclusion of CI in a Use Case diagram.

Accordingly, the existing Use Case notations need to be extended in order to support the description of context-aware features. What we mean here by mobility and dynamic is that the Use Case diagrams in a CAS should have the full ability to describe when the system functions can or cannot be invoked.

Therefore it is necessary to detail the shortcomings of each existing Use Case notation and to highlight why certain elements are unable to model a CAS.

- Actor

A context-aware actor is proposed to enable actor-oriented scientific workflow to be more personalised, adaptive, intuitive and intelligent; this is achieved by modeling the logic related to quality runtime adaptations [12]. In current Use Case diagrams, the actor is considered to be static, always interacting in a fixed environment and enjoying stable behaviour and attributes. More importantly, it is assumed that the actor is not affected by any effects in the surrounding environment; hence the current actor is always able to perform the relevant system function. In a CAS, the actor's behaviour should be treated as dynamic and adaptable to its current CI which, means that the actor can be influenced by the surrounding environment.

Returning to the example in Figure 3.1 for further clarification, in the M-learning system the current status context of the actor << *Student* >> is online; then, once s/he sits for the exam, and the system realises that the

location and time are for an exam, the student status will be switched to << *Offline* >> accordingly, which means that no other system functions may be accessed.

In short, the Actor in a CAS must have the capability of being aware of its physical environment or situation (context) and to respond proactively to environmental changes based on such awareness. Therefore, an enhancement that shows the dynamism of the actor in a CAS is required.

- Use Case

As explained above, CAS behaviour changes with changes in the environment. Therefore, we propose two new extensions to the existing Use Case, producing a new Use Case. One extension is that the Use Case (oval shape) is used to represent the system functions and to identify whether the status of these functions is optional or compulsory. It is impossible, however, to define the status of any function in a CAS as being either compulsory or optional, as such statuses are wholly dependent upon the CI. This idea was more extensively explained in Almutairi et al [11].

For example, the user << *Student* >> invokes the function << *Download materials* >>, which in a normal Use Case shape is considered to be a fixed function as it can be performed at any time and in any place. In contrast, a CAS will check the CI first in order to determine whether or not the environmental constraints will allow a function to be performed. Based on such an argument, there is a need to extend the existing Use Case notations to model the dynamic functions of a CAS, as well as to distinguish between the static and dynamic functions of the CAS. Accordingly, the normal shape is altered to a dotted shape represent function dynamism.

a Second extension stems from the observation that CI plays a key role in

deciding whether or not a particular CAS function can be invoked. Thus, our new type of Use Case, termed Context Information Use Case or (CI Use Case) for brevity, which is devoted to presenting the required CI that needs to be addressed in order to execute the Use Case functionality. The difference between a CI Use Case and a normal Use Case is that the newly proposed Use Case is not utilised to present any particular system function; rather, it depicts the CI that must be fulfilled to carry out the system functionality.

- Relationships

We have now proposed a new type of Use Case namely *<< CI Use Case >>*, and this makes it necessary to define the relationships that can serve. Therefore, we have developed a new type of relationship stereotype, called *<< Require >>*, which indicates the relationship between the CI Use Case and the main Use Case. This means that each Use Case requires a certain CI Use Case for it to be executed; therefore, this new relationship is used to link these Use Cases. The existing relationship stereotype types *<< Extend >>* and *<< Include >>* are not suitable for describing this kind of relationship because their behaviour is limited.

3.3.1 Extending the Use Case elements to model CASSs

This section is divided into two parts; the first part is conducted to fully describe the proposed extension of Use Case diagram to model a CAS. The second is devoted to capturing and modeling the security requirements for a CAS using the extended Use Case diagram.

As a result of the aforementioned shortcomings in Use Case modeling that was highlighted in previous section, a set of extensions are proposed to the existing Use Case modeling technique, enhancing its flexibility, so that it can cope with CAS

behaviour [11]. The essential idea behind the proposed enhancement is to show how the CI can affect the modeling of the CAS functions through a Use Case diagram.

This section continues by dividing the proposed extension that is to be added to the existing Use Case diagram notations into two parts, as follows:

- Adjusting the existing Use Case diagram elements
- Defining new Use Case elements

3.3.1.1 Adjusting the existing Use Case diagram elements

The existing notations are limited in functionality; therefore, it is imperative to adjust them to enable presenting the new behaviour. The notations that need to be refined are:

- Actor
- Use case

Table 3.2 depicts our notation adjustments. It can be seen that both the stick man and the oval are dotted in new notation.



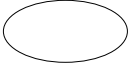

Existing notation	Adjusted shape	Description
		The dotted stick man indicates that the CI can have an influence on the actor, therefore the actor is able to be dynamic and adaptable in any a new environment.
		The dotted oval means the function availability could be influenced by changing CI.

Table 3.2: Adjusted notations

3.3.1.2 Defining new Use Case diagram elements

This is the most original contribution of this thesis, completing our extended notation that assists in presenting an outstanding blueprint for a CAS and then in capturing the relevant security requirements. The new elements can be seen in Table 3.3.

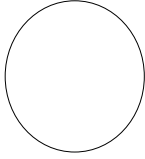

New elements name	Shape	Description
CI Use Case		It is used to define the required CI to perform some system function.
Require relationship		It is used to express the relation between the main Use Case and CI Use Case.

Table 3.3: The proposed new elements to model CASS and to capture their security requirements

Having refined some of the existing use case diagram elements, and having defined two new ones, let us consider an example and apply the proposed enhancements in order to demonstrate their effectiveness and considerable benefits. Let us consider the same example given above, and assume that there is required CI that must be checked first, as follows:

- Download materials: in order to invoke this service, the user must be within university boundary, say in location A. Additionally, the student status must be << *Online* >>.
- Do exam: in order to invoke this service, the student must be in a predefined location within the university, say in location B, and at a predefined time, say between 10:00 and 12:00. Additionally, the student status must be << *Offline* >>.

Thus, the extended Use Case diagram is enriched to reflect the CI as in Figure 3.2:

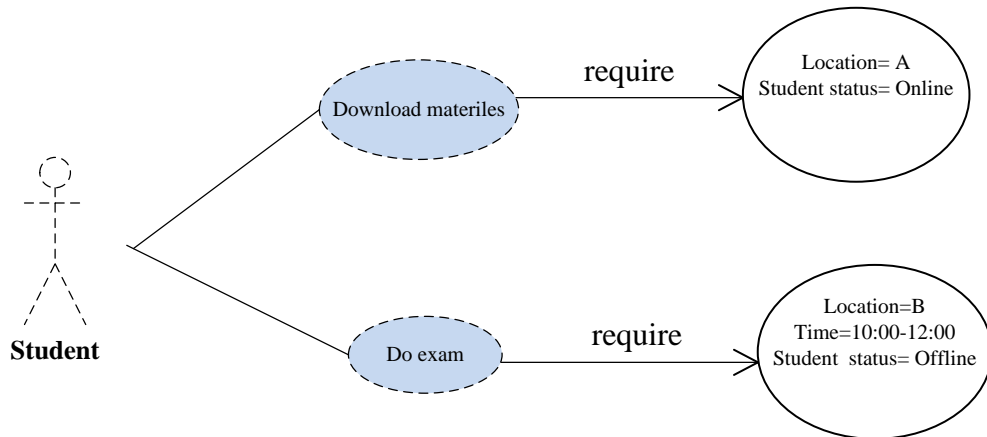


Figure 3.2: Extended Use Case diagram

From the new diagram, it can be clearly defined all the required CI; the function is not available unless this context is fulfilled. Having provided a clear explanation of the adjusted Use Case diagram, A Use-Case Specification (UCS) provides full textual details for a Use Case [18], as may be seen in the Table 3.4 for << *Download materials* >> function.

Name	Description
Actor	Student.
Main function	Download materials.
Brief description	The student can download the required materials.
Context Information	Location= A. Student status= Online.
Pre-condition	the student must be authenticated.
Post condition	The student must be having the proper CI to carry on downloading.
Flow of events	<ol style="list-style-type: none"> 1. The student requests downloading materials the invokes a function 2. CAS checks whether the student has gained the required CI 3. CAS provides the requested materials to the student

Table 3.4: Description of Use Case diagram

To sum up this section, were presented a number of changes to the Use Case notations to enable the use of Use Case diagram in order to model dynamic systems, which is precisely the nature of CASSs. These changes included the use of a new set of notations in the Use Case Model to enable the representation of CI and its effect on CAS functions. The Table 3.5 provides a compact comparison between the existing Use Case diagram, termed ‘Normal Use Case’ and our enhanced version, termed ‘Extended Use Case’.

Method name	Models system functionality	Describes static functions	Describes CI functions
Normal Use Case	✓	✓	X
Enhanced Use Case	✓	✓	✓

Table 3.5: Typical VS Adjusted Use Case Diagram

3.3.2 Gathering security requirements for CASs using the extended Use Case

Having provided a robust approach to modeling a CAS, we now need to gather the key security requirements. The Use Case diagram notations could be used to describe such requirements but is limited and found unable to model Integrity. Precisely, an Integrity prescribes to track information while it flows from an initial point to a final one, for example, user A sending a message to user B and, additionally, the receiver needs to know whether or not the data have been corrupted during the transfer. The nature of a Use Case diagram makes it unable to describe such requirement, and therefore we observe that a Use Case diagram is not capable of modeling all key security requirements.

Hence, in this section we concentrate on how to gather the other key security requirements, specifically Authentication, Authorisation and Confidentiality, for a CAS using our adjusted Use Case diagram. These observations are summarised in Table 3.6 which presents the security requirements that can or cannot be modeled in a Use Case diagram.

However, Integrity will be modeled in the sequel of this thesis, at the level of Activity and State diagrams, as we shall see.

Security requirement types	Use Case diagram
Authentication	✓
Authorisation	✓
Confidentiality	✓
Integrity	X

Table 3.6: Presenting security requirement using Use Case diagram

In the Section below, the adjusted Use Case diagram is employed to show how the security requirements for a CAS can be gathered and modeled. This will necessitate the inclusion of a few new elements in order to reflect the meaning of each security requirement. It is worth mentioning that the CI will also be the main factor in defining and managing the system’s security requirements.

3.3.2.1 Authentication

Authentication is the first stage in securing system in general [34]; it entails verifying the user identity by checking certain authentication parameters, such as user-name/password, location, time, etc.

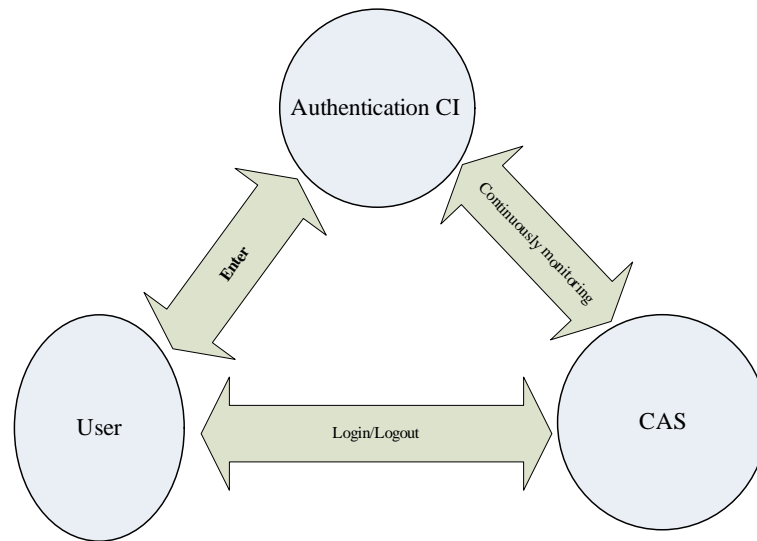


Figure 3.3: Authentication process in CAS

As stated in Chapter 2, the authentication process in a CAS differs from how it is carried out in other systems. It must be continuously re-iterated because the user environment is rapidly changing, which can possibly result in the user becoming unauthenticated by failing to satisfy the authentication parameters [70], [48]. Thus, in order to authenticate system users using Use Case diagram, we accordingly group two kinds of context parameter that must be addressed in verifying the user identity, as follows:

1. Static parameters: these are fixed details such as username/password and device IP address.
2. Dynamic parameters: these are changeable based on user context such as user location; this type is imperative, as the user needs to be continuously tracked to ensure their secure access to the system.

The authentication process can be presented in a Use Case diagram by expressing the login stage, which requires verifying both static verification details (such as

username/password) and dynamic ones (such as location,time etc..). For example in M-learning system the student has to have valid access details as well as be in the university boundary in order to access the system functions.

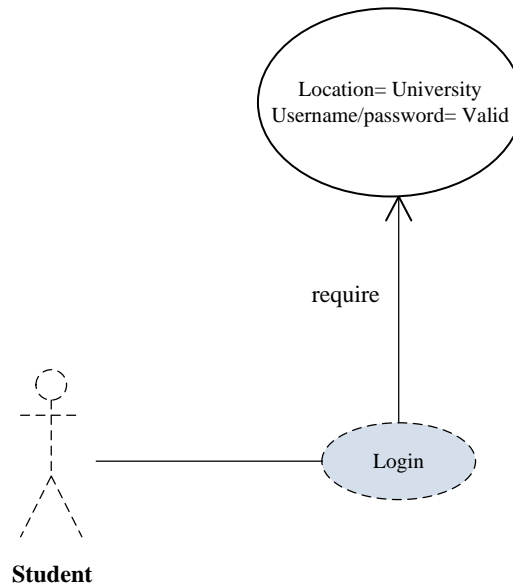


Figure 3.4: Authentication Use Case diagram

As shown in Figure 3.4, the student needs certain CI in order to pass the authentication stage to access and display the M-learning functions. The dotted shapes demonstrates how the diagram benefits from the enhancements we have described above.

Table 3.7 details how authentication information is captured; it describes all elements of the diagram.

Diagram elements	Description
Actor	Student.
Main function	Login.
Brief description	Describe the login process, which identifies the student.
Security type	Authentication.
CI	Username,password. Location.
Precondition	The student must be within the university boundary.
Post-condition	The student must not leave the university.
Flow of events	<ol style="list-style-type: none"> 1. Student enters username password 2. CAS verifies the student Location 3. CAS check all the provided CI 4. CAS displays the functions

Table 3.7: Description of Authentication Use Case diagram

3.3.2.2 Authorisation

Authorisation concerns permitting or denying privileges to users, and thus CI plays a major role in this. This implies that authorisation is particularly difficult to enforce within a CAS. This service follows the authentication stage. Although Use Case diagrams can visually present the behavioral system requirements, they are not

fully able to represent existing authorisation policies. At best, a Use Case diagram can show some authorisation by stating the roles that actors are permitted to invoke [6], [9]. Therefore, it is imperative to state through the extended Use Case diagram how the CI can impact on the system's decision of either permitting or denying an actor's access to certain resources.

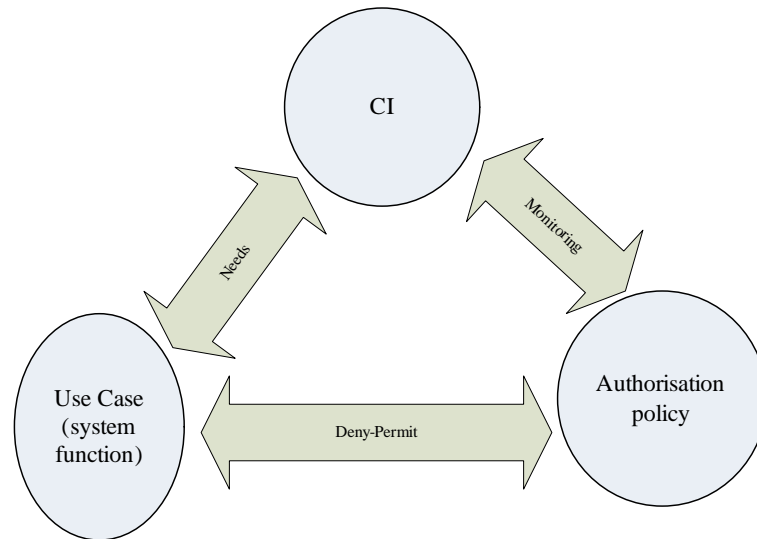


Figure 3.5: Authorisation process in CAS

The Figure 3.5 shows the impact that CI can have on the CAS function when presented by a Use Case. For instance, once the user decides to invoke an available service, s/he needs to provide certain CI and to fulfil certain constraints to be able perform that service; for example, in our running example of an M-learning system, the user << *Student* >> may want to download some files from the database; however, the downloading materials function requires certain CI and constrains to be in place, such as the user being in a particular location. Time may also be involved (e.g. between 9:00 and 17:00), and so once the student has fulfilled the necessary requirements, s/he can invoke the download function; otherwise, the service will be denied.

Thus, in order to present this, a new elements are defined for a use cases, namely << *Permit* >> and << *Deny* >>. Although these terms have been represented before as scenarios [6], they are adopted here without ambiguity, as use cases that are involved in the same Use Case diagram. This will help to reduce the complexity of Use Case description. We shall see that the performance of << *Permit* >> and << *Deny* >> use cases is namely their outcome totally dependent on the user CI.

The Table 3.6 describes the new use cases.


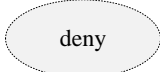
New Use Case name	Shape	Description
Permit		It means that the user has fulfilled the CI that is required for utilising the service.
Deny		It means that the current user CI is not consistent with the function requirements.

Table 3.8: The proposed Authorisation Use Cases

These new use cases << *Permit* >> and << *Deny* >> can be used continuously

(according to the CI). The example 3.6 depicts how the actor can invoke the service once the CI has been fulfilled, and vice versa (i.e. how he cannot if the user CI is not satisfactory). The reason for not providing as the shape of the classical use case is that *<< Permit >>* and *<< Deny >>* use cases are not part of the system functions, as they are considered as an action for process not a typical use case.

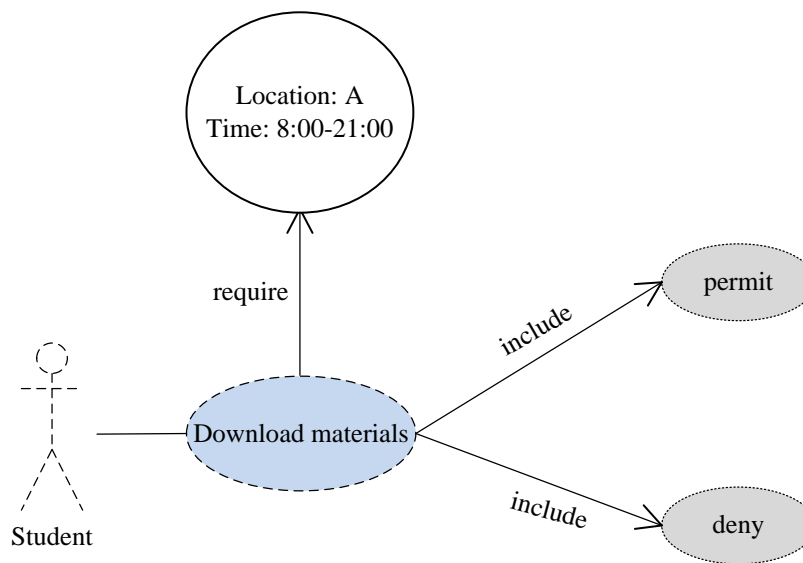


Figure 3.6: Authorisation Use Case diagram

The Table 3.9 explains the Use Case Authorisation in depth by providing all the details in the diagram:

Name	Description
Name of system user	Student.
Main function	Download materials.
Brief description	The student can access Download materials function once s/he acquires the required CI, therefore once the CAS approves these, the student will be permitted, otherwise will be denied.
Security requirement	Authorisation.
CI	Location: A. Time: 8:00-21:00.
Precondition	Student must remain having the proper CI.
Post-condition	Student must remain gaining the authorised parameters.
Flow of events	<ol style="list-style-type: none"> 1. Student requests download materials function 2. CAS verifies the student CI 3. CAS provides the requested materials 4. CAS continuously checks the student CI 5. CAS denies the service once the student CI is not applicable

Table 3.9: Description of Authorisation Use Case diagram

It can be seen that the authorisation mechanism can now be managed thanks to our enhanced notation and, in particular, the privileges for the user can be controlled by continuously monitoring the user context. In short, the chief benefits of our enhancement is that they simplify defining where and when a service can be granted

or denied in a CAS.

3.3.2.3 Confidentiality

Confidentiality is a significant requirement for a CAS and, as such, should be clearly captured in the Use Case diagram. It generally concerns protecting and hiding data from unauthorised persons or third parties. As explained above, each CAS function is inherently dynamic because it is granted depending upon the calling user's context information.

For the sake of demonstration, we consider a user who has been authorised to access a CAS function. We also assume that the function handles sensitive data, hence their confidentiality becomes an important requirement. Accordingly, the CAS will release the function to the user if and only if the user sits in a predefined safe location such as his/her office.

Thus, user location has been used as crucial context parameter [51] to influence the hiding of the data handled by the CAS function; as a result, user location will be exploited here to demonstrate how to model confidentiality in a CAS. It is assumed that:

the service has been already granted to the user, which means the user has satisfied the required CI. Thereafter, the system will continually check only the user location to control that service.

The Figure 3.7 shows the mechanism for protecting the data once the service is running.

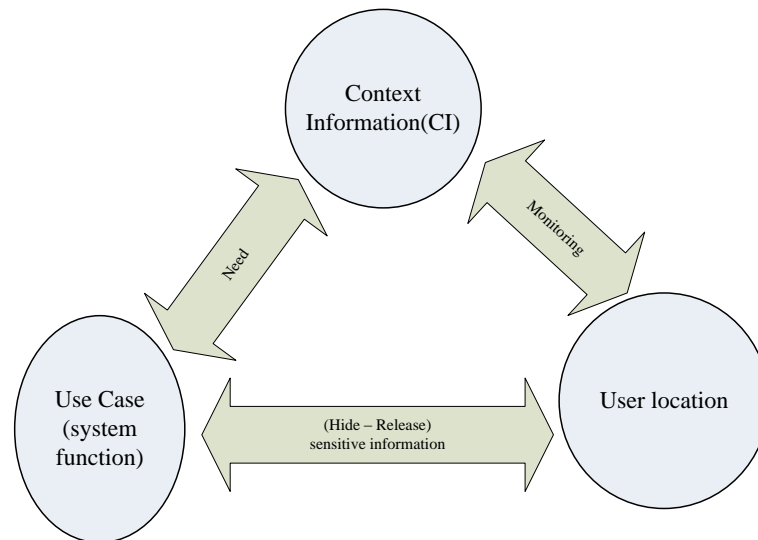


Figure 3.7: Confidentiality process in CAS

Consequently, the hidden data will be dynamically hidden upon the based of user location. In consequence, presenting confidentiality in a Use Case is slightly different from presenting other security requirements such as authentication and authorisation. As result, we propose a new use case called *<< Show information >>*, as depicted in the Table 3.10, to enhance the proposed Use Case diagram in order to clearly present confidentiality for CASs.

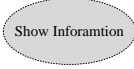
New Use Case name	Shape	Description
Show sensitive Information		Indicates that some of the data in the running function needs to be hidden.

Table 3.10: The proposed Confidentiality Use Case

To illustrate the above explanation with an example, we build the Use Case diagram in Figure 3.8. If a doctor wishes to check the record of a patient, the doctor must be located in his/her private office to be able to do the check successfully. Otherwise, the patient’s sensitive data, such as age and medical history, will be concealed.

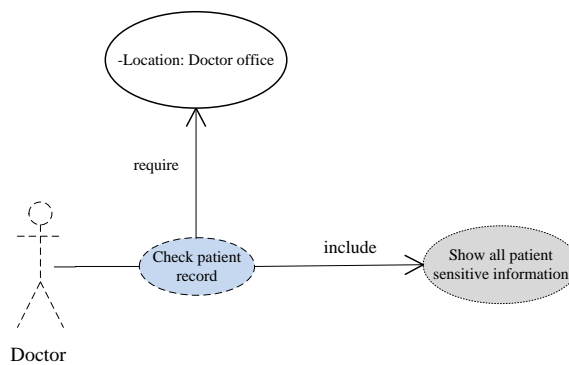


Figure 3.8: Confidentiality Use Case diagram

The Table 3.11 explains the Use Case Confidentiality in depth by providing all the details in the diagram:

Name	Description
Name of system user	Doctor.
Main function	Check patient's record.
Brief description	Doctor checks the patient's record.
Security requirement	Confidentiality.
Private location	Doctor office.
Sensitive Information	patient's age. patient's medical history.
Context Information	Location.
Precondition	Doctor must be located in his/her office.
Post-condition	Doctor location must not be changed, otherwise the patient sensitive information will be hidden.
Flow of events	<ol style="list-style-type: none"> 1. The Doctor invokes a function 2. CAS checks the Doctor location 3. CAS hides any sensitive user information once the Doctor location is deemed not private

Table 3.11: Description of Confidentiality Use Case diagram

3.4 Chapter Summary

In this Chapter, the researcher has studied how the Use Case notations can be made applicable to modeling a CAS, and then how the key security requirements can be gathered.

The Chapter started by detailing the proposed enhancements to the Use Case notations, which can greatly enrich their capability in modeling CASs. Then, this Chapter provided examples using our enhancements to depict how CI can affect the behaviour of CAS function.

Then the Chapter described the approach taken to gathering the security requirements, specifically authentication, authorisation and confidentiality, by utilising the extended Use Case diagram. Furthermore it clarified why the integrity could not be presented through Use Case diagram.

In addition this Chapter showed how the CI can play a major role in securing a CAS, either in the authentication stage, which can use static parameters and certain dynamic once, or in the authorisation stage, which can be managed by some CI to control the user privileges, and finally in the confidentiality stage, which exploited user location in order to modulate the release of data.

In short, this Chapter has presented our innovative framework to model CASS and gather most of their key security requirements by Use Case diagram.

Chapter 4

Enhancing the Activity diagram to support the extended Use Case diagram for CASs and gather their security requirements

Objectives:

- To describe the existing Activity diagram elements and their weaknesses in modeling CASs and their security requirements
 - To extend the Activity diagram notations to model a CAS
 - To use the extended Activity diagram to further demonstrate the extended Use Case diagram of the previous Chapter
 - To gather security requirements using the extended Activity diagram
 - To specifically target the Integrity requirement, which could not captured at Use Case level
-

4.1 Introduction

This chapter aims to describe how the Activity diagram can be used to develop a unique framework for modeling CASs. The main purpose of utilising an Activity diagram is to clarify each Use Case scenario. This will assist in demonstrating the value of the extension advanced on Use Case diagram and described in the previous chapter.

4.2 Existing an Activity diagram elements

For completeness of the presentation, we begin by outlining the main elements of Activity diagram. Typically well known to software developers, these are [24]:

- Initial (start) node: indicates the beginning of a workflow in an Activity diagram; it is drawn as a solid black circle.
- Control flow (Transition): an arrow showing the direction of the process between two or more actions or elements in the Activity diagram. It is drawn as a solid line with an open arrowhead.
- Action state (Activity): a model element that represents the performance of a task in the workflow or an operation in the process. It is presented as a capsule-shaped rounded rectangle with a name or description.
- Decision: a choice for a workflow to proceed along one of a number of possible paths, according to the guard conditions, which has only one incoming transition and has multiple outgoing transitions. It is presented as a diamond shape.
- Swimlane : it divides activity diagrams into sections. Each swimlane is separated from adjacent swimlanes by vertical, solid lines on both sides.

- Merge: a number of flows leading to the same Activity. It is indicated as a diamond with several flows entering and one leaving.
- Synchronization: a facility for the modeling of simultaneous workflows. It is of two different types:
 - Fork: this is presented as a bar with one incoming transition and two or more outgoing transitions.
 - Join: this is presented as a bar with two or more incoming transitions and only one outgoing transition.
- End (final) node: indicates the end of the workflow. The end state is drawn as a filled circle inside a large unfilled circle. The name on it is optional.

4.3 Enhancements

As stated previously, a CAS depends totally on the CI and originating in a certain context. In contrast, one of the advantages of an Activity diagram is that it can show the sequence and conditions for action execution [27], [24]. Moreover, an Activity diagram can also be used to specify the behaviour of parameters as well as the conditions that control each function (based on the gathered CI).

Therefore using an Activity diagram to model the behaviour of a CAS is incredibly fruitful, as we shall see. Before we engage with innovative extensions, it is necessary to evaluate the existing Activity diagram notation to assess whether or not it is applicable to efficiently present CAS behaviour. To the best of our knowledge, there seem to exist no studies conducting such assessment. We observe that using Activity diagram to model a CAS necessitates finding a suitable representation of object movement in order to make tracing and tracking the object in CAS possible.

More importantly, the notation ought to be able to show the influence of changing CI on CAS functions.

In consequence, we propose a general model notation by taking into account the sequential process of the object within a CAS (from the gathering and sensing CI stage, then passing through the processing stage, until reaching the acting stage in order to deliver the service). The behaviour of CASs will be depicted by presenting the main CAS action states as well as their functions, all of which must be possible in any CAS, regardless of their nature. In addition, it can be clarified who can carry out the system functions and where they can be carried out. In consequence, we prescribe that:

There are only two main swimlanes, namely << User >> and << Context Aware System >>, which describe the interaction between the system user and the CAS.

Each swimlane contains the possible functions that can be invoked through it. The user swimlane represents the functions that the user can do, whereas the CAS swimlane contains all the functions that the system can do and provide. For example, the << Login >> action state is only performed by the user, whereas << Get CI >> can only be done within the CAS. These can be seen in Figure 4.1, which illustrates several functions:

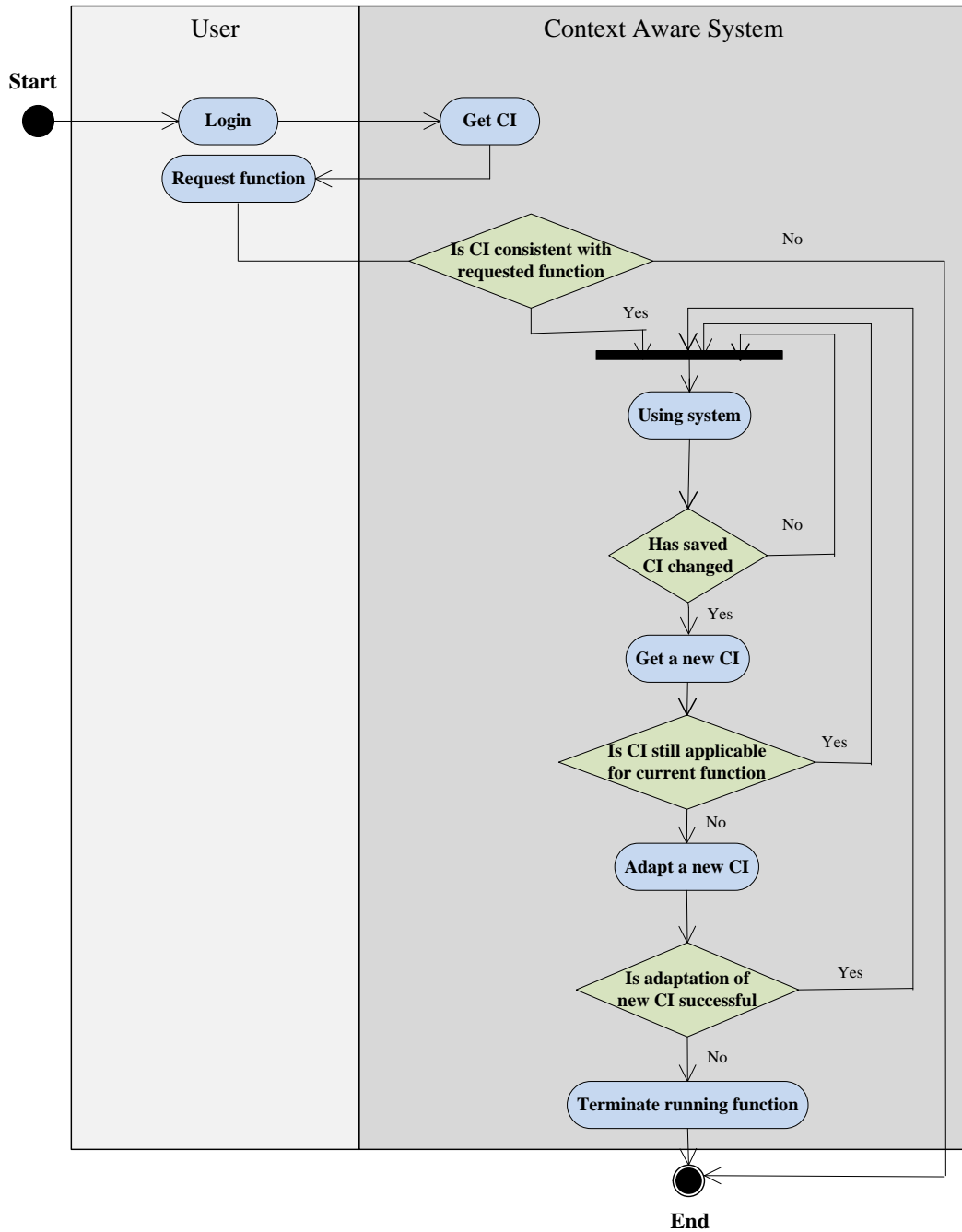


Figure 4.1: CAS Activity diagram framework

Figure 4.1 clearly shows the two main swimlanes, which represent the interac-

tion between the system user and the CAS. Table 4.1 describes the CAS functions grouped by the swimlanes in which they can be performed.

User swimlane functions	Context Aware System swimlane functions
Login	Get CI
Using system	Is CI consistent with requested function
	Has saved CI changed
	Get a new CI
	Is CI still applicable for current function
	Adapt a new CI
	Is adaptation of new CI successful
	Terminate running function

Table 4.1: CAS swimlanes functions

We have now defined all the CAS functions; it is therefore now time to classify the CAS stages and to determine where each function can be run. For example, the *<< Get CI >>* action state is considered as a context acquisition (gathering) stage, which collects all the CI (upon which the rest of the system depends). On the other hand, the reasoning stage mainly controls all the CAS functions by checking whether or not the CI that has been gathered is applicable, which can be done through the condition *<< Is CI consistent with requested function >>*. Finally, and based on processing the user request, the acting stage yields the result, which is presented in *<< Using system >>*.

As we discussed in the literature review (Chapter 2), Context-Aware System Lifecycle (CASLC) is divided into three stages, as follows:

- Context Acquisition.
- Context Reasoning.

- Context Acting.

The reason for mentioning these types again here is to state that they will be exploited to demonstrate where/how data can flow in a CAS via an Activity diagram; as well as to define precisely where the system function is invoked. Therefore, we advance the following enhancement to depict the CASLC clearly.

- Sub-swimlanes: as explained above in the Activity diagram elements, there is a swimlane to present a certain separated system; however, in a CAS we need to divide that swimlane into three parts (sub-swimlanes) to illustrate the CASLC. The reason behind this is that the CASLC is not a different and separate system; rather, the three are all parts of one swimlane and are connected to each other (although each sub-swimlane does have a certain duty).

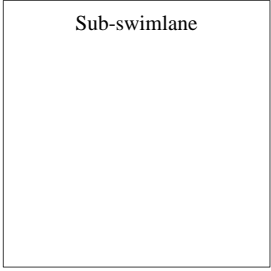
Notation name	Proposed notation shape	Description
Sub-swimlane		To state that each main system may have sub-systems inside, these swimlanes are not separated as each one is part of each other.

Table 4.2: CASLC swimlanes table

Figure 4.2 shows how the CAS swimlane is separated into three sub-swimlanes,

CHAPTER 4. ENHANCING THE ACTIVITY DIAGRAM TO SUPPORT THE EXTENDED USE CASE DIAGRAM FOR CASS AND GATHER THEIR SECURITY REQUIREMENTS

presenting its stages individually, with each stage containing the functions that can be executed in it.

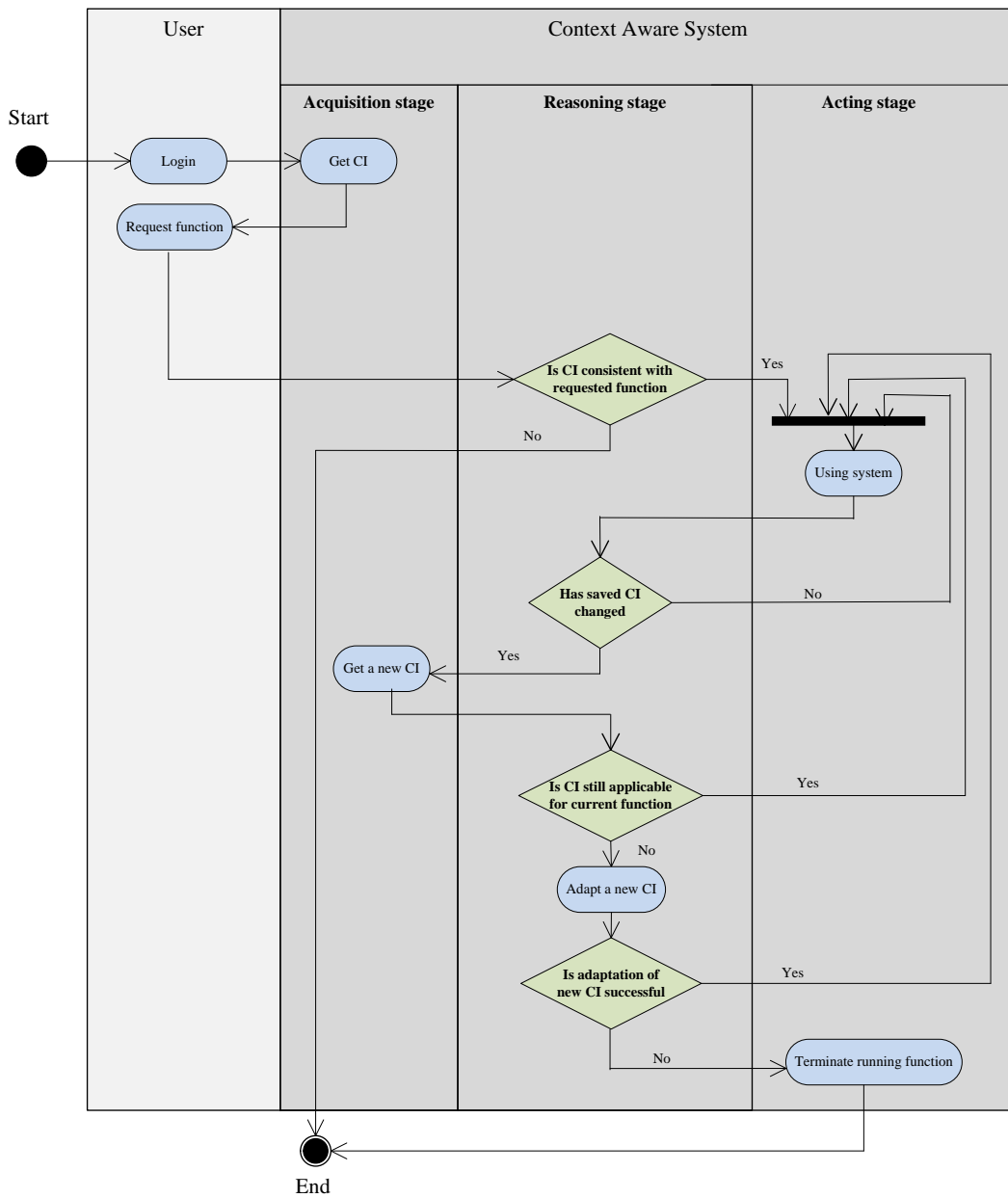


Figure 4.2: CASLC Activity diagram

It is worth noting that so far we have only modeled a CAS and its CASLC.

Adaptation mechanisms will not be considered in this thesis.

4.3.1 Extending the Activity diagram to support the extended Use Case diagram

To enable an Activity diagram to describe the extended version of Use Case diagram introduced in the previous Chapter, further extensions are necessary. These are as follows:

- Context Information Store (CIS): this contains all the gathered context information during function execution; this store is continuously checked and updated in order to ensure whether or not the user has applicable CI to invoke the required function. The CIS mechanism is considered as an internal part of the system, it is similar to a database, but it is very dynamic.
- Context links: these show how the context data can flow in an Activity diagram, and also depict how the decisions can be made. These links do not partake in object flow, and therefore they differ from the other links.
- Function Requirements (FR): this is to depict the required CI that must be fulfilled to perform the function.

Table 4.3 shows the proposed enhancements to the Activity diagram notations, which help to explain the extended Use Case diagram.


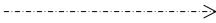

Notation name	Proposed notation shape	Description
Context Information Store (CIS)		Contains all gathered CI
Context links		This is not part of control flow; it is only to show how CI can affect system functions.
Function Requirements(FR)		This contains the required CI and constraints that must be addressed to carry out the function.

Table 4.3: Proposed notations to enhance the Activity diagram to present the adjusted Use Case diagram

Note that some relationships in the Activity diagram have been coloured to show both how the CAS works in detail, and how the system checks the condition before invoking any system function.

As shown Figure 4.3, the CIS is connected with the action state *<< Get CI >>* in order to store the gathered CI. This saved CI is then continuously checked in order to ensure whether or not the CI is consistent with the required or running

function. Each function has a certain CI, which is stated in the FR, and therefore once the user invokes a function, the system checks whether s/he has addressed the requirements for the selected function. That is achieved through the condition *<< Is CI consistent with requested function >>*; this verification is made in both the CIS and the FR at the same time. When a service is running, the system verifies that the CI has not changed after a limited period of time. If the answer to this verification is 'yes', the CIS will again capture the CI and update the CIS through the action state *<< Get a new CI >>*, and then the CAS rechecks the new CI against the FR to determine whether or not the CI is still applicable for continuing to use the service. This verification is done by *<< Is CI still applicable for current function >>*.

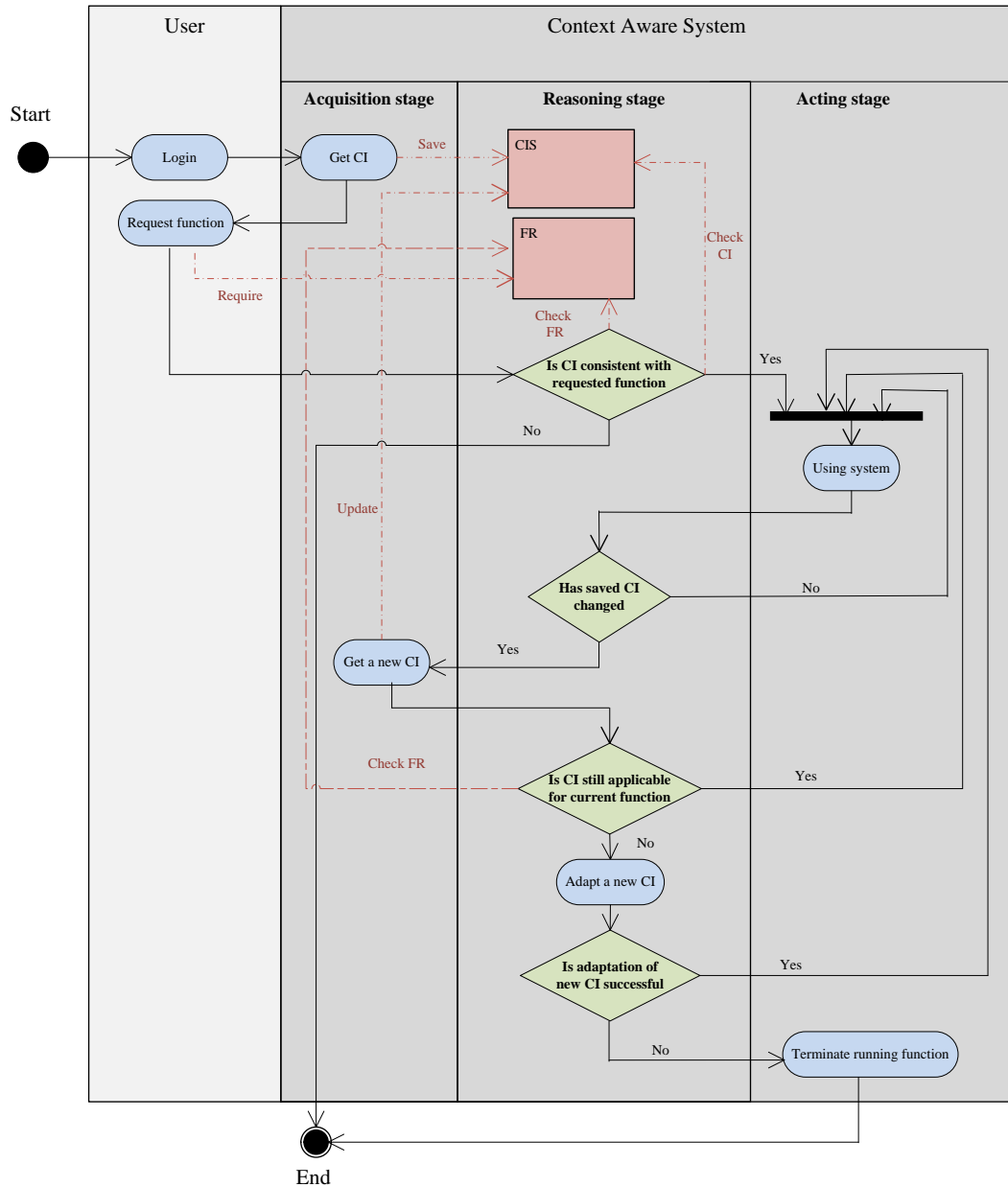


Figure 4.3: CASLC Activity diagram with proposed enhancements

As explained earlier, the CI is the key to extending the Use Case diagram in order to model a CAS. Therefore, the benefits of proposing both the new relation

`<< Require >>` and Use Case `<< CI Use Case >>` can be presented in the Activity diagram by showing the conditions that verify whether or not the current user CI matches the required function. Thus, once the required CI is satisfied, then the service/function will be granted accordingly. The gathered CI, as can be seen in Figure 4.3, is continually updated and checked.

First, let us have an example to illustrate what has been explained above in order to show the value of our proposed extension on the Use Case diagram in modeling a CAS.

Let us consider a library system: it is assumed that the Librarian wishes to utilise the library system to check the books that have been borrowed. The library system requires the Librarian to only invoke this service inside the library and between the hours of 8:00 and 21:00.

Figure 5.5 portrays the Use Case diagram corresponding to this example.

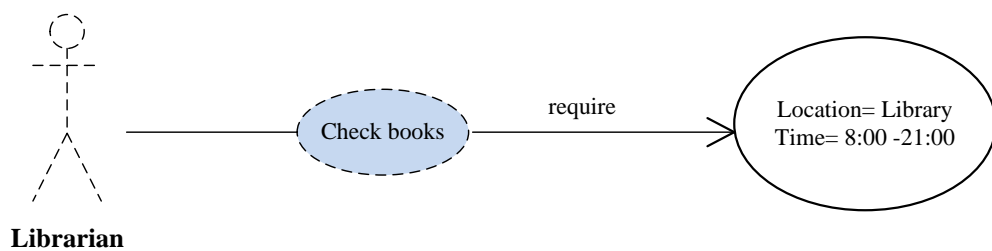


Figure 4.4: Check books Use Case

We can use our extended Activity diagram as shown Figure 4.5 to present the library Use Case scenario.

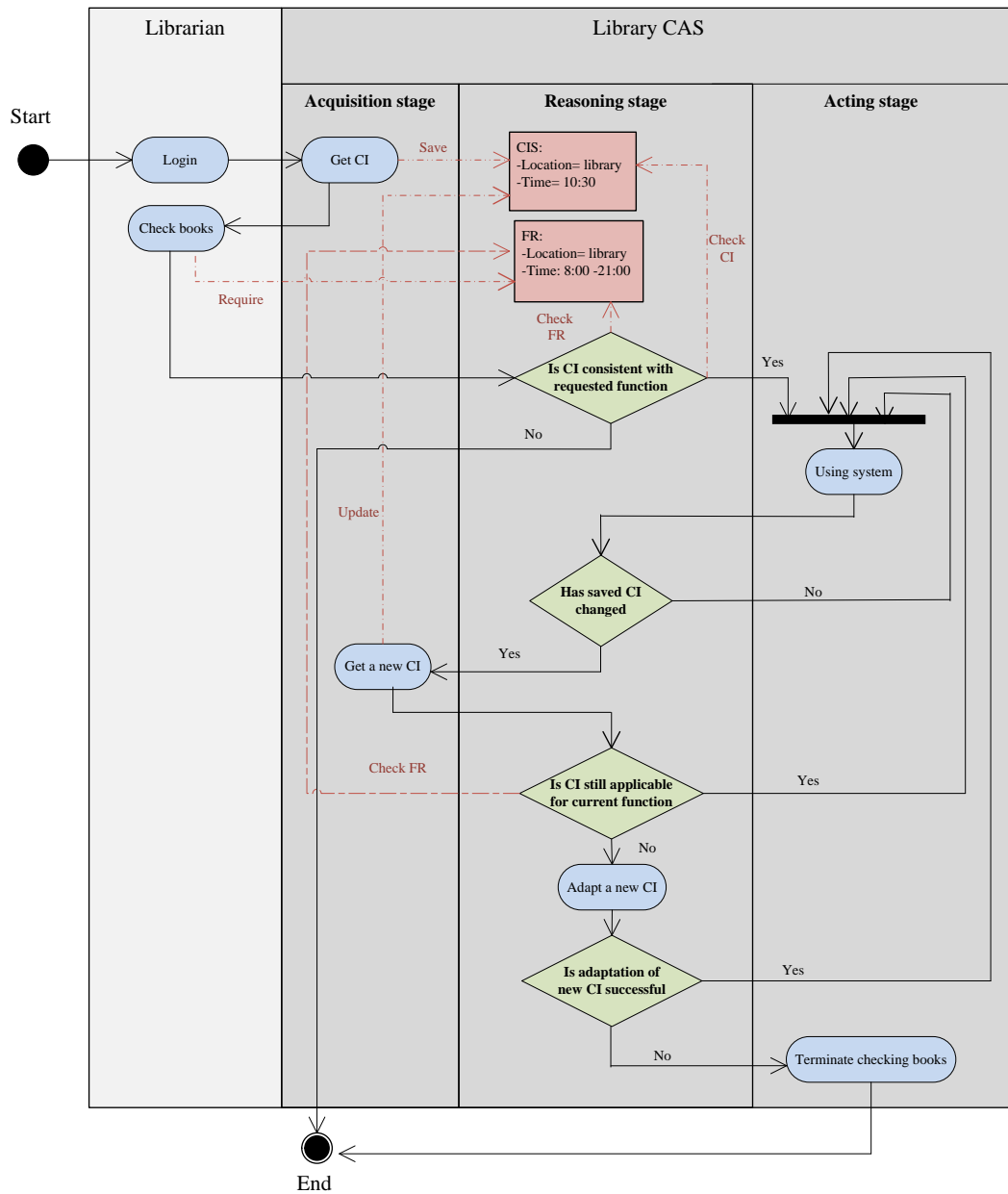


Figure 4.5: Check books Activity diagram

As it can be expected, the CI plays major role in controlling the *Check book* function. Thus, Figure 4.5 demonstrates the ability of the enhanced Activ-

ity diagram in modeling the Use Case diagram. Notably the impact of the changing CI on system functions is explicitly represented. This was not possible with traditional Activity diagram.

4.3.2 Gathering security requirements for CASs using the extended Activity diagram

Activity diagrams are generally used to show the system process workflow. This can assist in defining where the system security requirements can be performed. Therefore, we seek out to develop a notations to model the key security requirements also within an Activity diagram. We leverage upon the outcome of the similar effort put for a Use Case diagram in previous Chapter.

- **Authentication:** in the extended Use Case diagram, the login Use Case was exploited to present the process of capturing the required authentication parameters. Therefore, the Activity diagram will describe this through a set of action states that show how to authenticate and validate the CAS user's identity.
- **Authorisation:** this was presented in the Use Case diagram as a mechanism for showing when the function can be either permitted or denied. Hence, this process will be depicted here in an extended Activity diagram through a set of action states in order to manage the user authorisation process.
- **Confidentiality:** the extended Use Case diagram exploited user location to manage the hidden data, and therefore this will be fully illustrated in the enhanced Activity diagram; it will be done through a set of action states that are used to protect and hide any user-sensitive information.
- **Integrity:** this security requirement could not be captured in the extended

CHAPTER 4. ENHANCING THE ACTIVITY DIAGRAM TO SUPPORT THE EXTENDED USE CASE DIAGRAM FOR CASS AND GATHER THEIR SECURITY REQUIREMENTS

Use Case diagram because it needed an initial node and a final node, and the extended Use Case diagram did not provide such elements. However, we find out the Activity diagram is capable of presenting the integrity process; it can be done through a set of action states that are used to ensure that the data are not corrupted or modified by unauthorised persons or any third party.

Generally, one action state or a group of action states in an Activity diagram may present a certain type of security requirement (based on performance). For example, the << *Login* >> action state is adequate for expressing the authentication process, whereas we use a number of action states to explain the data integrity mechanism.

To present the key security requirements in an Activity diagram, we develop some extended notations. It relies on suitable icons, and is summarised in Table 4.4. The lock symbol is used to state the security concept, and two letters are used to identify the specific security requirement.





Authentication	Authorisation	Confidentiality	Integrity
 AC	 AR	 CO	 IN

Table 4.4: Icons used to present the security requirements in the extended Activity diagram

4.3.2.1 Authentication

The authentication stage in an Activity diagram can be presented by the << *Login* >> action state, which mainly concerns verifying the user identity. To check the user identity in a CAS, we may use several verification parameters together, such as static and physical ones (user name/password, device IP, etc.) and dynamic CI ones (location, time, etc).



Figure 4.6: Present Authentication by login action state

Thus, in order to express how the CI can be used to verify user identity as well as to fully spell out the authentication procedure in a CAS, we have therefore expanded the << *Login* >> action state; authentication stage passes through many action states, starting with << *Enter user details* >>, through verifying the user identity, until reaching the action stage << *Show system function* >>, as shown 4.7.

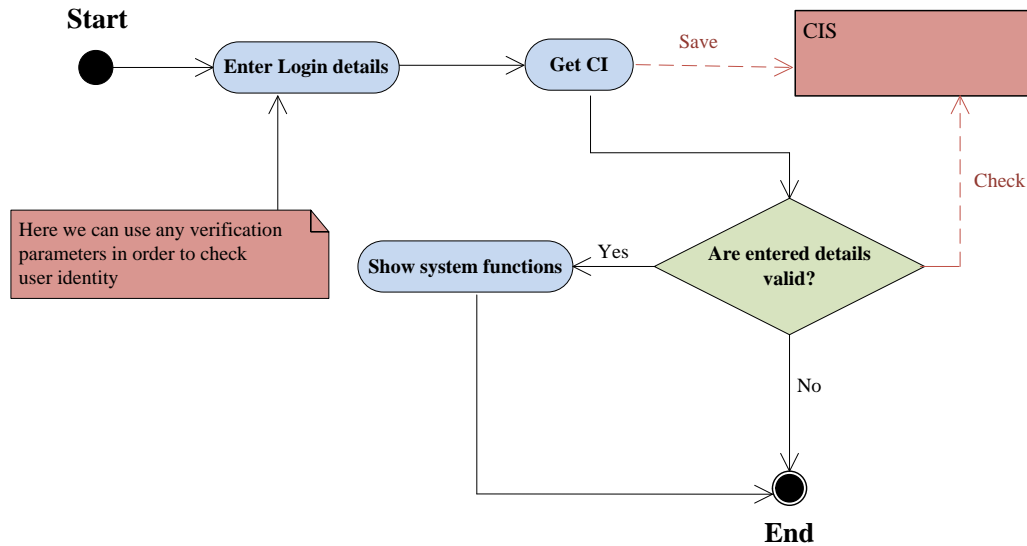


Figure 4.7: Expanded login action state

Figure 4.7 shows that, as the user enters the required details, the CAS gathers the user CI, and then all of this information is verified through the following condition *<< Are entered detail valid >>* in order to check whether or not the user is authorised to access the system. If ‘yes’, all the system functions will be displayed by the action state *<< Show system functions >>*; otherwise, access will be denied. However, authentication as we have just described needs yet to be positioned within a CASLC. This is done in Figure 4.8.

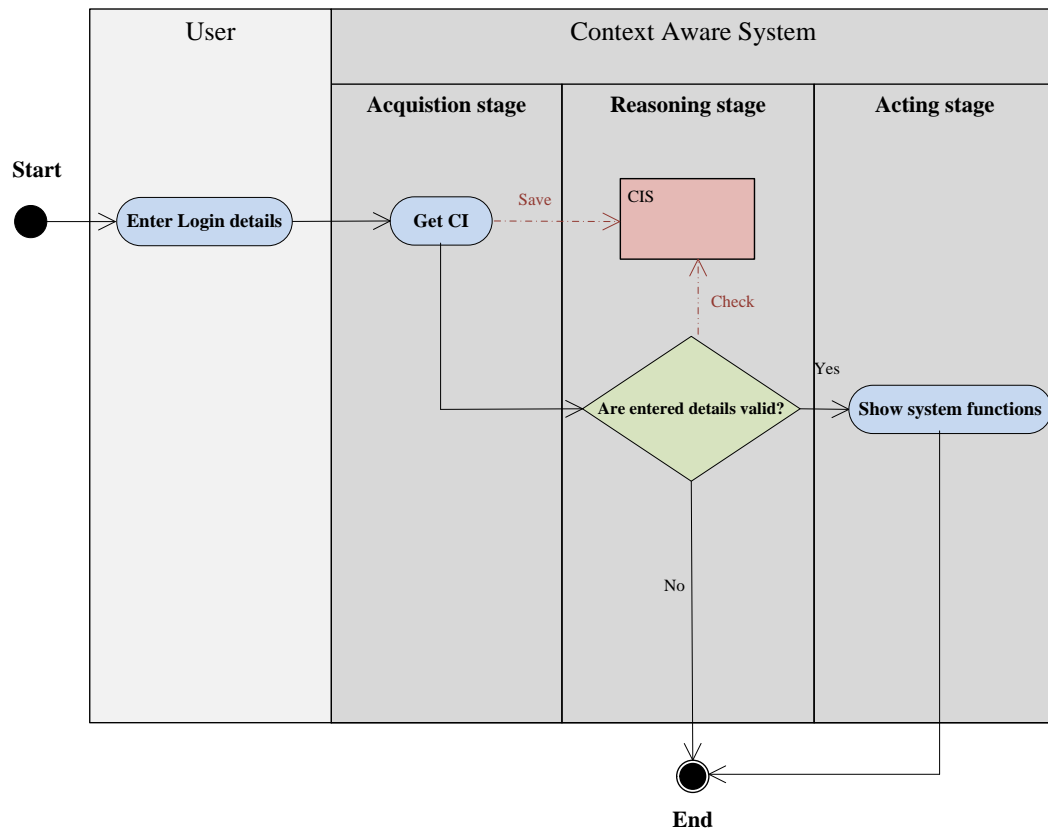


Figure 4.8: Authentication with CASLC

To further demonstrate our extended notation, we return to the liberian example and focus on the login phase. Its Use Case diagram is presented in Figure 4.9.

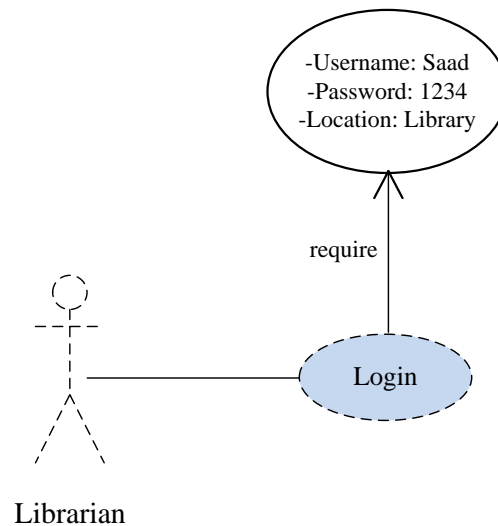


Figure 4.9: Library system login Use Case

The user << *Librarian* >> wishes to access the library system. S/he is required to enter the necessary authentication details, which are: username/password and a certain location (for example, within the library boundary). Then, once the details have been processed, the CAS checks whether the user has satisfied the necessary authentication parameters.

CHAPTER 4. ENHANCING THE ACTIVITY DIAGRAM TO SUPPORT THE EXTENDED USE CASE DIAGRAM FOR CASS AND GATHER THEIR SECURITY REQUIREMENTS

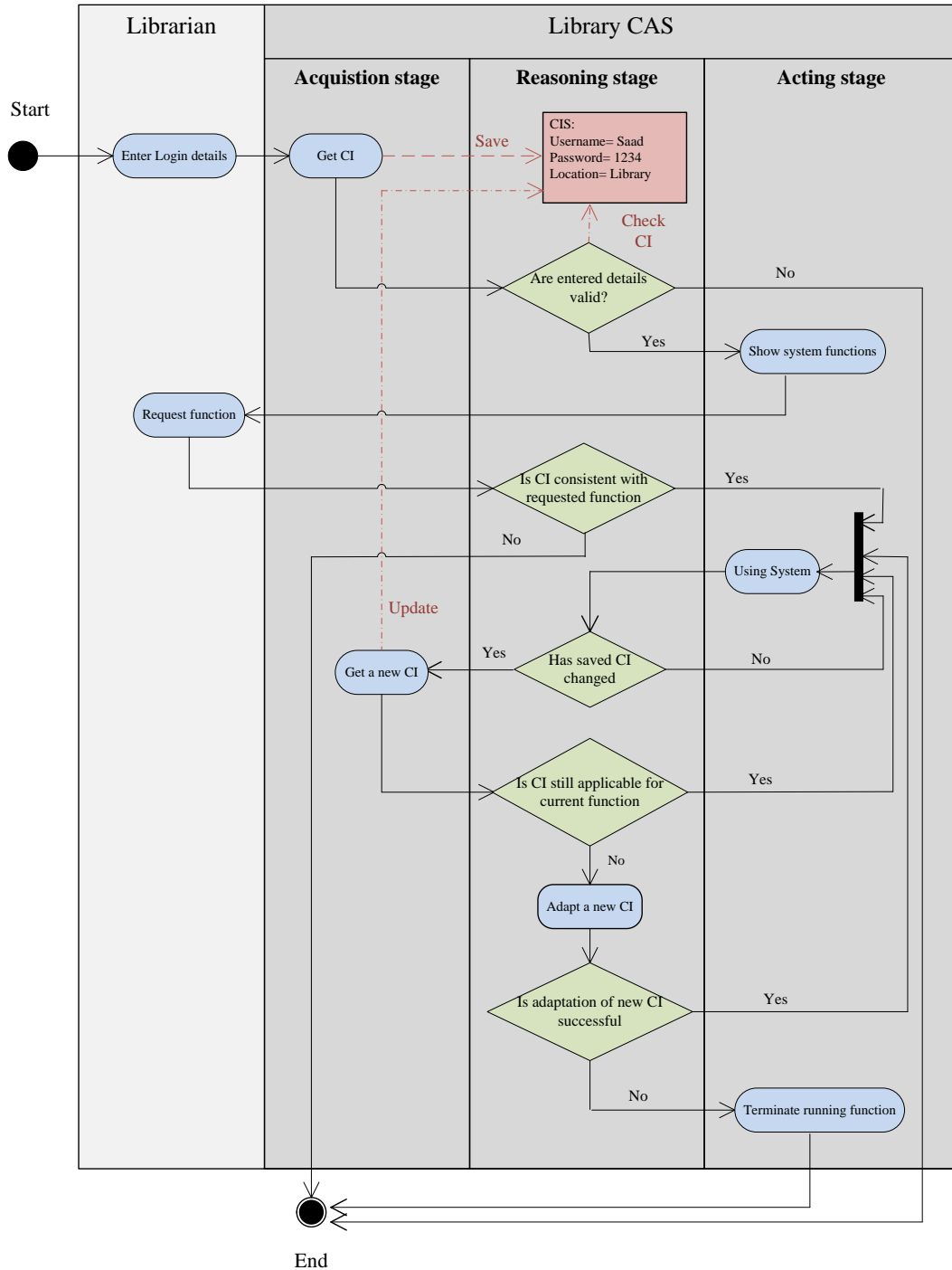


Figure 4.10: Librarian Authentication process

Figure 4.11 also demonstrates our notations for identifying the authentications process. Our dedicated AC icon has been stretched to cover all elements in the Activity diagram that pertain to this security requirement.

CHAPTER 4. ENHANCING THE ACTIVITY DIAGRAM TO SUPPORT THE EXTENDED USE CASE DIAGRAM FOR CASS AND GATHER THEIR SECURITY REQUIREMENTS

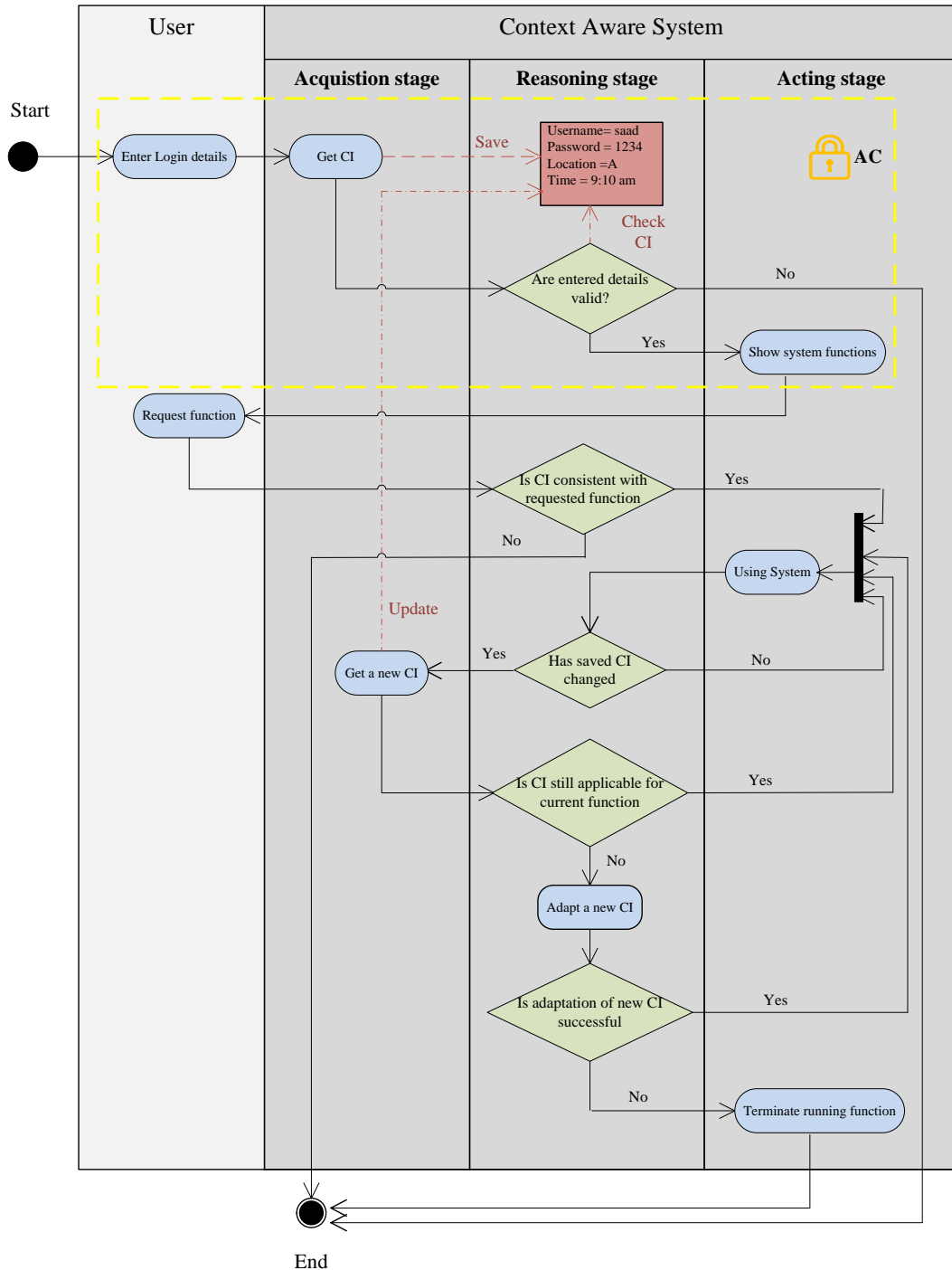


Figure 4.11: Determining the Authentication process during CAS

Having illustrated Authentication in detail, we will use only the login action state in the following for the sake of simplicity. This sharpens the focus on the other key security requirements.

4.3.2.2 Authorisation

Authorisation can be presented in an Activity diagram without adding additional elements to the proposed Activity diagram framework. Authorisation needs to be checked frequently and through various conditions in order to ensure that the user has full permission to perform the selected function. Therefore, user privileges must be dynamic and continuously monitored in order for the system to address the mobile nature of the CAS behaviour. As explained above, CI can manage the function privileges, which precisely describes the authorisation behaviour.

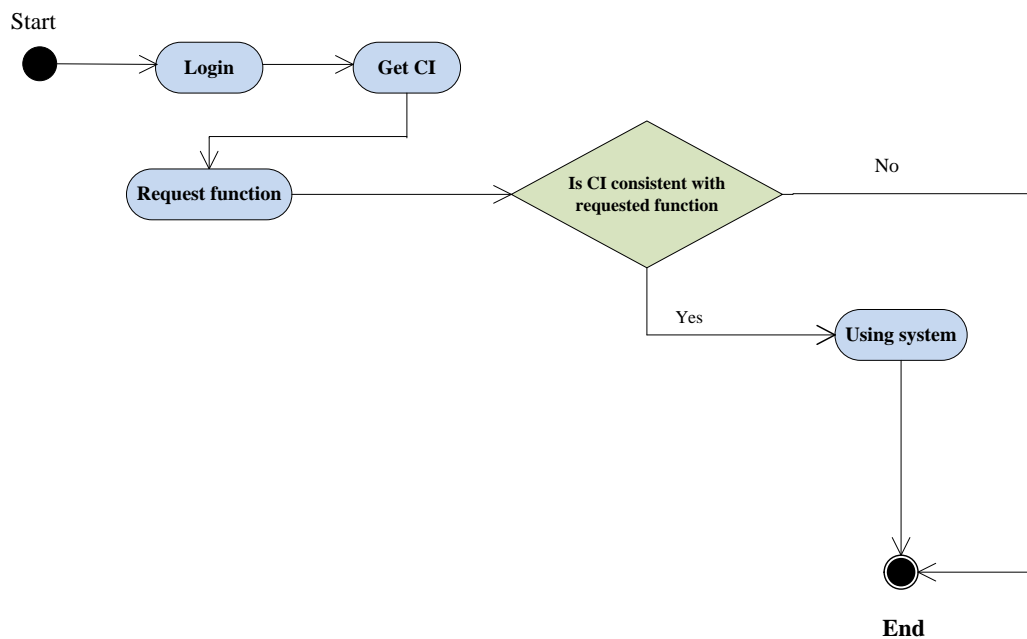


Figure 4.12: Basic concept of Authorisation mechanism

As depicted, authorisation process is required whenever the user requests any available system function. It is necessary to highlight the context-aware condition checker *<< Is CI consistent with requested function >>* ; this checks whether or not the required CI is adequate for invoking a CAS functions.

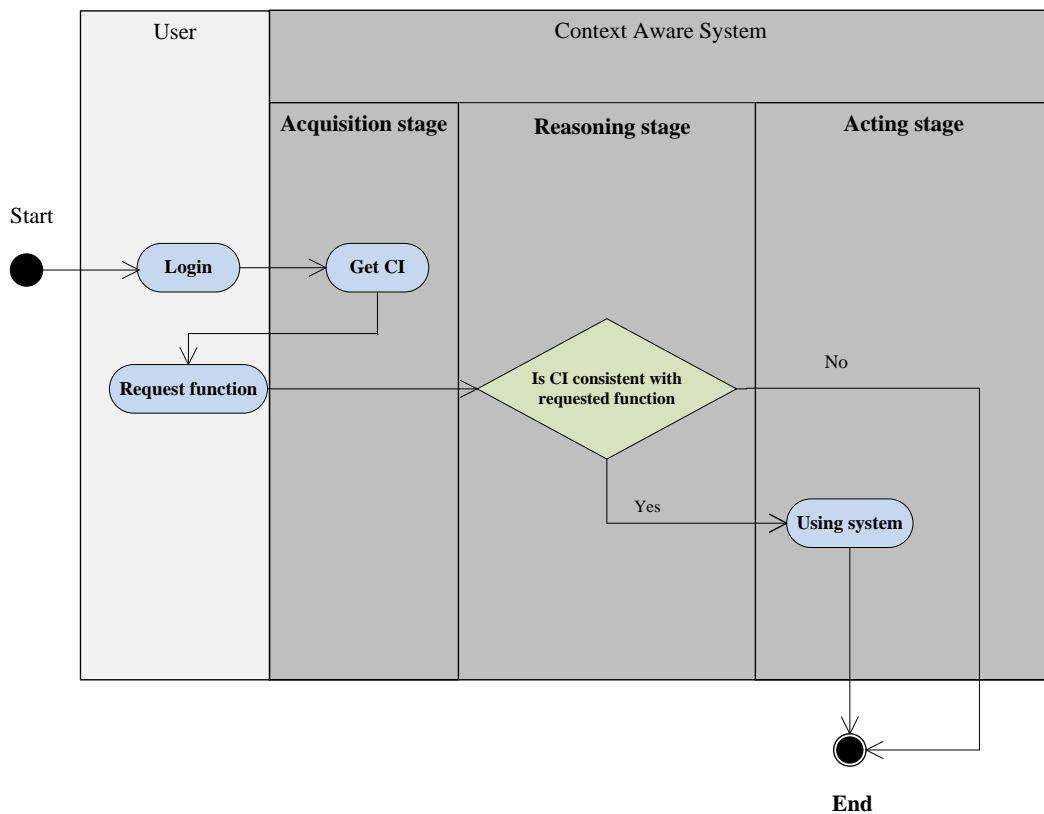


Figure 4.13: Authorisation mechanism with CAS

Returning to our main running example of library system, the librarian wishes to *<< Check books >>*; to do this s/he needs to address all the required CI, such as location (in our example, across the library) and time (between 8:00 and 21:00); these constraints would be compulsory for all students. We can represent this in an Activity diagram through *<< Is CI consistent with requested function >>* as in

Figure 4.14.

It can be seen that the authorisation condition occurs in the reasoning stage. Having defined how/where the authorisation procedure can be done, we can develop the Activity diagram that unveils the full details of authorisation within a CAS. This can be found in Figure 4.14.

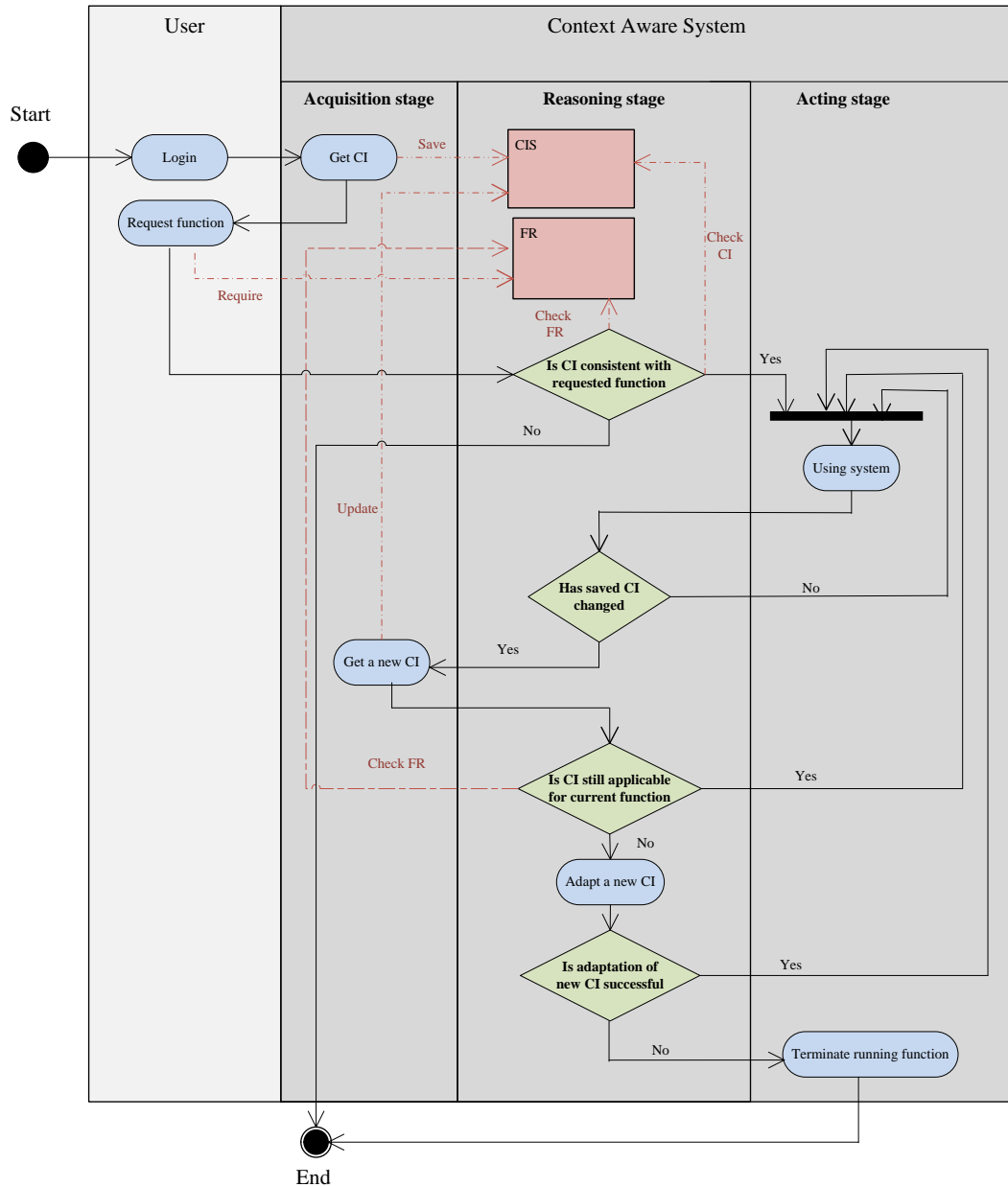


Figure 4.14: Authorisation within CAS Activity diagram

Figure 4.15 also demonstrates our notations for identifying the authorisation process. Our dedicated AR icon has been stretched to cover all elements in the

Activity diagram that pertain to this security requirement.

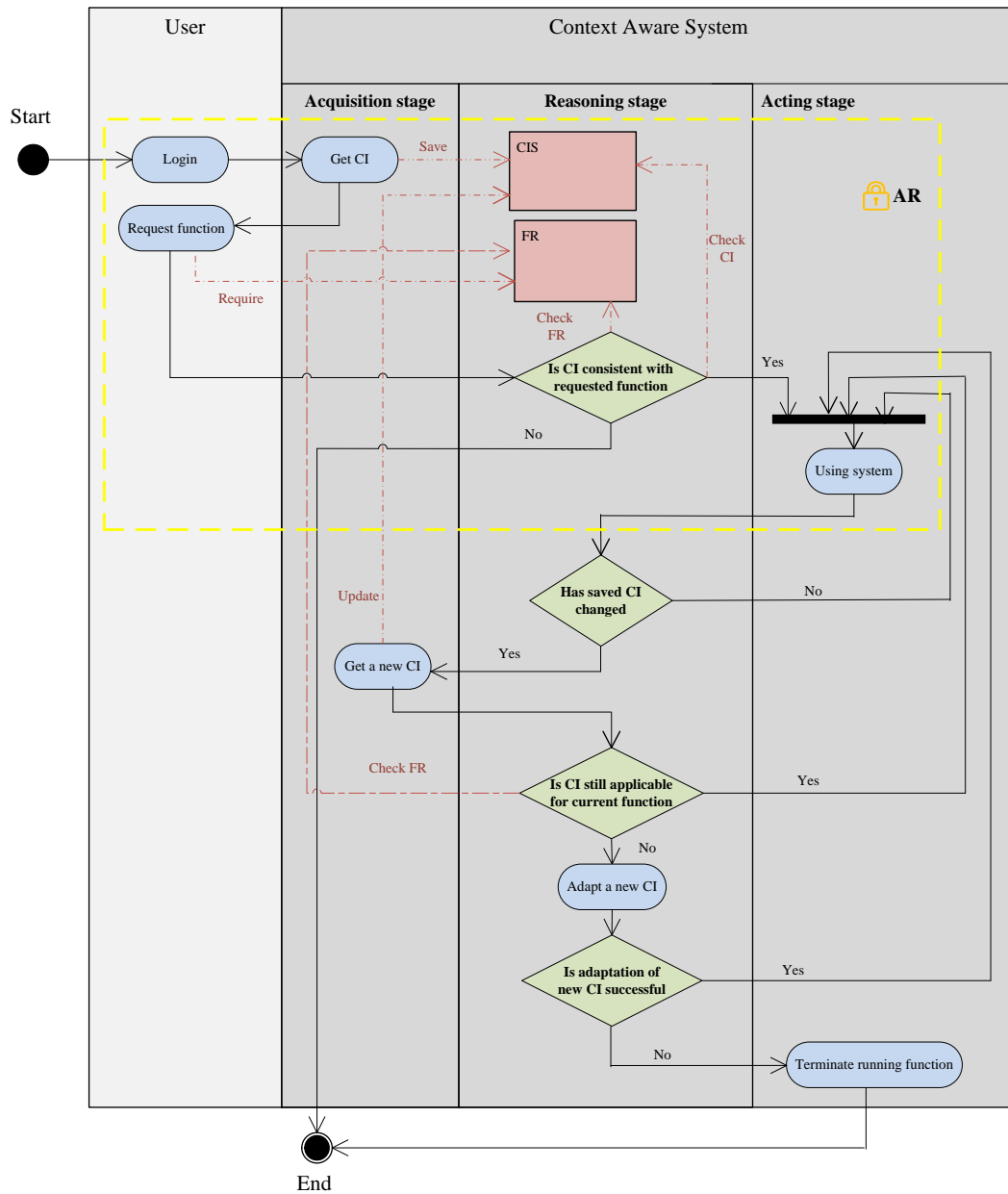


Figure 4.15: Determining the Authorisation process during CAS

4.3.2.3 Confidentiality

We have already discussed our approach of protecting potentially sensitive information on the user depending on the user location. Therefore by checking the location information, the system can control the decision over whether to hide or make available any requested service. Confidentiality in an Activity diagram is achieved through many steps. For seek of presentation, we assume that:

the service has been already granted to the user, which means the user has already passed authorisation. Then, the CAS in its turn will continually check only the user location to control that service. Most importantly, the CAS also checks whether or not the user is located in a safe place before running that service.

The Activity diagram 4.16 fully illustrates this procedure.

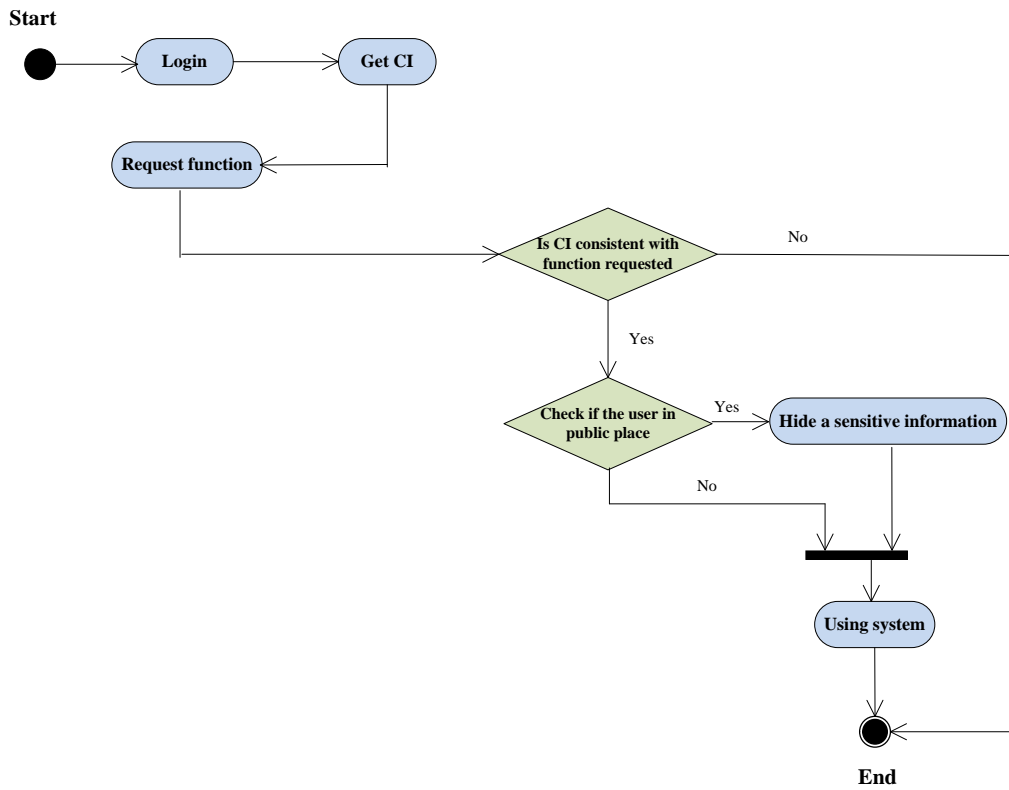


Figure 4.16: Basic concept of Confidentiality mechanism

Confidentiality as we have just described needs yet to be positioned within a CASLC. This is done in Figure 4.18.

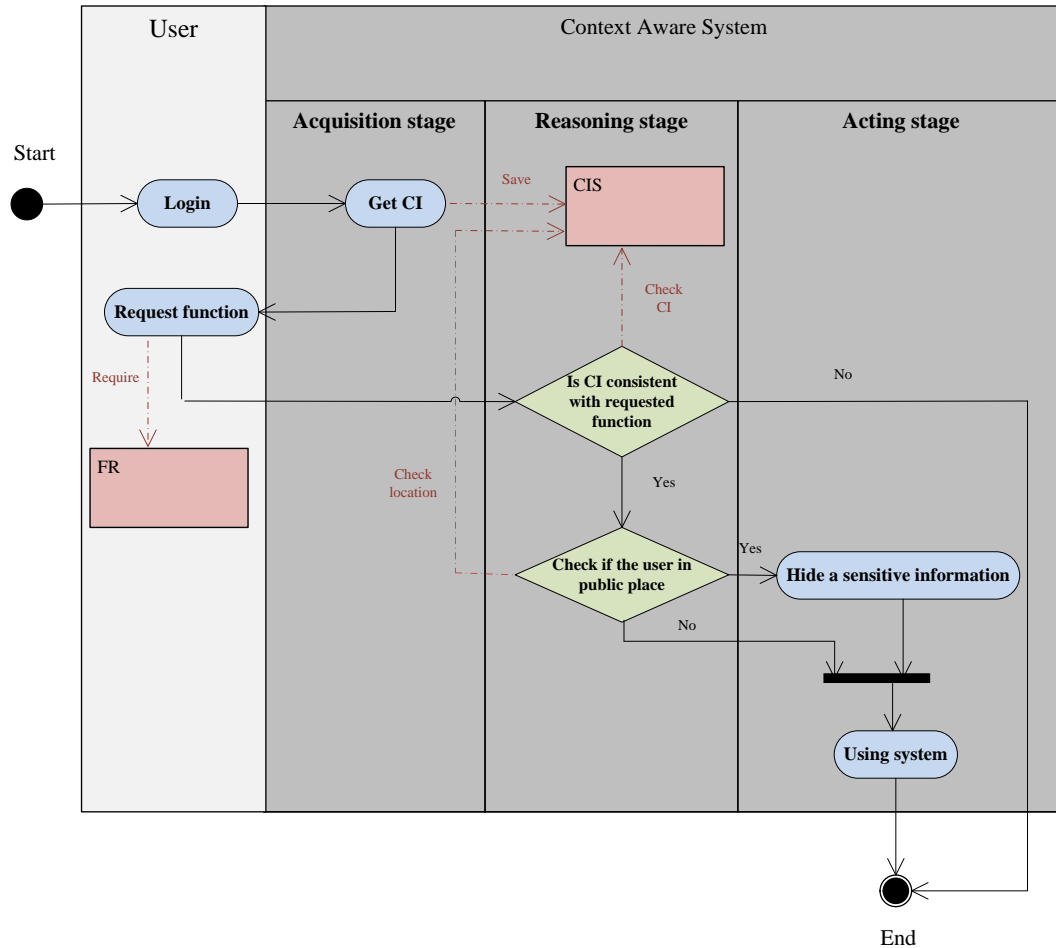


Figure 4.17: Confidentially with CASLC

As can be seen in Figure 4.18, the confidentiality states are distributed throughout CAS stages, based on each state's nature and behaviour. However, these states need to be joined in the whole CAS framework; this will greatly assist in showing how/when the data can be concealed during any CAS execution, as presented in Figure 4.18.

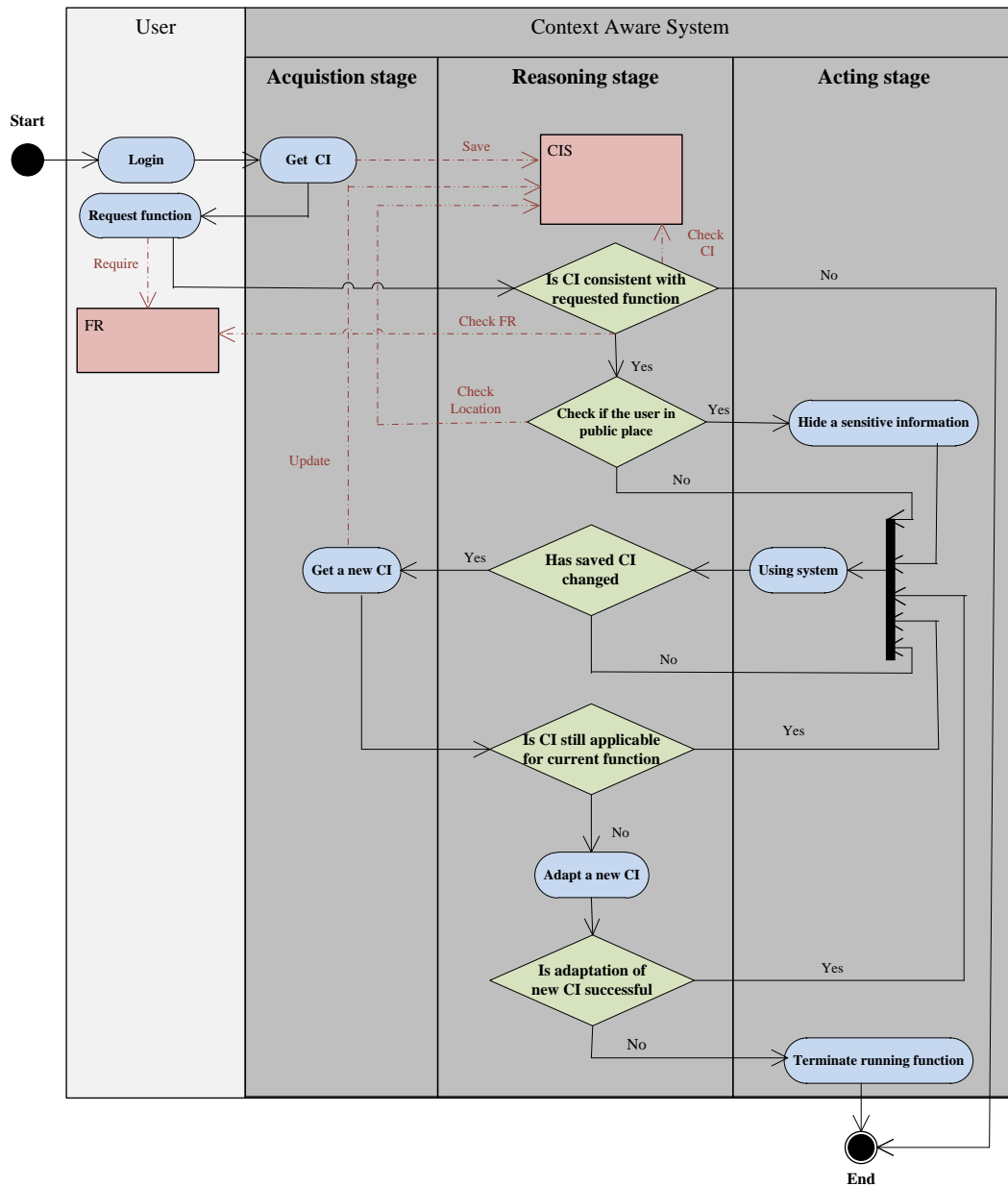


Figure 4.18: Confidentiality within CAS Activity diagram

also with confidentiality, the Activity diagram can be enriched to express the complete details of the CAS that orchestrates this important requirement. Our CO

CHAPTER 4. ENHANCING THE ACTIVITY DIAGRAM TO SUPPORT THE EXTENDED USE CASE DIAGRAM FOR CASS AND GATHER THEIR SECURITY REQUIREMENTS

icon be suitably drawn, as Figure 4.19 shows.

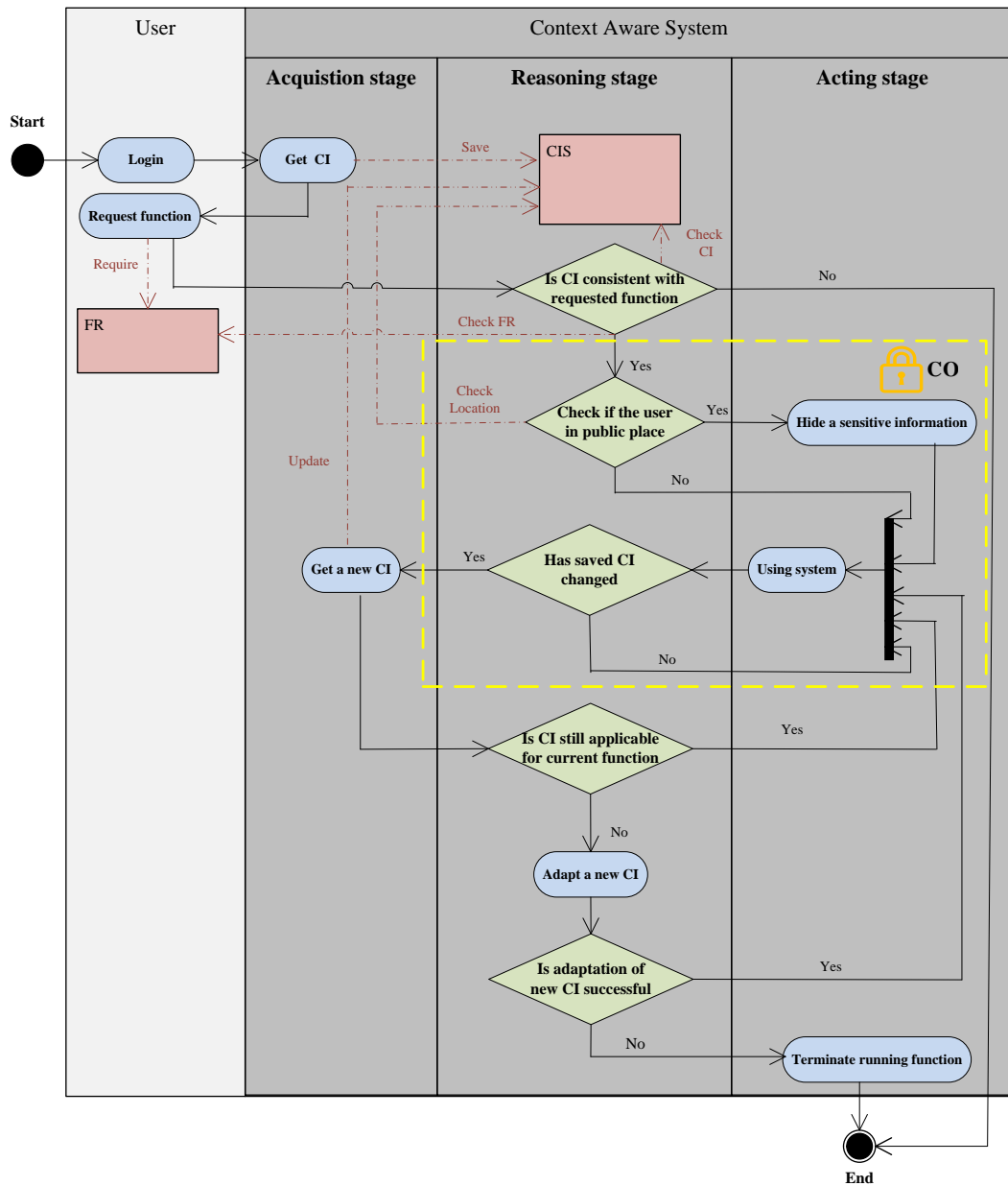


Figure 4.19: Determining the confidentiality process during CAS

4.3.2.4 Integrity

We already observed that Integrity cannot be captured at the level of a Use Case diagram. However, it can be described now using an Activity diagram. This type of security requirement is concerned with ensuring that the data are as described and in original condition, so the requirement is designed to prevent any modifications by unauthorised persons and to protect all data from corruption. We assume that:

the user has already satisfied the required CI and accordingly the service has already been granted. More specifically, once a service is invoked, the CAS immediately and in parallel creates a record that has a unique number, and that contains all relevant connection details, namely the user CI and the function invoked.

To record the CI, we propose the use of a table that be updated continuously. This table has a unique number per each user and that contains all relevant connection details, namely the user CI and functions invoked. The table should contain at least three lines, but extended to record additional information driving from future needs. The first line contains the name of the entity, which could be a virtual identity such as a number, a name, etc. The second line contains entity CI such as user location, time, user name, etc. Finally, the third line shows the functions available to entity, for instance, download, upload, etc. The table is continuously updated and monitored in order to ensure that the entity details are up-to-date and hence reliable while the system function.

This table containing user records can be profitably used to establish Integrity of the crucial data, namely user CI and function invoked; it can be demonstrated by means of an Activity diagram that describes its use as it can be seen in Figure 4.20.

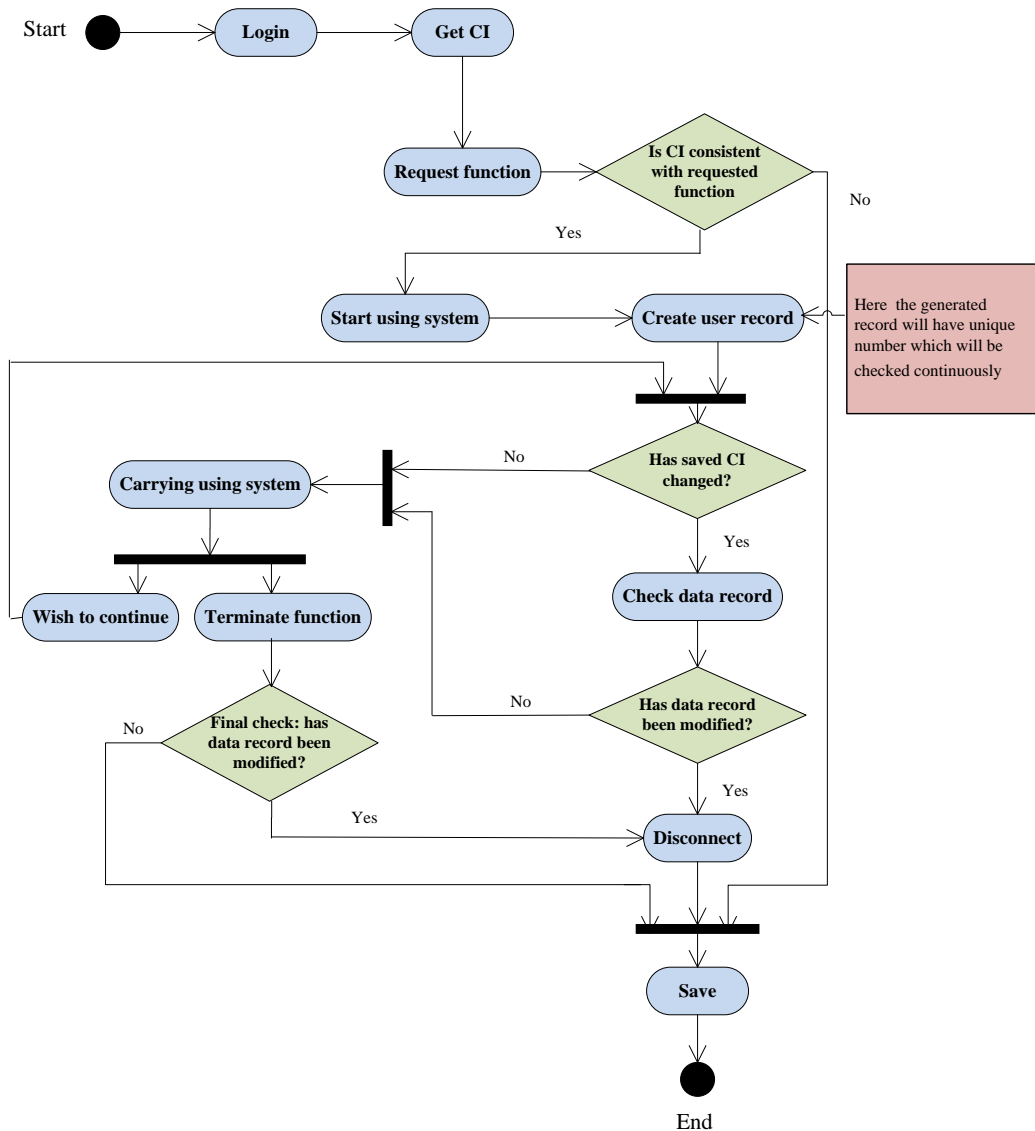


Figure 4.20: Basic concept of Integrity mechanism

Its Activity diagram shows that a process that commences once a user gains access to a service, the CAS gathers the user's current CI and saves it in a user record, so that when s/he invokes any action within that service or moves to another one, the CI is changed accordingly; the system in this case will then access and update

the user record.

Thus, the record is checked several times; the important ones are twice; the first time is whilst the service is running (and the CI is changed), and the second is once the service has been terminated by the user. The system will check before saving any data whether the record has been changed. This is shown in Figure 4.21, where the full CAS, expressing also integrity of crucial user data, is presented. In particular, it can be seen how our IN icon pinpoints the integrity requirement being captured.

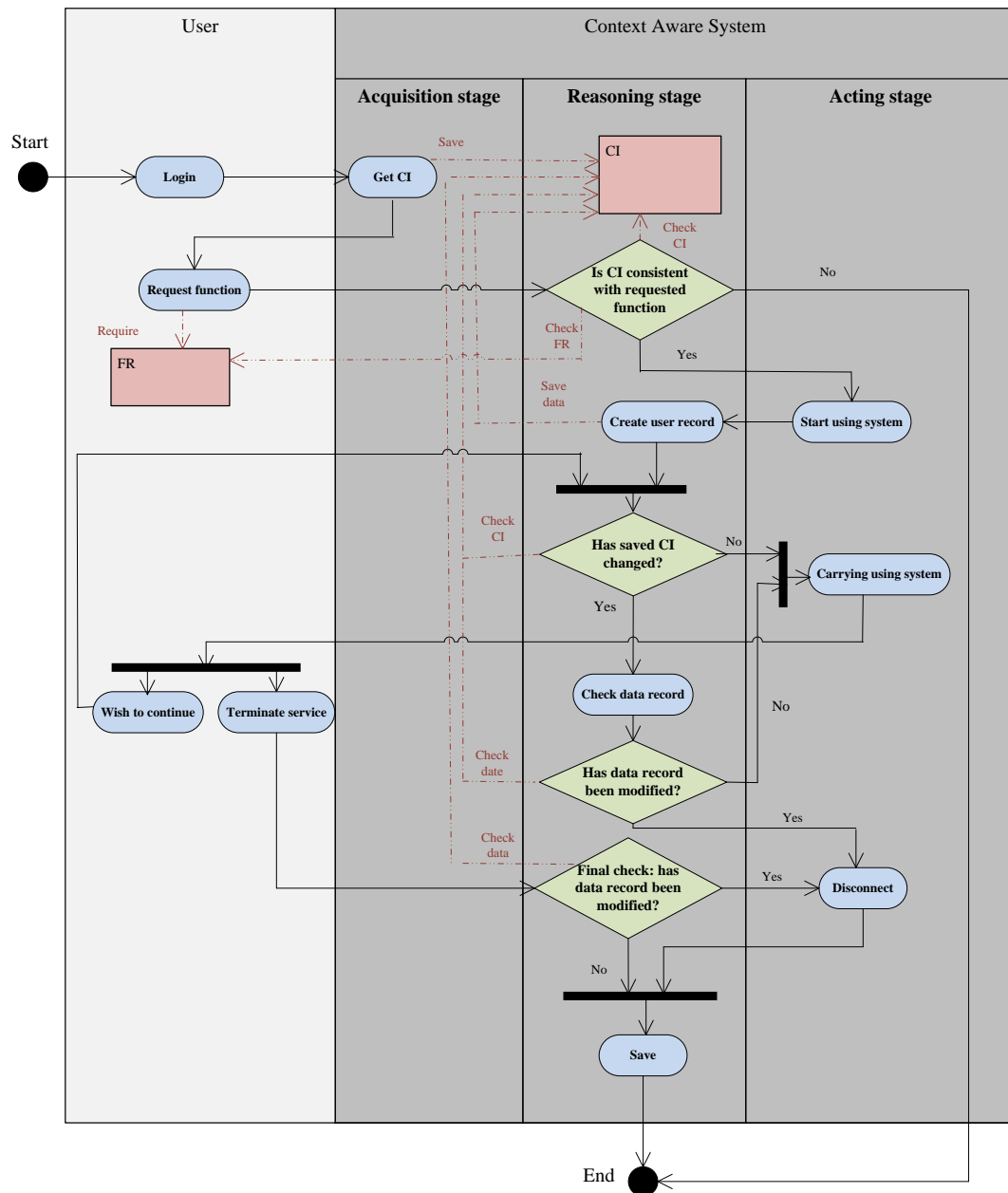


Figure 4.21: Integrity within CAS Activity diagram

The Activity diagram 4.21 shows that, the general processes for checking data integrity is lengthy; it starts by invoking a system function through the *<< Start - using system >>* action state, and then a user record is automatically generated

through the action state << *Create data record* >>. the data record will be saved in CIS. Following this, that record is checked by the condition << *Has saved CI been changed* >>; if 'yes', then the system will check the data record through << *Check data record* >>, and if not, the user will continue to use the service.

After verifying the data record, the system checks whether the record has been modified; if 'yes', then the user will be disconnected through the << *Disconnect* >> action state, and it will then save the data, but if not, the service will then be continued, and so on. Finally, once the user decides to end the current function (through << *Terminate service* >>, the system will run a final check to ensure that the recorded data have not been corrupted; this is done through the condition << *Final checking has data record been modified* >>. Finally, all the user's details are saved. The rest of the diagram shows that the data integrity is traced, and thus any unauthorised access can be prevented.

Also with Integrity, the Activity diagram can be enriched to express the complete details of the CAS that orchestrates this important requirement. Our IN icon be suitably drawn, as Figure 4.22 shows.

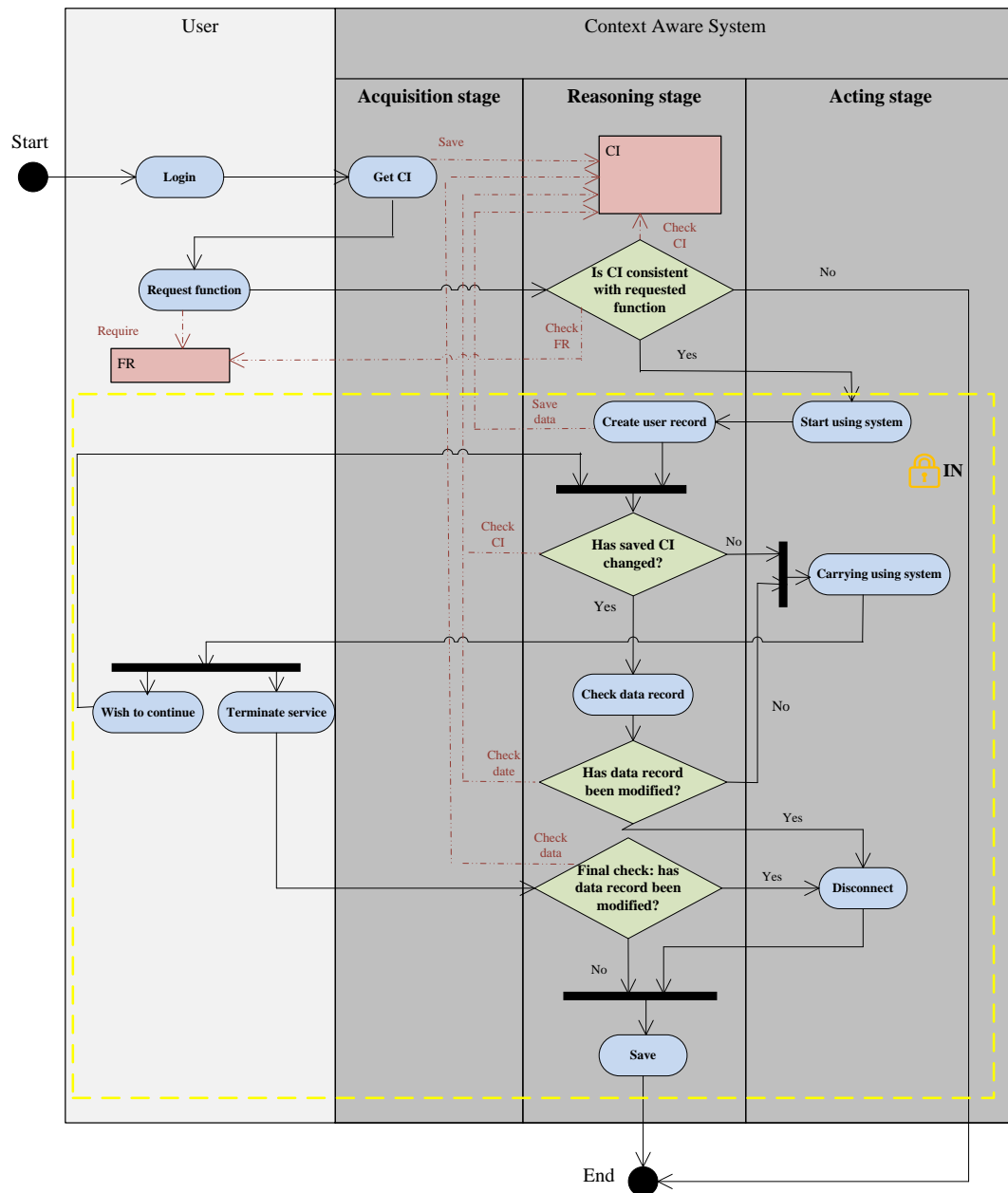


Figure 4.22: Determining the Integrity process during CAS

4.4 Chapter Summary

This chapter extends the Activity diagram technique coherently with the extension advanced before for the Use Case diagram technique.

More precisely, the Chapter started clarifying the limitations that hinder those notations in modeling CAS behaviour. The Chapter proposed a general framework that is able to model and show the dataflow within a CAS.

Then the Chapter demonstrated the proposed adjustments on the Use Case notations by extending also the Activity diagram notations. This Chapter presented an example in terms of its Use Case diagram first, and of its Activity diagram later. These examples support the evaluation of our innovative contribution to modeling CASs using UML.

Finally the Chapter demonstrated how to model all the key security requirements (Authentication, Authorisation, Confidentiality and Integrity) by means of an Activity diagram, In particular, it showed how CI can play a major role in securing any CAS, in the authentication stage (which can use static and dynamic parameters), in the authorisation stage (which can be managed by certain CI to control user privileges), and in the confidentiality stage (which exploits user location in order to organise the intensive data display). At the end, an innovative technique for tracing data integrity within a CAS was advanced and also demonstrated using an Activity diagram. In conclusion all the key security requirements could be modelled at this level.

Chapter 5

Enhancing the State diagram to support the extended Use Case diagram for CASs and gather their security requirements

Objectives:

- To describe the State diagram elements
 - To extend them in order to model CASs
 - To extend them in order to capture all the key security requirements
 - To verify whether the required extensions are in line with those previously advanced on other diagrams
-

5.1 Introduction

No previous study has been dedicated to modeling CASs and capture CI using State diagram techniques. Therefore, an original framework that uses a State diagram to describe object movement in CASs is proposed here.

The main purpose of exploiting State diagram is at least twofold. First, State diagram is good at describing the behaviour of an object across several Use Cases [92]; second, as CAS is dynamic in nature, then using State diagram is very fruitful to present the object mobility during a CAS functioning [72].

The following treatment shows at the level of State diagram how CI influence a CAS, thus reconfirming what the previous Chapters found out at the different level.

5.2 Existing State diagram elements

Although a State diagram consists of several elements, it is useful to recall here the main ones that will assist us in modeling a CAS [115].

- Initial state: an initial pseudo state represents a default vertex that is the source for a single transition to the default state of a composite state. There can be at most one initial vertex in a region. The outgoing transition from the initial vertex may have a behaviour, but not a trigger or guard.
- State: state model is a situation during which some (usually implicit) invariant condition holds. The invariant may represent a static situation such as an object waiting for some external event to occur. However, it can also model dynamic conditions such as the process of performing some behaviour (i.e.), the model element under consideration enters the state when the behaviour commences and leaves it as soon as the behaviour is completed).

- Transition: a transition is a directed relationship between a source state and a target state. It may be part of a compound transition, which takes the state from one state configuration to another, representing the complete response of the state diagram to an occurrence of an event of a particular type.
- Transition (join/fork): forks and joins have the same notation: either a horizontal or vertical bar (the orientation is dependent on whether the control flow is running left to right or top to bottom). They indicate the start and end of concurrent threads of control.
- Self-transition: a self-transition is a transition whose source and target states are the same.
- Note and constraint: a note (comment) gives the opportunity to attach various remarks to elements. A comment carries no semantic force, but may be useful to a modeler.
- End : the final state.

5.3 Enhancements

CAS behaviour can change rapidly based on a context changes [68], and a State diagram notations are fully capable of presenting object changes based on event or time; therefore, using state diagram to model CASs clearly becomes a line of research and development. We start off by developing a skeleton for modeling CASs using State diagram. This skeleton will be utilised as foundation for all the rest of this Chapter work.

Generally, a CAS is considered to be part of Mobile computing, and therefore it is convenient to begin our modeling by defining the main states for any mobile device behaviour [77], [83], [35]. We identify two essential states for describing

mobile device behaviour: the first state is $\ll Idle \gg$ (sometimes referred to as passive), in which nothing can be done until the device senses and responds to the environment; when this happens, the devices switches to the second state, which is $\ll Active \gg$, as Figure 5.1 shows:

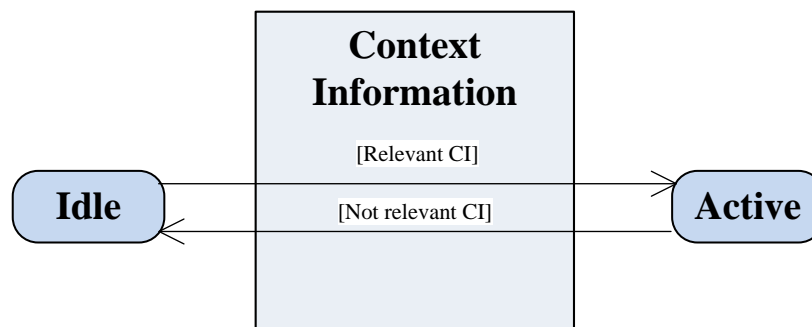


Figure 5.1: General states for mobile device

It can be seen that, the CI plays a major role in controlling the states behaviour. This general model will be employed to illustrate the blueprint of a CAS and its states; for example, the portable device mode in a CAS is initially in the $\ll Idle \gg$ state, which means that it has satisfied certain specific parameters that have led to an Inactive state.

In contrast, once these parameters are changed and the surrounding environment is sensed, the mobile device mode changes accordingly into the $\ll Active \gg$ state, which means the system moves into an operational mode. Hence, the $\ll Active \gg$ state is considered more imperative in modeling CAS states as it describes when/how the system becomes aware, therefore, it is necessary to decompose the $\ll Active \gg$ state and express it in depth as sequential substates; this will assist in expressing object mobility in CASs by defining the relevant states using the

observation made above as a basis, and specifically taking into account the CI either in sensing or processing states, we have produced a coherent and reliable skeleton as Figure 5.2 depicted to detail in details the $\ll Active \gg$ state taking into account the sequential process of the object within a CAS (from the Sensing and Gathering CI stage, then passing through the Processing stage, until reaching the Acting stage in order to deliver the service).

This proposed diagram will be exploited as a foundation model and used in all the subsequent sections to express the CAS states and later their security requirements.

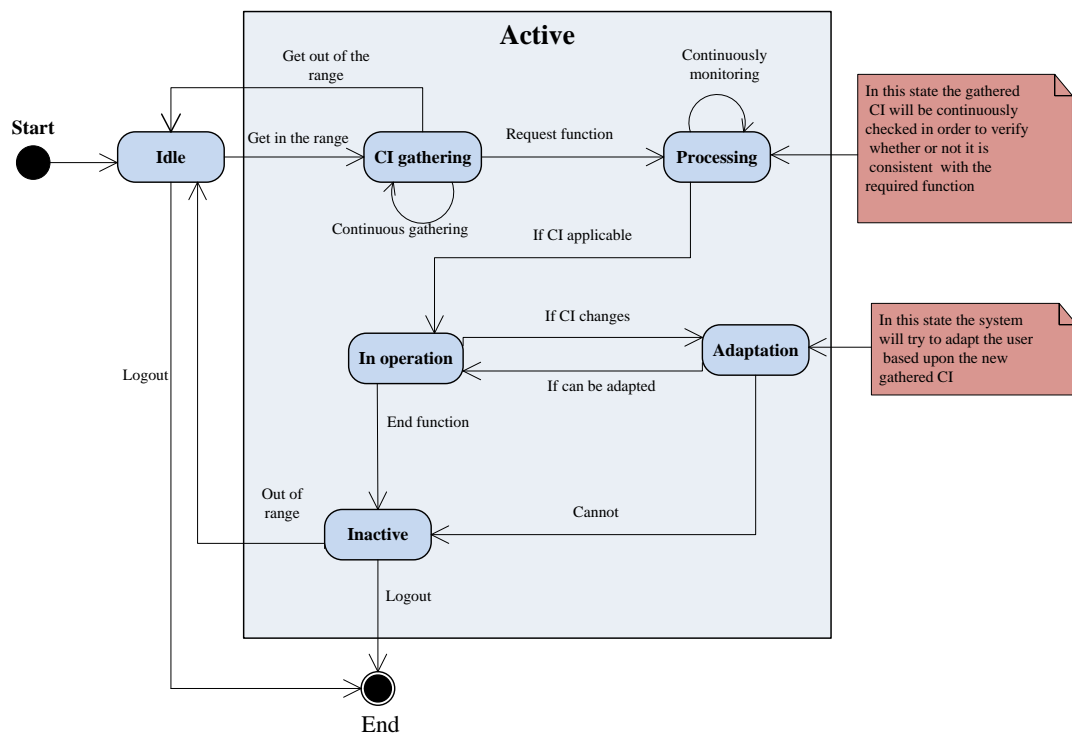


Figure 5.2: Decomposed Active state

Figure 5.2 indicates that, the $\ll Active \gg$ state is initiated when the CAS senses the surrounding environment, gathers the required CI, and then processes it once the user has requested any available function in order to ascertain whether or

CHAPTER 5. ENHANCING THE STATE DIAGRAM TO SUPPORT THE EXTENDED USE CASE DIAGRAM FOR CASS AND GATHER THEIR SECURITY REQUIREMENTS

not the CI is appropriate for accessing the requested service. Moreover, the diagram shows when the function in the CAS can be adapted once the CI has changed. The Table 5.1 lists all the CAS states and events:

State name	Events in state	Description
Idle	Get in the range. Logout	The initial state.
CI gathering	Continuous gathering. Re-quest function. Get out of the range	The first state in Active mode, which senses and gathers CI.
Processing	Continuous monitoring . If CI is applicable	This starts once the user requests a function, which checks whether or not the gathered CI is consistent with the requested function.
In operation	If CI changes while using the system. End function	This state indicates that the system is in operation mode, and the service is being performed.
Adaptation	If can be adapted. Can-not	This state tries to adapt a new CI.
Inactive	Out of range. Log out	This describes the system once there is no running function.

Table 5.1: Description of Active state

The expressiveness of the diagramming techniques presented above will be demon-

strated below.

5.3.1 Extending the State diagram to support the extended Use Case diagram

Having developed a reliable CAS model using a State diagram, it is imperative to support the extension that was done previously on the Use Case diagram. Chapter 4 introduced appropriate notations for the CIS and Context links in an Activity diagram. The same notations can be used also in a State diagram. To do this, a State diagram must yet account the CIS and the Context links. For convenience, we recall them here:

- Context Information Store (CIS): this contains all the gathered CI during a function execution; this store is continuously checked and updated in order to ensure whether or not the user has CI that is applicable for invoking the required function.
- Context Links: these are to show how the CI can affect the CAS states an State diagram as well as to depict how any decisions are made. These links do not partake in object flow, and therefore they differ from the other links.

Accordingly, Figure 5.3 expands Figure 5.2.

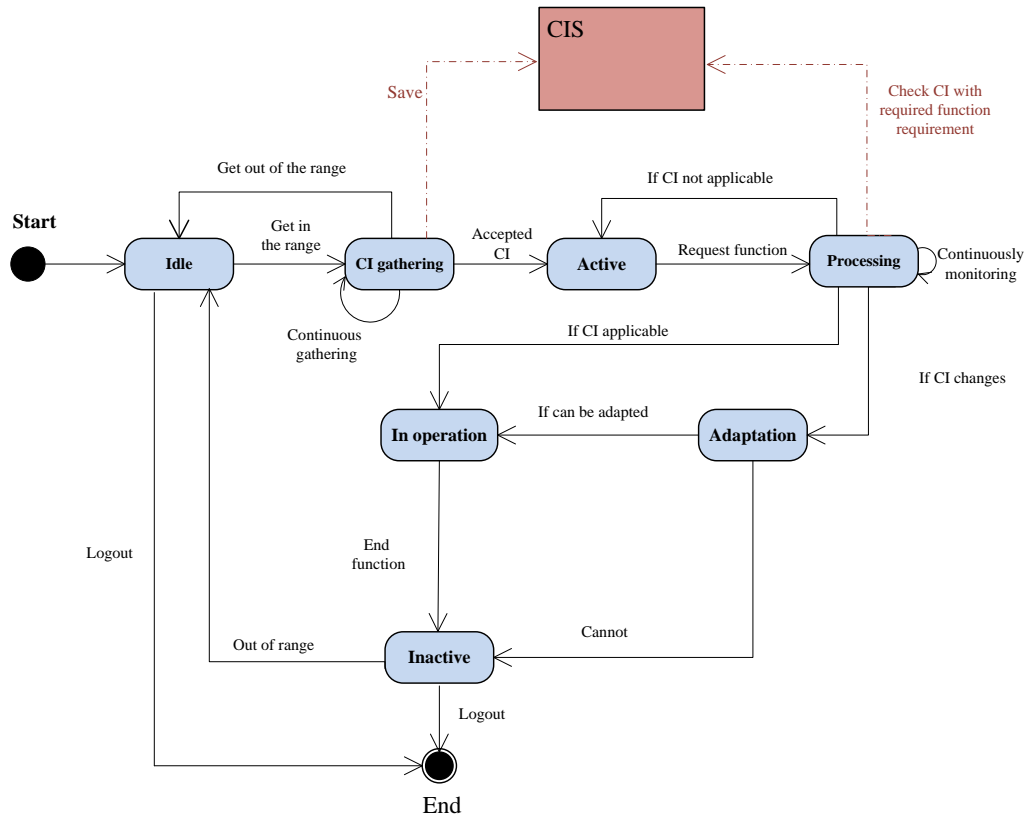


Figure 5.3: State diagram model with proposed enhancements

To complete our CAS model based on a State diagram, we must yet consider the CASLC stages of Acquisition, Reasoning and Acting. The reason behind this is to clarify in which CAS stages the presented states can be performed. This is done in Figure 5.4.

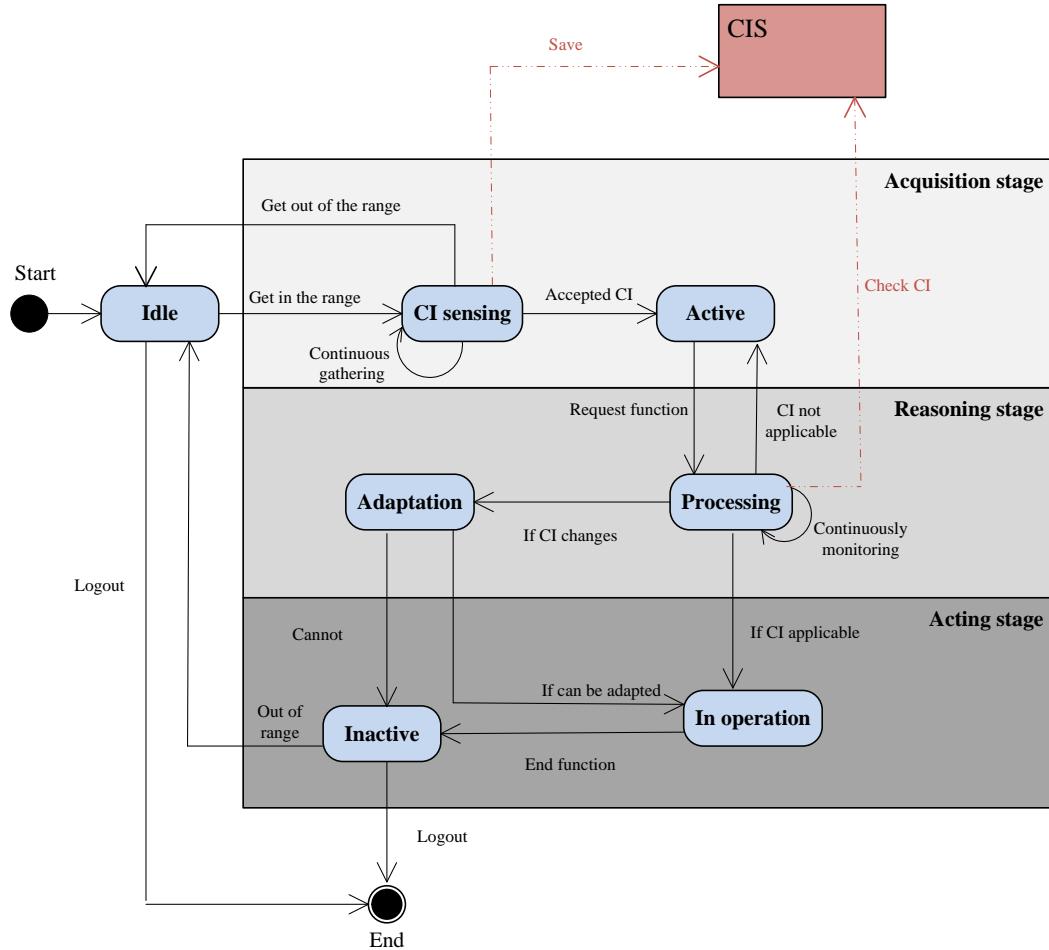


Figure 5.4: State diagram model with CASLC

It can be seen that each stage has certain states that are invoked only in that stage. As mentioned earlier, the adaption mechanism will not be covered, as we only focus on the impact of CI on the CAS states.

We are not going to express the function itself in detail, we are rather going to express how the object in CAS can be moved from state to another states.

In order to evaluate how the enriched State diagram can support the extended Use Case diagram which showing how CI can control the object in CAS. We will again recall the same example of library system Use Case 5.5 to apply the proposed State diagram model.

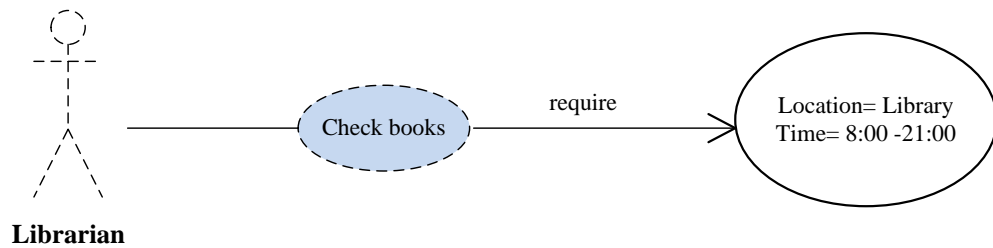


Figure 5.5: Check book Use Case

We can use our extended State diagram as Figure 5.6 shown to present the object of library Use Case scenario.

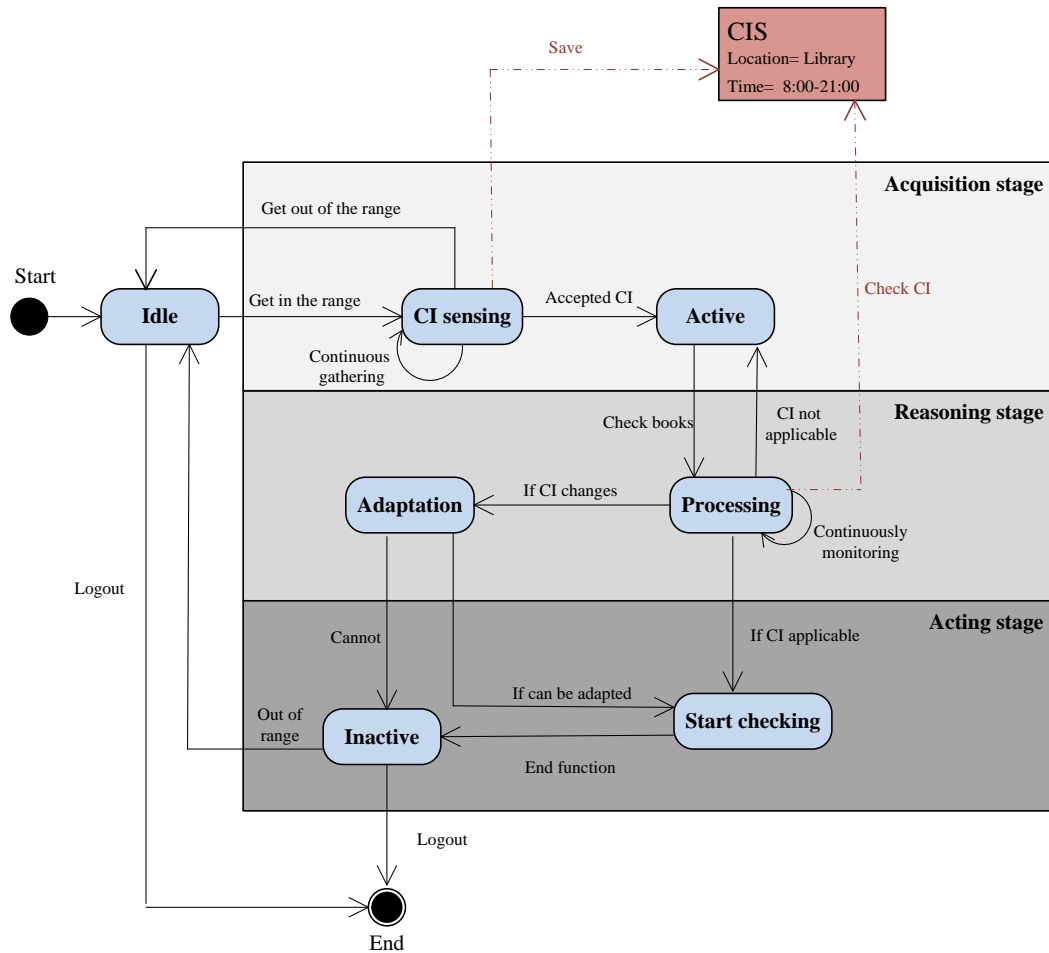


Figure 5.6: Check book State diagram

This Section has convinced that a State diagram Model can be effectively used to model a CAS and, in particular, to highlight the role that CI plays. The next step is to account for security requirements.

5.3.2 Gathering security requirements for a CASs using the extended State diagram

A vital reason for using a State diagram here is to describe the object mobility of CASs, and therefore to model their security requirements (using a State diagram). It is also necessary to make them dynamic, and this behaviour will be entirely manipulated by CI. Thus, each security type needs several related states in order to model and present the required actions. The proposed State diagram model will be used to show how/when the security type can be invoked till reach the << *Active* >> or << *In operation* >> state, based on the nature of security type.

This section defines diagramming techniques to capture all the key security requirements within a State diagram. In particular, we shall see that the complications arising the dynamic nature of CI are only modest, and the resulting diagrams remain very readable. For convenience, we begin by reviewing the key security requirement in terms of states:

- Authentication: it is captured by a set of states representing how to validate the user identity in a CAS.
- Authorisation: it is captured by a set of states representing how to manage the user authorisation.
- Confidentiality: it is captured by a set of states representing how to protect a any user-sensitive information.
- Integrity: it is captured by a set of states representing how to ensure that the transferred data in CASs is not corrupted or modified by unauthorized persons or any third party.

5.3.2.1 Authentication

Generally, the user details are utilised in order to verify whether or not that user is allowed to access the system. There are many context parameters that could be used to authenticate the user identity, such as username/password, location and device IP, and they govern the process that occurs between the *<< Idle >>* and *<< Active >>* states. This process is detailed by the State diagram in Figure 5.7

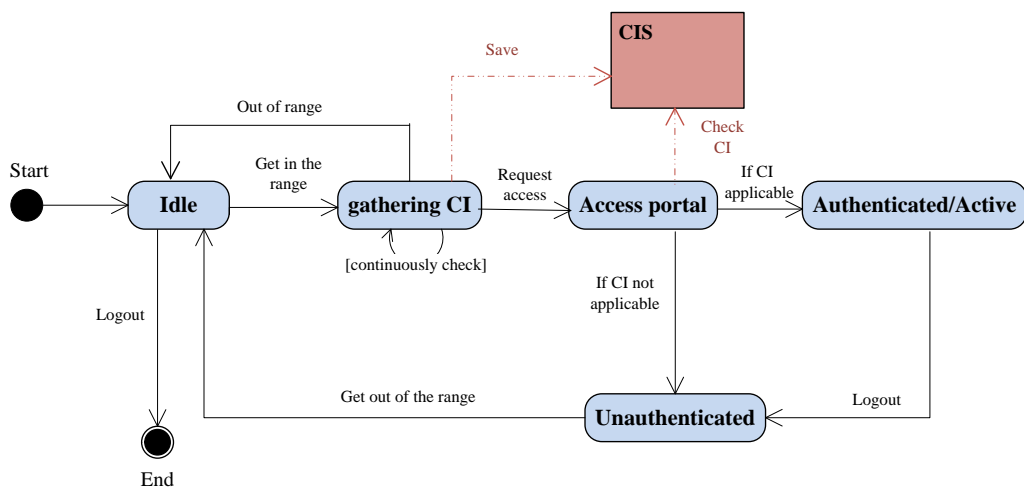


Figure 5.7: Basic concept of Authentication mechanism

Our diagram shows that, the first state to model a mobile device system is *<< Idle >>*; this state is maintained until the device is in range and can sense the surrounding environment. Then, the user device may be activated, following which, the CI is gathered. Some of this information is checked in order to verify the user identity in case the user makes a request to access the CASS through an access portal. Only two states become possible:

- Authenticated: if the required CI for verifying the user is applicable;
- Unauthenticated: otherwise.

The following table 5.2 summarises the states and events utilised in modeling the authentication requirement of a CAS.

State name	Events in state	Description
Idle	Get in the range. Logout	The initial state.
CI gathering	out of range. Continuously check. Request access	Gathering CI and making it ready to use once the user has requested a service.
Access portal	If CI applicable. If CI not applicable	Interface to enter details.
Authenticated	Logout	User status once the details are accepted.
Unauthenticated	Get out of range	User status once the details are not accepted.

Table 5.2: Description of Authentication states

Finally, once the authenticated user has logged out, the user's current state is switched to the << *Unauthenticated* >> state, and then to the << *Idle* >> state once s/he is detected as being out of range. These states need to be defined in terms of the CASLC in which they can be preformed. Our diagram is modified as Figure 5.8, which also shows the use of CIS and related context link.

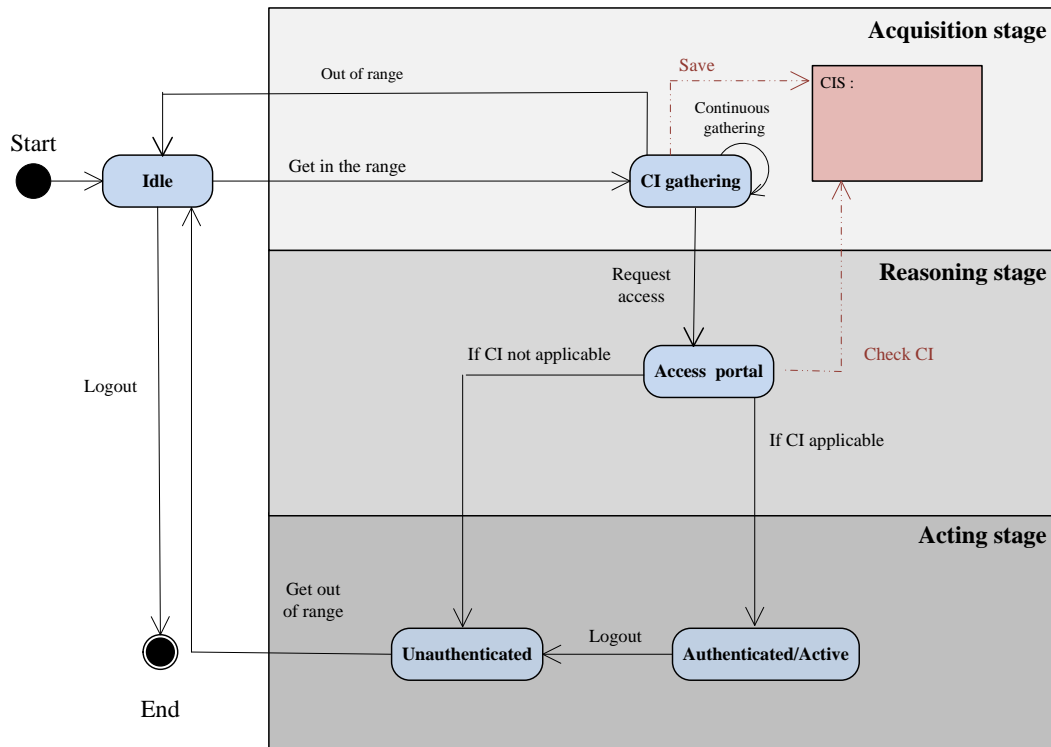


Figure 5.8: Authentication with CASLC

Our latest diagram is the most detailed representations of the authentication requirement for a CAS using an UML State diagram.

5.3.2.2 Authorisation

We shall see that also the authorisation requirement can be presented in a State diagram through a number of appropriate states. We assume that:

the user is already identified and authenticated before reaching this stage, and therefore we consider the user to be in the active mode.

Thus, the authorisation process in a State diagram commences with the << *Sensing anew CI* >> state; this state is devoted to double-checking whether or not the user still has valid CI following authentication and prior to authorisation.

Immediately after authentication stage, the user may not be allowed to in state to access any system functions, as it is likely that the CI will have changed. Therefore, once the CI has been approved, the user request will be processed and checked by the << *Authorisation policy checker* >> state, which may be considered the fundamental state within authorisation as it decides whether or not to authorise access to the required service by checking the authorisation parameters.

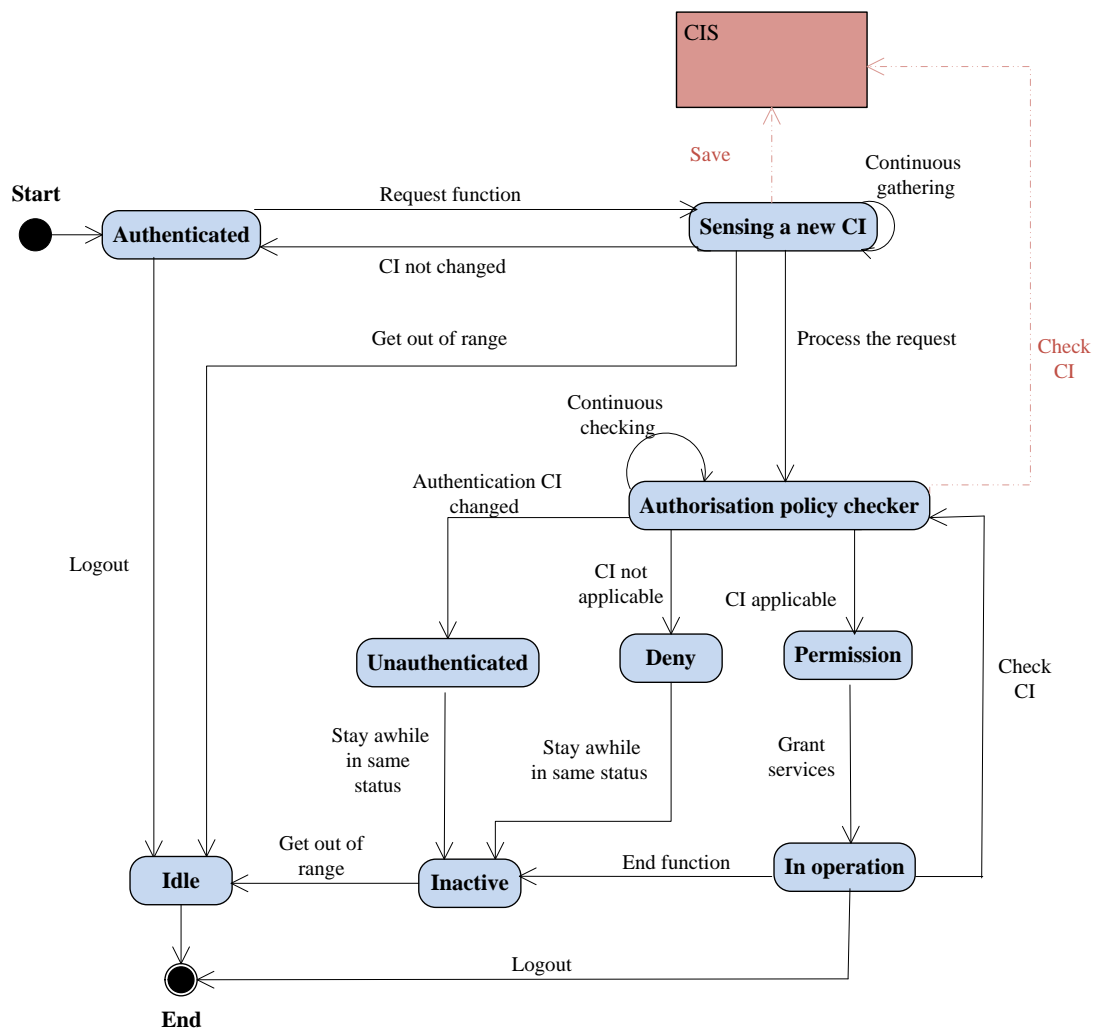


Figure 5.9: Basic concept of Authorisation in State diagram

The diagram 5.9 depicts the relevant states that involved for managing the authorisation function for a CASs. The following Table 5.3 summarises the states and utilised events.

State name	Events in state	Description
Authenticated	Request function. Logout	The user is able to access the system functions.
Sensing a new CI	Continuous gathering. CI not changed. Process the request	Realising the environment and gathering new sensor data.
Authorisation policy checker	Continuous checking. CI applicable. CI not applicable. authentication CI changed	Here, the system continuously checks authorisation conditions.
Deny	Stay a while in the same statue	The required service cannot be performed.
Permit	Grant services	The required service will be activated.
In operation	Check CI. End function. Logout	Once the service is being used.
Unauthenticated	Stay a while in the same statue	The system functions will not be displayed.
Inactive	Get out of range	This describes the system once there is no running function.
Idle	End	The end state.

Table 5.3: Description of Authorisation states

Accordingly, all the aforementioned states will be utilised in order to illustrate the authorisation function in a CASs as well as to show in which stages they can be run. The next step is to frame our diagram for authorisation within the CASLC stages. This is represented in Figure 5.10. It can be seen that the main logic of

authorisation is captured in Acting stage.

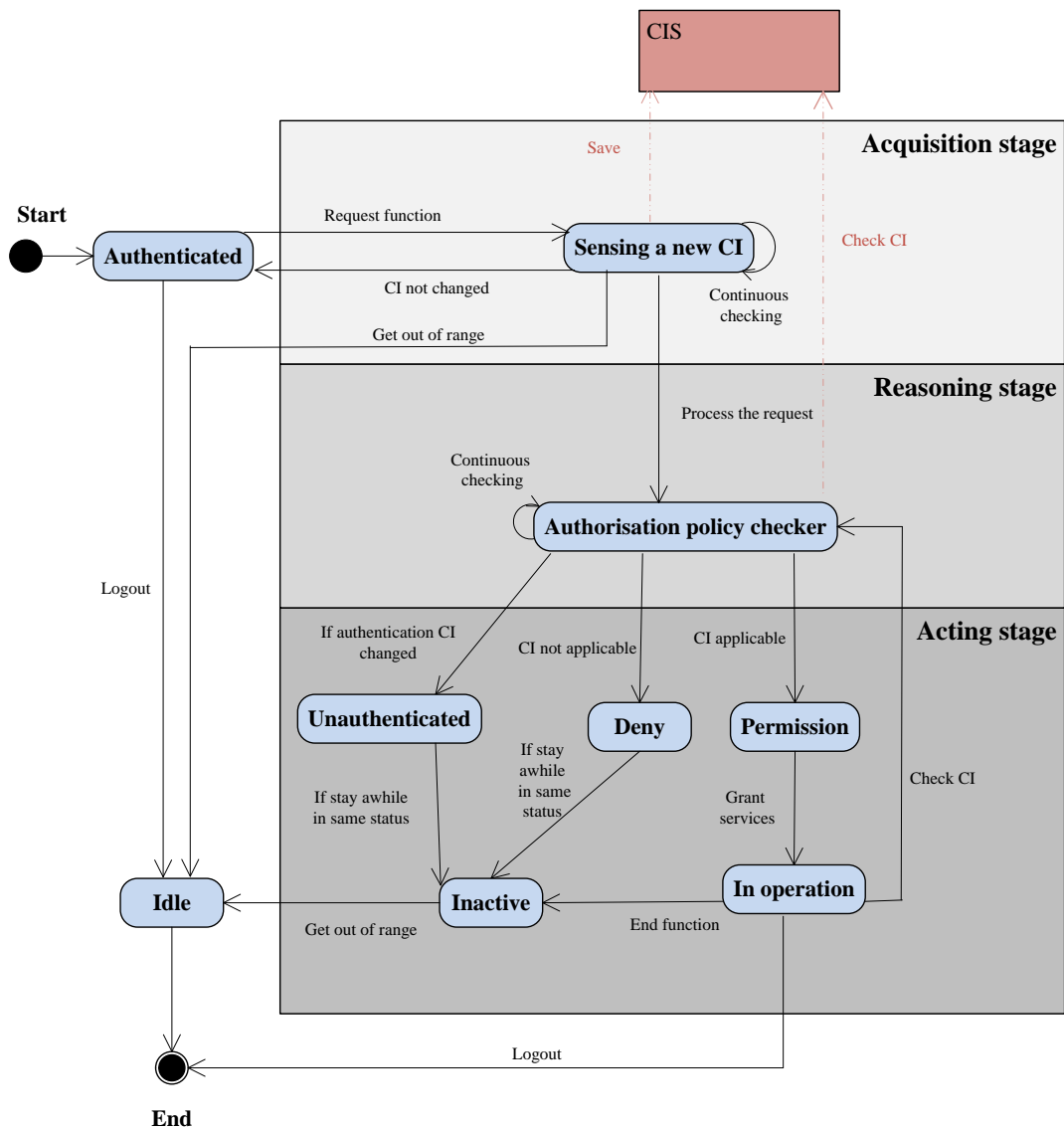


Figure 5.10: Authorisation with CASLC

5.3.2.3 Confidentiality

How to define confidentiality within a CAS was explained in previous Chapters. It involved determining specific constraints that hide or release a service. Thus the user's location will be exploited as a key parameter for managing the confidentiality requirement in a State diagram. The State diagram below 5.11 depicts the mechanism for confidentiality in a CAS; it also illustrates how the location parameter can be used to control CAS functions.

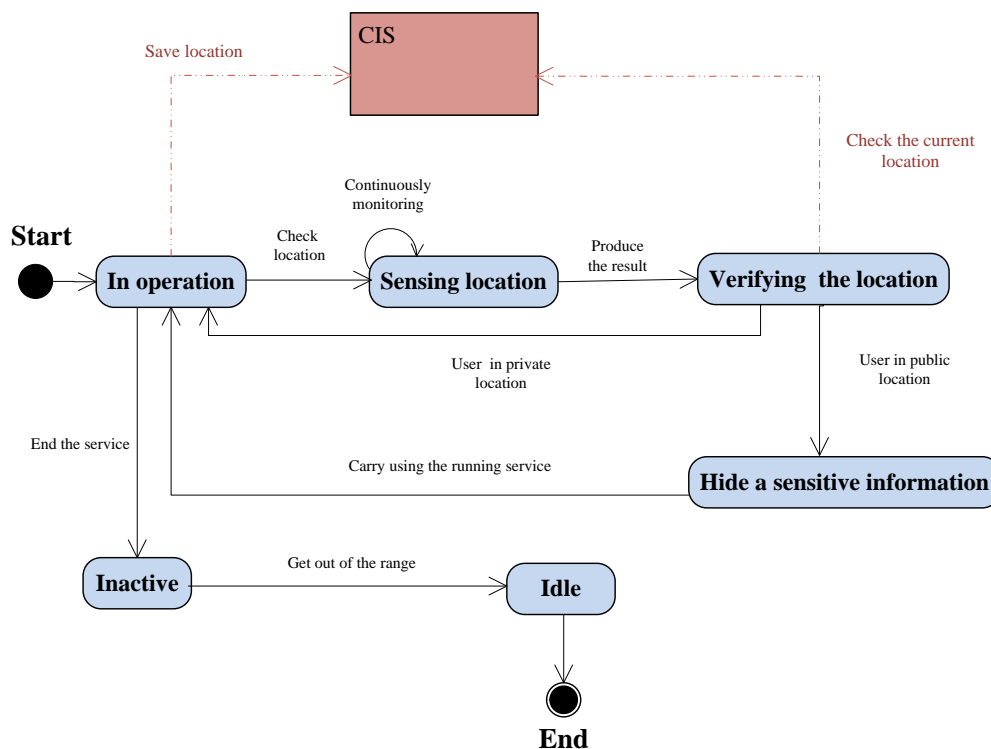


Figure 5.11: Basic concept of Confidentiality in State diagram

The reason for starting the presentation of confidentiality by the $\ll In\ operation \gg$ state is that we assume that:

the user has already been authorised to access the required service; the confidentiality function should be invoked the moment a service is requested, and that service will not be granted until the user has satisfied all the authentication and authorisation requirements.

Therefore, as Figure 5.11 stated, the main CAS states that are invoked to ensure confidentiality are firstly the << *In operation* >> state, which means the service has been already granted and invoked, and then the CAS state is changed once the user requests a function to << *Sensing location* >>; then the result of sensing the location will be passed to the next state << *Verifying the location* >>, in this state, the CAS detects the environment and decides whether it is public or private. Thus, the service will be granted in accordance with the system decision; if it detects a public place, then some sensitive information will be hidden, and if not, the whole service will be released.

State name	Events in state	Description
In operation	Check location. End service. Save location	The user is already in active mode, following the authentication step.
Sensing location	Continuous monitoring. Produce the result	Sensing user location only and pass the result to next state.
Verifying the location	Check the current location. User in private location. User in public location	Deciding whether the user is in a private or public location.
Hide a sensitive information	Carry using the running service	Hiding the sensitive information while the user is located in public place.
Inactive	Get out of the range	Once a user terminates the service.
Idle	End	Final state.

Table 5.4: Description of confidentiality states

Having identified all the CAS states that assist in modeling confidentiality within a CAS, it is important to frame them within the CASLC stages; this facilitates presenting the whole blueprint for a CAS as well as revealing what functions are carried out within what stages. For example, it can be seen that the decision on whether it is safe or not to release the sensitive information or service is taken during the Reasoning stage.

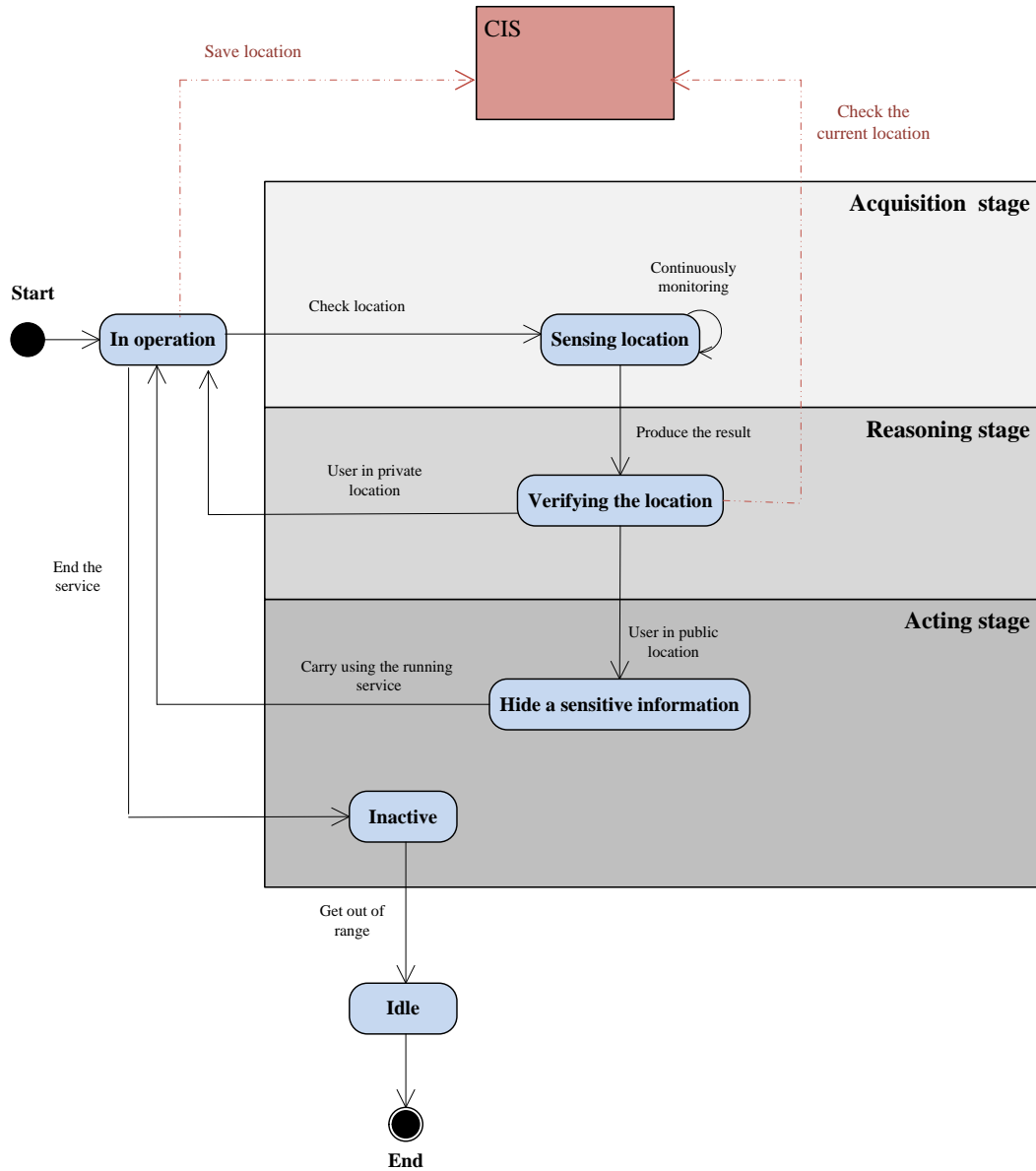


Figure 5.12: Confidentiality with CASLC

From all diagrams that describe the mechanism for protecting the sensitive user information, and they also demonstrate where/how user confidentiality can be achieved in a CAS through a State diagram.

5.3.2.4 Integrity

This type of security requirement concerns protecting data integrity, and seeks to ensure that it is not modified through any unauthorised source. Therefore, we would also utilise here the data record approach as explained in Activity diagram (Chapter 4). State diagrams are outstanding for describing data integrity in a CAS, as diagram 5.13 clearly shows; they can demonstrate how to model data integrity, which technically comes after authentication and authorization steps, and, once the user has already invoked the function to either send or retrieve data.

In order to track any user request and to protect data from unwanted alteration, a number of states are necessary.

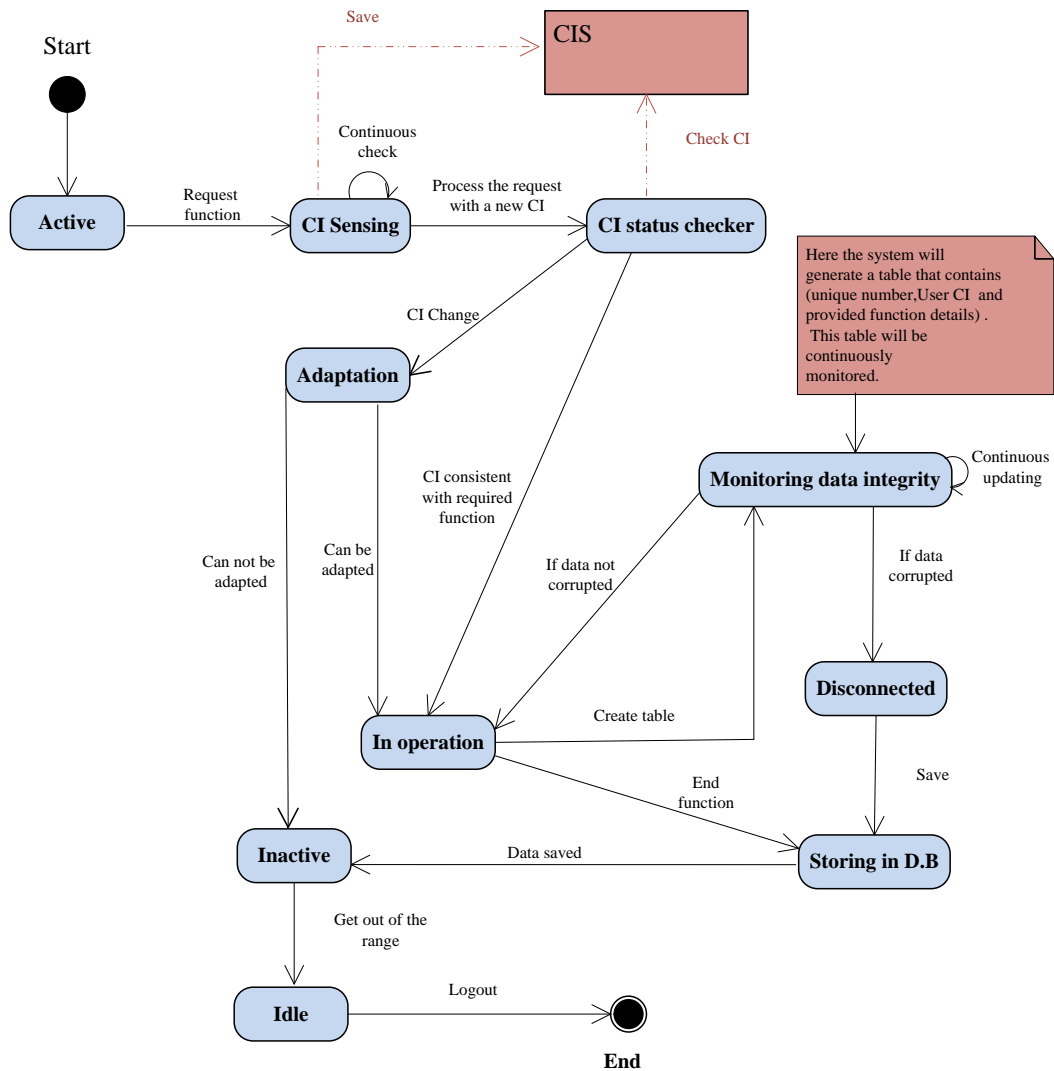


Figure 5.13: Basic concept of Integrity in State diagram

The following Table 5.5 summarises the states and events utilised in modeling the Integrity function for a CAS.

CHAPTER 5. ENHANCING THE STATE DIAGRAM TO SUPPORT THE EXTENDED USE CASE DIAGRAM FOR CASS AND GATHER THEIR SECURITY REQUIREMENTS

State name	Events in state	Description
Active	Request function	This mode is initiated once there is interaction with the environment.
CI sensing	Continuous check. Process the request with a new CI. Save	Gather a new CI and pass it to the CI checker.
CI status checker	Check CI. CI consistent with a required function. CI changed	As CI is rapidly changeable, it is imperative to keep checking to decide whether or not the CI is applicable for the requested function.
Adaptation	Can be adapted. Cannot be adapted	The state that can depict whether or not the object can be adapted.
In operation	Create table. End function	Once the service is being used, the table will be created.
Monitoring data integrity	Continues updating. If data corrupted. If data not corrupted	To keep checking whether the data have been altered.
Disconnected	Save	This state is initiated on discovering that the data have been modified.
Storing in D.B	Data saved	Saving the data.
Inactive	Get out of the range	Once the service is terminated.
Idle	Logout	The mode becomes an Idle.

Table 5.5: Description of Integrity states

Through the above, we can guarantee user confidentiality by protecting the data

from any alteration or unauthorized modification. These states can also be applied to show how the data can be kept safe through the CAS stages, as shown in Figure 5.14.

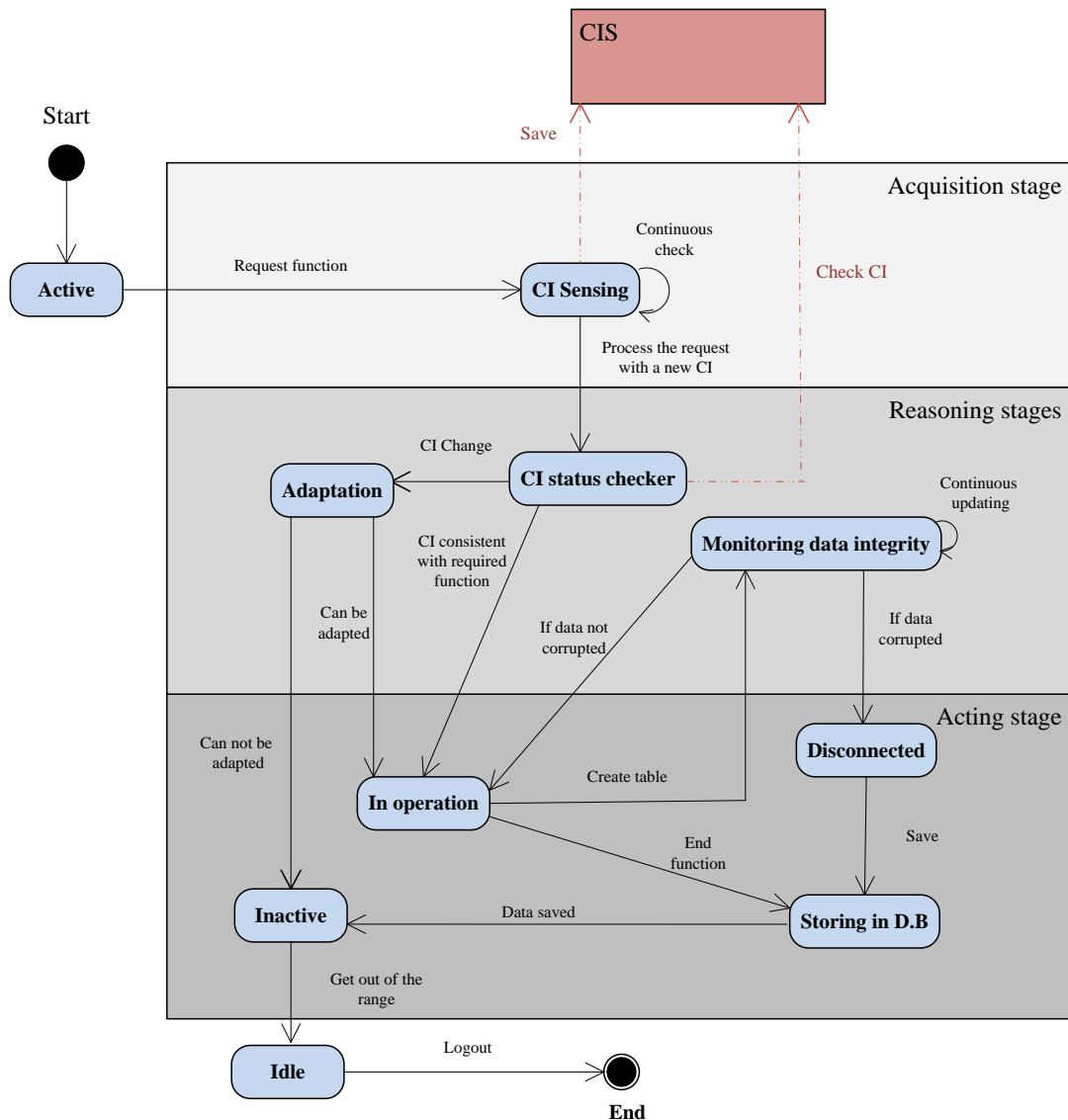


Figure 5.14: Integrity with CASLC

For example, during the Acquisition stage, the user CI is gathered only through the *<< Sensing CI >>* state, whereas many states are involved in the Reasoning stage, such as *<< Active >>*, *<< Monitoring data integrity >>* and *<<*

CI status checker >>. Finally, in the Acting stage, the service is used through the << *In operation* >> state, and is terminated through the << *Disconnected* >> state.

5.4 Chapter Summary

This Chapter has provided the general diagram technique for presenting a CAS and then expressing its security requirements using a UML State diagram. This framework is mainly utilised to demonstrate the extension that was done on the Use Case diagram by showing the high impact of changing CI on CAS states. These findings indicate that the same enhancements seen in the previous chapter for Activity diagram scale up to the State diagram level.

The Chapter started by describing in depth the State diagram notations. It also underlined the limitations that hinder the notations in modeling CAS behaviour. Accordingly, the Chapter then advanced a general diagramming technique to account for the changing context in a CAS.

The rest of the Chapter described our techniques for gathering the key security requirements by leveraging upon the proposed State diagram enhancement. It was also showed how the CI can play a major role in securing any CAS, either in the authentication stage, which can use static and certain dynamic parameters, or in the authorisation stage, which can be managed by certain CI to control the user privileges, or in the confidentiality stage, which exploits user location in order to organise the intensive data display. Then, our State diagram presented the way of tracing data in a CAS in order to check their integrity.

Finally, it can be observed that the diagramming techniques developed in this Chapter rest on extensions (to the State diagram) that are in line with those made in previous chapters on Use Case and Activity diagrams respectively.

Chapter 6

A Real-World Case Study: An Infostation-based M-Learning System

Objectives:

- To introduce an M-learning system(definition - structure - work explanation)
 - To define the M- Learning users and their functions
 - To define the security requirements for M-learning
 - To model the M-learning system functions using the UML diagrams
 - To model the security requirements for an M-learning system using UML diagrams
-

6.1 Introduction

The main aim of this work is to extend existing UML diagrams elements; specifically, the Use Case diagram is extended to model a CAS, and then to capture its security requirements, and the proposed framework for both Activity and State diagrams will demonstrate that extension. Accordingly, this Chapter presents a case study in order to apply the proposed modifications on a real-world system; this is conducted in the M-learning environment. M-learning is a classic example of a system that needs to be context aware if it is to function properly. It is necessary for the M-learning system to be aware of the user's changing requirements for two main reasons: the mobility of the users and the diversity of the devices (such as smartphones, personal digital assistants (PDAs), etc.).

Education methods have passed through many evolutionary stages to reach the M-learning stage. This started with the traditional approach of face-to-face learning, and then passing on to e-learning with the advances in technology. M-learning methods are becoming common in the educational environment and are thought of as representing the future of learning, in which mobile systems and bricks-and-mortar universities will complement each other.

6.2 M-learning definition

The M-learning technology is one step ahead of e-learning, as users are free to access lessons and download material onto their mobile devices with unlimited choice in terms of location and time. This new technology has dramatically changed the concept of delivering lessons, as the user is not required to physically attend lessons in a classroom. The main concept behind M-learning represents a shift of environment, mainly from wired to wireless technologies in order to make the learning experience

more flexible and accessible for users.

6.3 M-learning Infrastructure

To better understand the concept of M-learning, we consider here a real-world case study, which is about an infostation-based M-learning system [42]. The infostation system paradigm was initially posited by Frenkiel et al [119], and entails short-range communication between wireless nodes. This service enables the user to maintain an uninterrupted service when moving between different infostations. The major idea behind this system is to allow the user to access M-learning services, such as m-lecture, m-tutorial and m-test, and also to use a range of other communication services, such as private chat, intelligent message notification and phone calls; all of these are performed through portable devices such as phones, laptops and PDAs. The M-learning system is principally comprised of three aspects, as described below:

- Central Infostation (CenStat): this is the system server that controls all transactions and functions. This can also be used to save all the data and to send requested files to the user via the context information provider (CIP).
- Context Information Provider (CIP): these are generally considered as nodes that are deployed regularly on the university boundary in order to cover all university faculties. Each CIP can be accessed by different mobile nodes using various technologies (such as Wi-Fi, Local Area Network (LAN) or Bluetooth).
- Mobile devices: these can be used to utilise the available services, and include mobile phones, laptops and PDAs; these mobile devices are used by multiple users such as students, lecturers and administrators.

There are three levels in the architecture of the infostation-based M-learning system, as depicted in Figure 6.1:

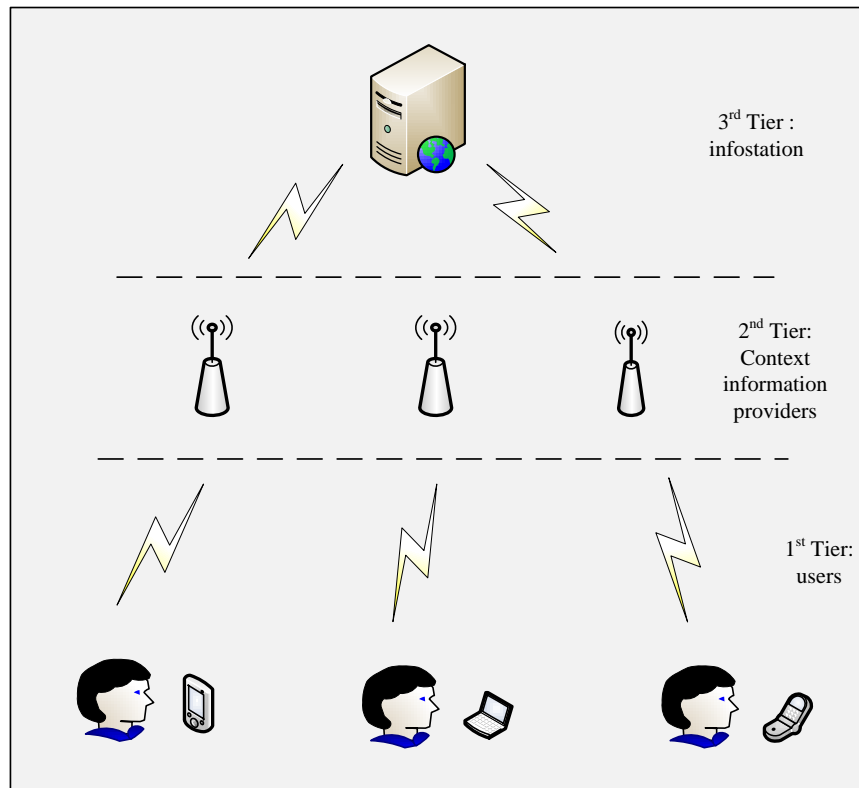


Figure 6.1: The three-tier architecture of the infostation-based network

6.4 Description of M-learning system

The M-learning system provides several functions/services to users; however, each service/function has special and required CI and constraints. These constraints are already predefined and stored in the CenStat in order to match them with any user-required function. User CI is essentially gathered through many sensors that are deployed within the university boundary and then automatically saved in the CenStat database; for example, when a user attempts to access an M-learning service, the current user CI is gathered by the CIP and relayed to the CenStat. The CenStat then matches the CI with the given constraints to allow or deny the user

access to that particular service. To better define and track the user, the university environment is divided into geographical boundaries based on the locations of the CIPs, each of which may cover a complete university faculty. Therefore each CIP can provide different services to the users, for example faculty-based information or general announcements. Figure 6.2 depicts how CIPs could be deployed around the university campus (at key points to provide a uniform service):

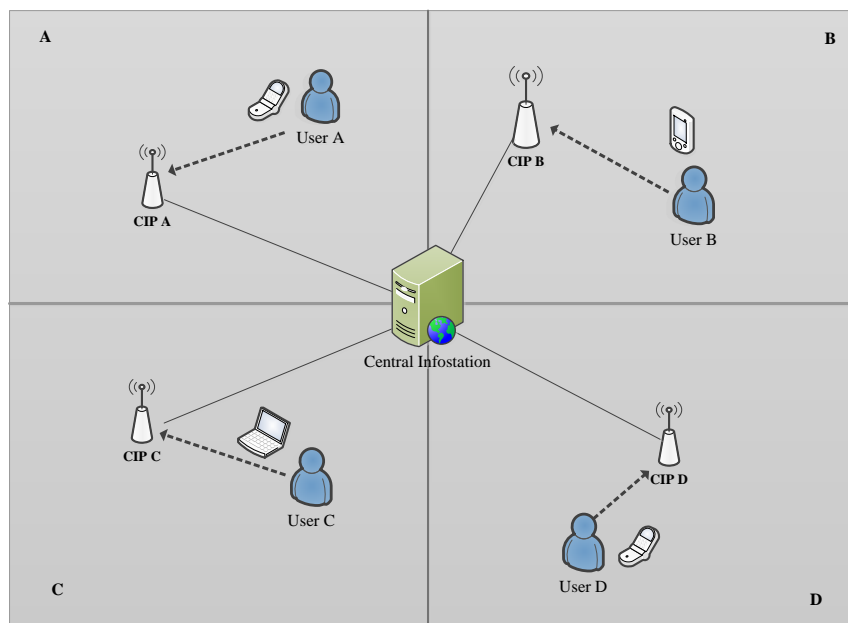


Figure 6.2: University environment divided in accordance with deployed CIPs

We begin by putting our focus on a user who is in the service area of one of the university's CIPs. The system under consideration is an M-learning environment, and hence, it is assumed that the service area is defined by the coverage of the wireless signals from the CIP. The next step starts with the user trying to access a system service; before being able to access a service, the user first needs to register with the system through the CIP. All user information will be passed to the CenStat, which will check whether the user is an existing one or new, and then will search the user identity in its database of existing users.

In the case of using an M-learning system for the first time, the user will be required to complete user details on a special entry page; both these details and the user CI will be processed and stored in the CenStat database. In another scenario, if the user is already registered, the system will gather the user CI, such as location, time, user name, user status, etc., and create a special record that contains the entire user CI during the connection. Once the registration is approved, the m-system will send an acknowledgement to the user and display all the available functions/services. In this case, the system will constantly monitor the user's moves and update his/her newly created record accordingly. The user is therefore allowed to choose any available system service and can access it through their mobile device.

Once the user requests a service, the CIP will pass this request to the CenStat, which in turn will check whether or not the current user CI is sufficient to grant the required service. If so, the service will be granted, otherwise the user request will be rejected and the user will be informed of the reason.

Whilst accessing the M-learning system, the CIP will keep track of the user moves as well as monitor any user CI changes, and will then update the CenStat accordingly; all user information will be saved in the user-created record. Moreover, the CIP will also check the user's device type, and then pass it to the CenStat in order to respond to the user with the most applicable format.

6.5 M-learning system users and functions

It is imperative to State that we assume that all the user duties must be performed within the university campus. The M-learning system is mainly designed to be used by the following users:

- Student: These are intended to be the main users of the M-learning system; they have a variety of services available to them. The students are classified

based on their Study Academic Year (S.A.Y), for example foundation year, first year, second year and final year, therefore each of the M-learning system services is designed accordingly to the following:

- Download materials: this service allows the students to access their lectures using portable devices.
 - View materials: this service allows the students to display the relevant course materials. Therefore the user device should be compatible with the required materials.
 - Make test: having a testing service is crucial to the efficacy of the M-learning system. The m-test allows the students to take their exams and submit them, and then receive their results via their mobile devices.
 - Make Private chat: This service provides a private forum for two (or more) students within the same CIP range. Thus, the student is free to choose from a list of available students with whom to chat. In addition, the user can select chatting types, such as the face-time service for privacy purposes; however, this service will not always be available as the system will have to check that there is no other student nearby.
- Lecturers: a lecturer is allowed to perform only two functions in the M-learning system: upload materials and mark student exams.
 - Upload materials: the lecturer can utilise this service in order to upload both lecture materials and student exams; the uploaded files should be consistent with all lecture formats, whether text, video or audio. This service requires a high connection speed in order to quickly upload all the necessary materials.

- Mark student exams: this service enables the lecture to access the completed student exams and mark them. This needs a high speed and secure connection.
- Make Private chat: this is the same service that is provided to the students, as mentioned above.

6.6 Capturing the key security requirements for M-learning system functions

The M-learning system is subject to all the key security requirements treated earlier in this thesis (Authentication, Authorisation, Confidentiality and Integrity). As illustrated above, the university boundary is divided into four distinct geographical regions, and these are named A, B, C and D. Each geographical region may contain one university faculty, and thus each part has special and different privileges based on faculty regulations and policies. Accordingly, these policies will assist in determining the security requirements for the M-learning system functions, and examples for this system are given below.

6.6.1 Authentication

This is an essential requirement for any M-learning system, and is invoked at the earliest stage of using the system. This step is designed to guarantee that the accessing user is not an imposter; only after this has been satisfied is the user allowed to access the system and view its functions. Thus, in order to authenticate system users, accordingly two kinds of context parameters are specified that must be addressed in verifying the user identity, as follows:

- Static parameters: user name/password.

- Dynamic parameters: user location.

In this case study, the verification of both parameters are required. Thus, the user is required to enter the username and password, and the CAS verifies the user's location in order to check whether or not s/he is within the allowed boundary. Only once the appropriate and valid details are provided, can the user access the CASs. The CAS will continue to monitor the user's movement, and therefore once the CAS senses and realises that the user has left the CASs signal coverage area, the service is terminated accordingly. Therefore, to continue accessing that service, s/he will need to register again.

6.6.2 Authorisation

This type of security requirement follows the authentication stage; it is devoted to managing access to the system functions. Therefore, in this case study, we utilize certain CI to achieve this, for instance, user location, time and user status. Thus, the user must address all of these predetermined CI data in order to access the CAS functions. However, each function requires its own CI in order for it to be accessed, as follows:

- Download Materials: firstly, this service can be only delivered to the user once s/he is enrolled in the relevant faculty. Therefore the CAS will check the user status first to ensure that s/he belongs to the relevant faculty. The user is required to be within the university campus boundary whilst downloading materials. In respect of time and university regulations, the student must typically invoke this service between 8:00 and 21:00.
- View Materials: in order to invoke this service the user must be within the university boundary.

- Do exam: this service is strictly controlled by the CAS; it prevents the use of any other CAS services during the exam. Consequently, the CAS changes the student status to *<< Offline >>* once the exam has started. Moreover, this service is also subject to a specific timeframe which means there is limited time to start and finish the exam. Also, the user must be located in a predefined location.

In this case, the CAS will verify each student's identity using some physical identification methods, such as speech-recognition or fingerprint. Should the online connection be lost for any reason during the exam, the user must return to the m-test within a predefined number of minutes; otherwise, s/he will forfeit the opportunity to complete the exam on the same day.

In the case of the m-test becoming dysfunctional (and dropping the connection), the saved CI will help the CAS to restart the exam from the point of disconnection. However, this can only be done with human intervention based on approval by the invigilator.

- Upload Materials: this service is divided into two main services as follows:
 - Upload lectures: to invoke this service, the lecturer is required to be within the university boundary.
 - Upload exam: this service is crucial, and therefore requires strict constraints. The lecturer must be located in his/her office and not be connected to any other CAS service; accordingly, the lecturer's status will automatically become *<< Offline >>*.
- Mark student exam: this service also is designed for the lecturer, and can only be performed in the lecturer's office, and thus the lecturer's status will accordingly be *<< Busy >>*; therefore, s/he must not be connected to any

other service.

- Make private chat: the service is only available using the university network; also, the user status must be online to establish any chat.

6.6.3 Confidentiality

This is used to protect sensitive user information from unauthorised persons. In an M-learning environment, the exam tests are most sensitive, and the examiner must keep them confidentially, from student specially. The user location is the key context element in achieving this. We will accordingly consider the M-learning service that needs high levels of confidentiality.

- Upload exam: this service is devoted to lecturers, and to favour confidentially of the exam tests, the key condition to access this service is that it must be done in the lecturer's office. This means that only the office is a consider a safe location, and therefore if the lecturer invokes this service and moves outside his/her office, the CAS will sense that, and will then hide any test details accordingly.

6.6.4 Integrity

This type of security requirement concerns ensuring that the user data cannot be modified during transmission over work, and it thereby allows the user to move within the system boundary and to switch from one CIP to another. The purpose of an M-learning system is to send and receive data, and during these operations the CAS must check that the data are not modified or corrupted. Accordingly, in this case study, we seek to protect the user's connection whilst utilising the M-learning system; in addition, the data must not be affected once the CI has changed. For example, once the user connects to a CIP and starts invoking a CAS function, and

should that user then move from one CIP to another, the CAS must ensure that no data are altered during the transition.

6.7 Modeling the M-learning system using UML

This section demonstrates on the M-learning case study the proposed extensions and enhancements that have been done in this thesis on the UML diagrams in order to model a CAS. We have seen that UML can be utilised to model and specify CAS functions and behaviour. UML has several diagrams, each of which is designed for specific purposes. This case study by means of Use Case diagram, Activity diagram and State diagram. Therefore the aforementioned functions will be fully presented by these diagrams. As documented fully in the preceding chapters, the Use Case notations are not mature enough to cope with and present CAS functions or to define their security requirements, and consequently extensions have been advanced to enable the Use Case notations to address this shortcoming. This then necessitated developing corresponding extensions for both the Activity and State diagrams to support the Use Case extension (Chapter 4 and 5).

6.7.1 Use Case diagram for the system

This diagram of UML is mainly concerned with defining system functions as well as capturing security requirements from the user's point of view. Therefore, it is the first important step to express the M-learning functions as well as the relevant CI. This is done using the proposed extension, as in Figure 6.3.

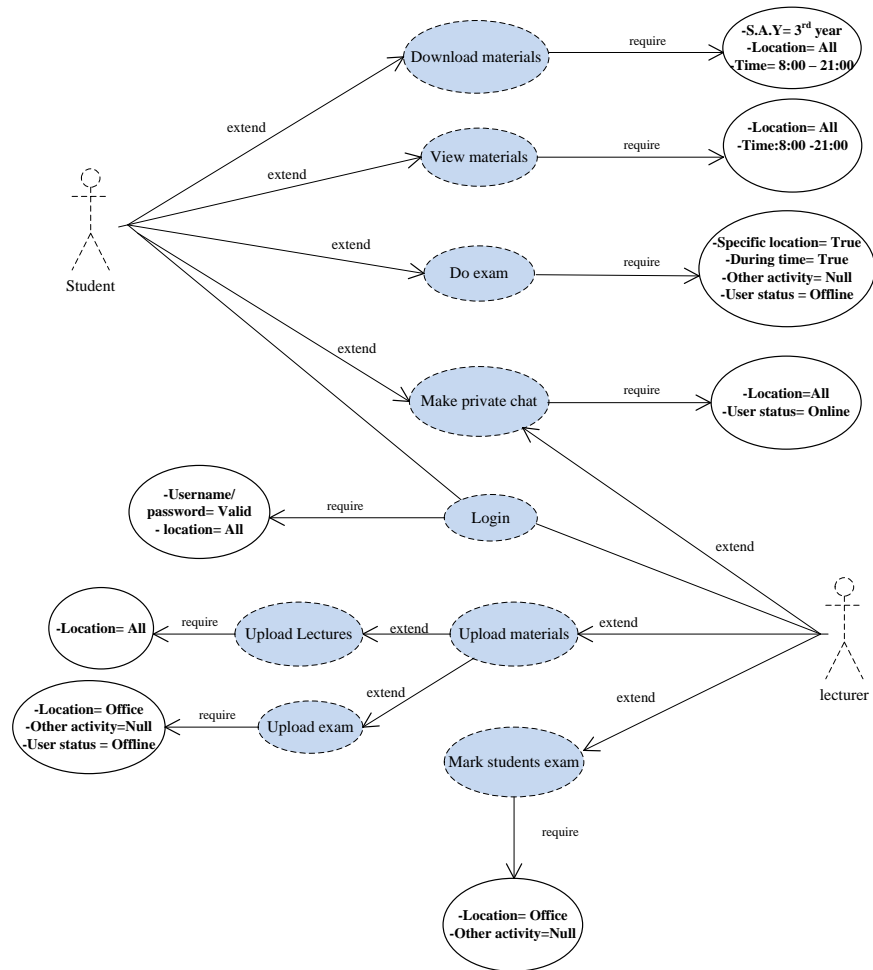


Figure 6.3: Main Use Case diagram for M-learning

The diagram in Figure 6.3 shows the M-learning system Actors and their possible functions. In addition, it shows the required CI to carry out each of these functions. The Tables 6.1, 6.2 list the system Actors, functions and required CI.

Student function name	Required CI
Login	Username/password= Valid. Location= All.
Download materials	Location= All. Time= 8:00 - 21:00. S.A.Y= 3rd year.
View materials	Location= All. Time= 8:00 - 21:00.
Do exam	Specified location= True. During specified time= True. Other activities= Null. User status= Offline.
Make private chat	Location= All. User status= Online.

Table 6.1: Student functions

Lecturer function name	Required CI
Login	Username/password= Valid. Location= All.
Upload lecture	Location= All.
Upload exam	Location= lecturer's office. Other activity= Null. User status= Offline.
Mark student exam	Location= lecturer's office. Other activity= Null.

Table 6.2: Lecturer functions

We have seen that one of the main function of the M-learning system is << *Download materials* >> Figure 6.4 focuses on it. This function will be modeled in both Activity and State diagrams in the sequel of this Chapter, while all other function models are deferred to the Appendix.

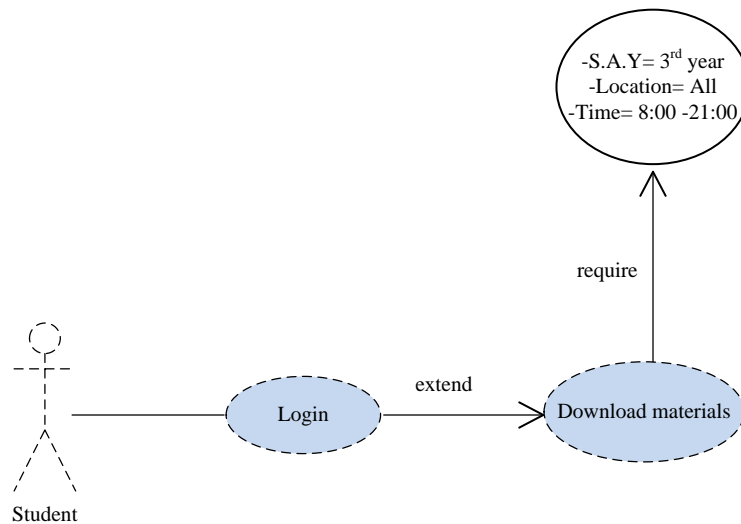


Figure 6.4: Download materials Use Case

The Table 6.3 depicts the description of << *Download materials* >> Use Case diagram.

Name	Description
Actor	Student
Main function	Download materials
Brief description	The student can access the download materials function once he successfully gains the required CI.
Context Information	Location= All. Time= between 8:00 and 21:00. S.A.Y= 3rd year.
Precondition	The student must be firstly authenticated, and be in location A with the time between 8:00 and 17:00.
Post-condition	The student must remain having the proper CI.
Flow of events	<ol style="list-style-type: none"> 1. Student requests download materials function. 2. CAS verifies the student's CI with required CI. 3. CAS provides the service.

Table 6.3: Download materials Use Case description

6.7.2 Activity diagram for the Download Materials function

As described in Chapter 4, we have enhanced also the Activity diagram in order to demonstrate the value of the Use Case enhancement; these can be seen in the Activity diagram Figure 6.5 to describe the scenario of the function << *Download materials* >>.

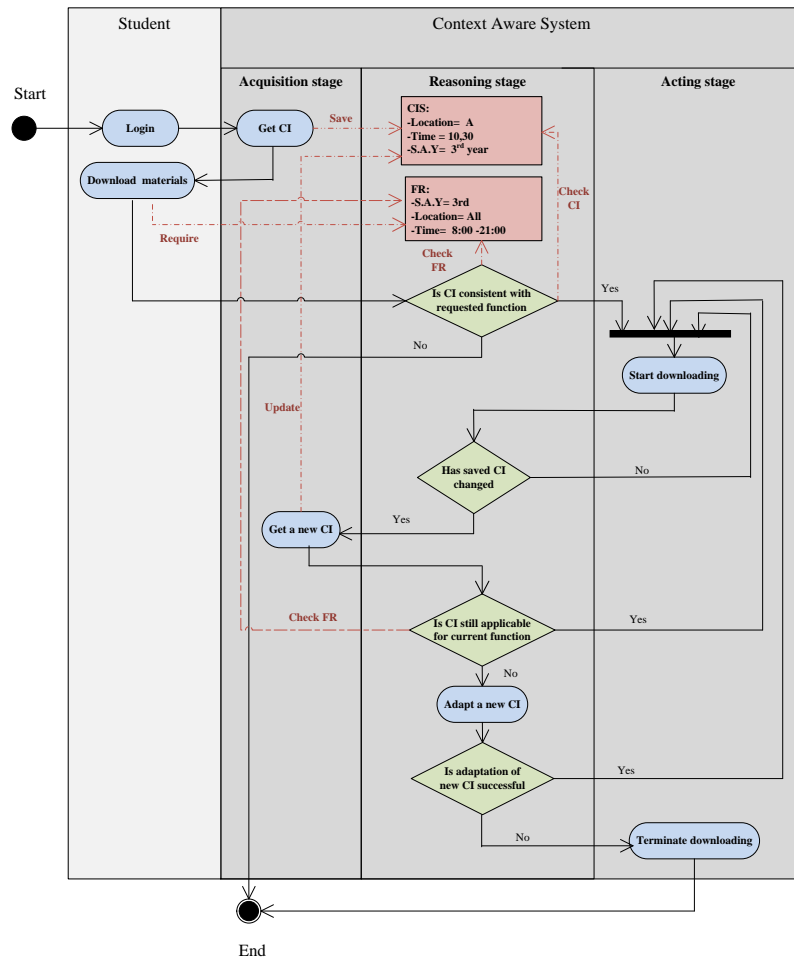


Figure 6.5: Download materials Activity diagram

6.7.3 State diagram for the Download Materials function

As described in Chapter 5, we have enhanced also the State diagram to demonstrate the value of the Use Case extension; therefore, these elements are shown in the State diagram given in Figure 6.6 to model the function $\ll Download\ materials \gg$.

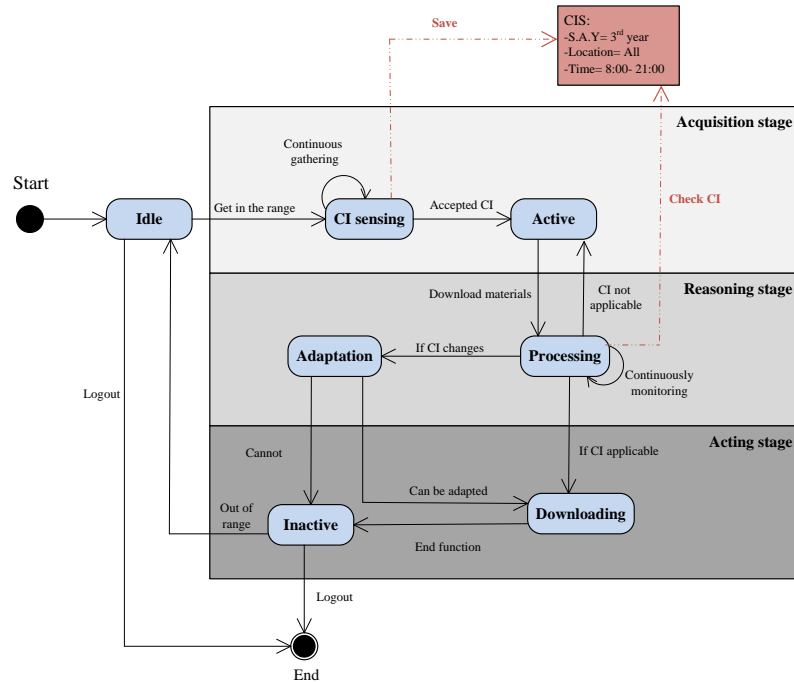


Figure 6.6: Download materials State diagram

To conclude this section, all the proposed extensions and enhancements have been applied for modeling the CAS through the M-learning function $\ll Download materials \gg$.

The extended Use Case diagram shows the required CI that must be fulfilled before performing the requested function. The Activity diagram depicts the flow of the $\ll Download materials \gg$ function together with the effect that CI has on it. Finally, the State diagram illustrates the M-learning states once the user has invoked $\ll Download materials \gg$ as well as the impact that CI has on the states.

6.8 Capturing the security requirements for M-learning functions using UML

This section captures the security requirements for the M-learning system modelled above. This is done by utilising the techniques for capturing security requirements this thesis discussed previously.

6.8.1 Authentication

All UML diagrams developed in the previous chapters will be used here to capture the authentication requirement for an M-learning system.

6.8.1.1 Using our Enhanced Use Case diagram

As explained above in the security requirements section, there are two required forms of authentication in the M-learning system; one of them originates with dynamic behaviour (we saw that the existing Use Case diagram notations are limited in modeling dynamic behaviour). For example, in order to access M-learning, the user is strictly required to be inside the university boundary, and therefore it would be profitable to be able to use some Use Case notations to model this. We observe that this can be clearly presented by the extended Use Case diagram as shown in Figure 6.7.

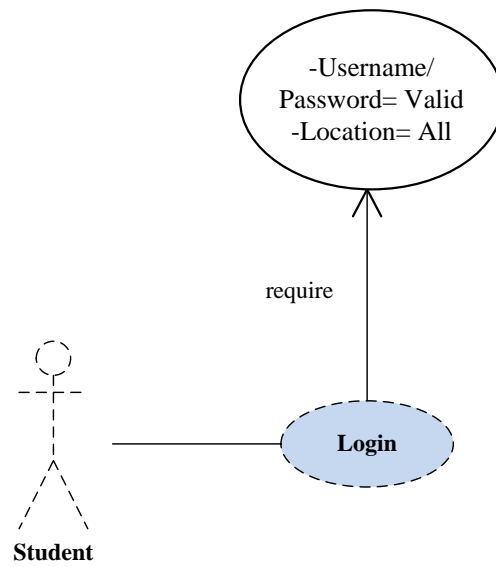


Figure 6.7: Login Use Case

The diagram elements are described in Table 6.4

Name	Description
Actor	Student
Main Function	Login Use Case
Brief description	The student can be verified by entering username/password, and meeting other CI such as location. Once the system approved these, the user can choose one of the available services.
Security Requirement	Authentication
Context Information	Username/Password. Location
Precondition	The Student must be in a certain and known location as well as using an applicant device.
Post-condition	The Student must remain in an authorised location.
Flow of events	<ol style="list-style-type: none"> 1. The user enters username/password. 2. The CAS verifies the user location. 3. The CAS verifies all user access details. 4. The CAS connects the user.

Table 6.4: Authentication Use Case diagram description

6.8.1.2 Using our Enhanced Activity diagram

The CAS is exploited here to present the authentication process through the Activity diagram; therefore, in order to verify the user, the M-learning system checks the parameters username/password and user location, as is clearly shown in the diagram Figure 6.8.

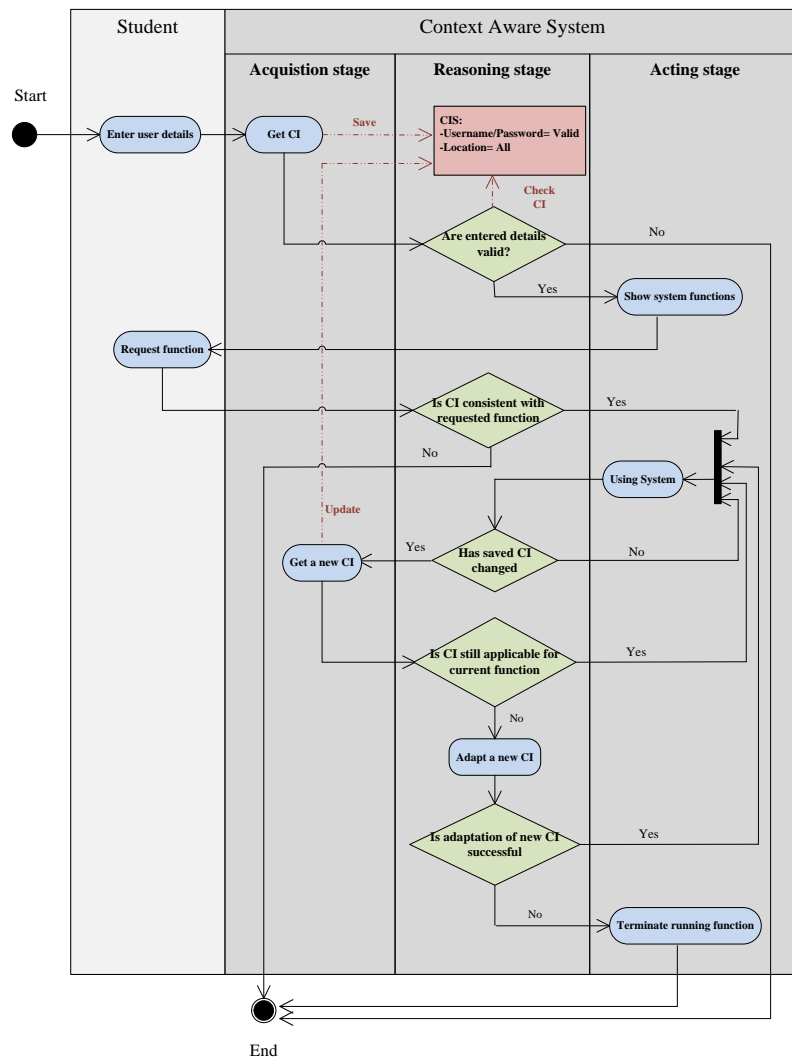


Figure 6.8: Authentication through Activity Diagram

6.8.1.3 Using our Enhanced State diagram

The State diagram for Authentication describes the user states during the Authentication process; the Authentication parameters (username/password, user location) are verified as shown in Figure 6.9.

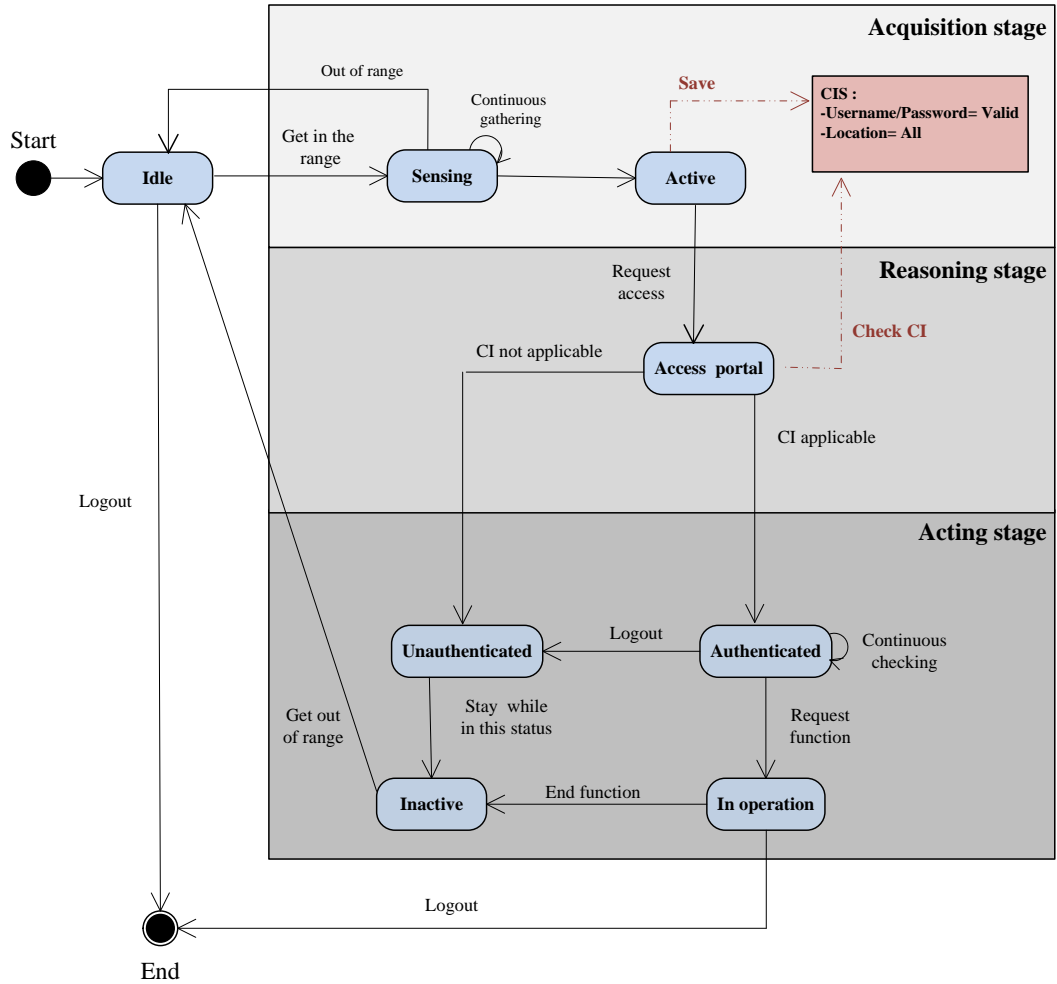


Figure 6.9: Authentication through State diagram

6.8.2 Authorisation

All UML diagrams developed in the previous chapters will be used here to capture the authorisation requirement for an M-learning system.

6.8.2.1 Using our Enhanced Use Case diagram

As described in Chapter 4, it can be assumed that the user will not reach this stage until s/he has been authenticated. We consider that the lecturer wishes to access << *Upload materials* >> and then << *Upload exam* >>; the major reason behind choosing these services rather than any other M-learning functions is to demonstrate the sequence of actions, as these functions also need to present confidentiality and then integrity (as we shall see in the following sections).

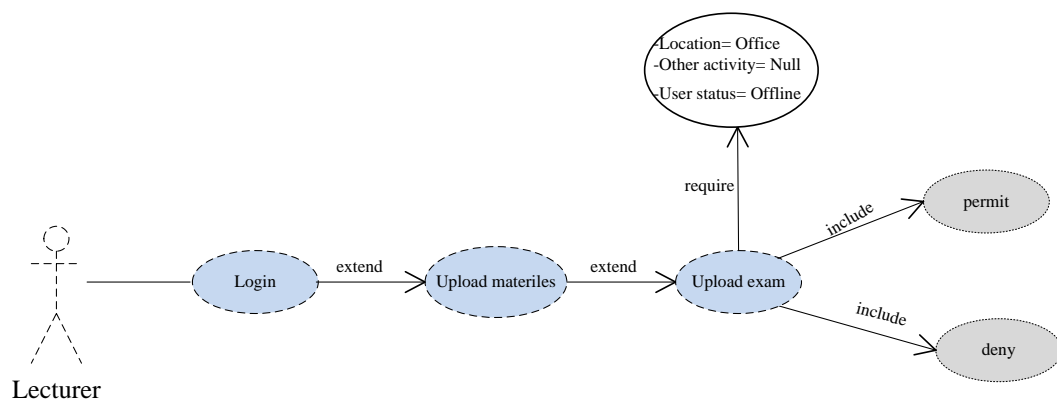


Figure 6.10: Authorisation for upload exam using Use Case diagram

The diagram elements are described in the Table 6.5.

Name	Description
Actor	Lecturer
Main Function	Upload exam
Brief description	The lecturer can access the download materials function once s/he successfully gains the required CI.
Security Requirement	Authorisation
Context Information	Location= Office. Other activity= Null. User status= Offline
Precondition	The lecturer must be in a certain and known location as well as using an applicant device.
Post-condition	The lecturer must be in location A,B as well as during the time between 8:00 and 17:00.
Flow of events	<ol style="list-style-type: none"> 1. User requests download materials. 2. The CAS verifies the user CI. 3. The CAS provides the requested function. 4. The CAS continuously checks the user CI. 5. The CAS denies the service once the user CI is not applicable.

Table 6.5: Authorisation Use Case description

6.8.2.2 Using our Enhanced Activity diagram

The Activity diagram 6.11 shows the authorisation process for managing the function << *Upload exam* >> by following the gathered CI.

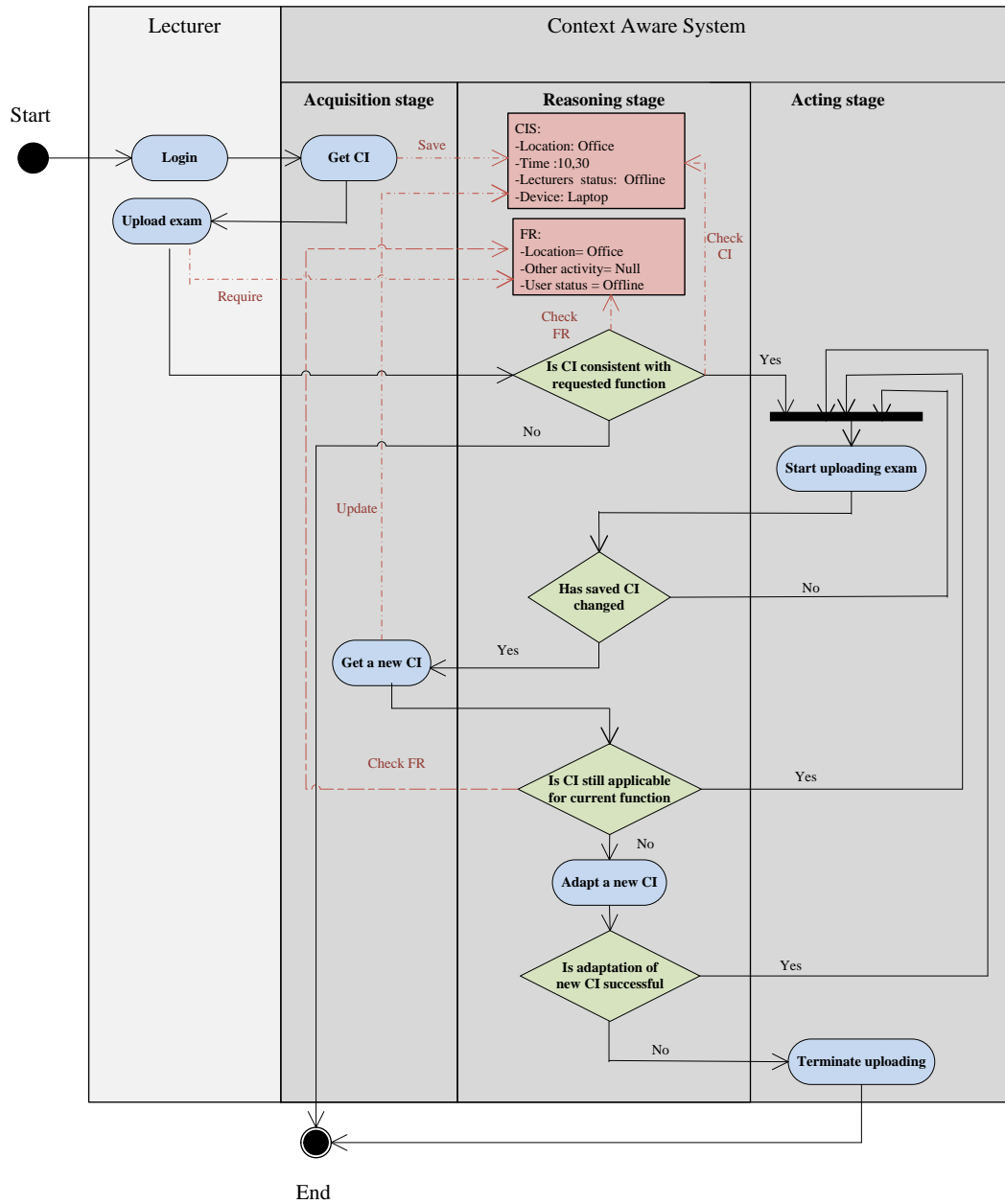


Figure 6.11: Authorisation for upload exam using Activity diagram

6.8.2.3 Using our Enhanced State diagram

The State diagram 6.12 shows the authorisation process for managing the function << Upload exam >> by following the gathered CI.

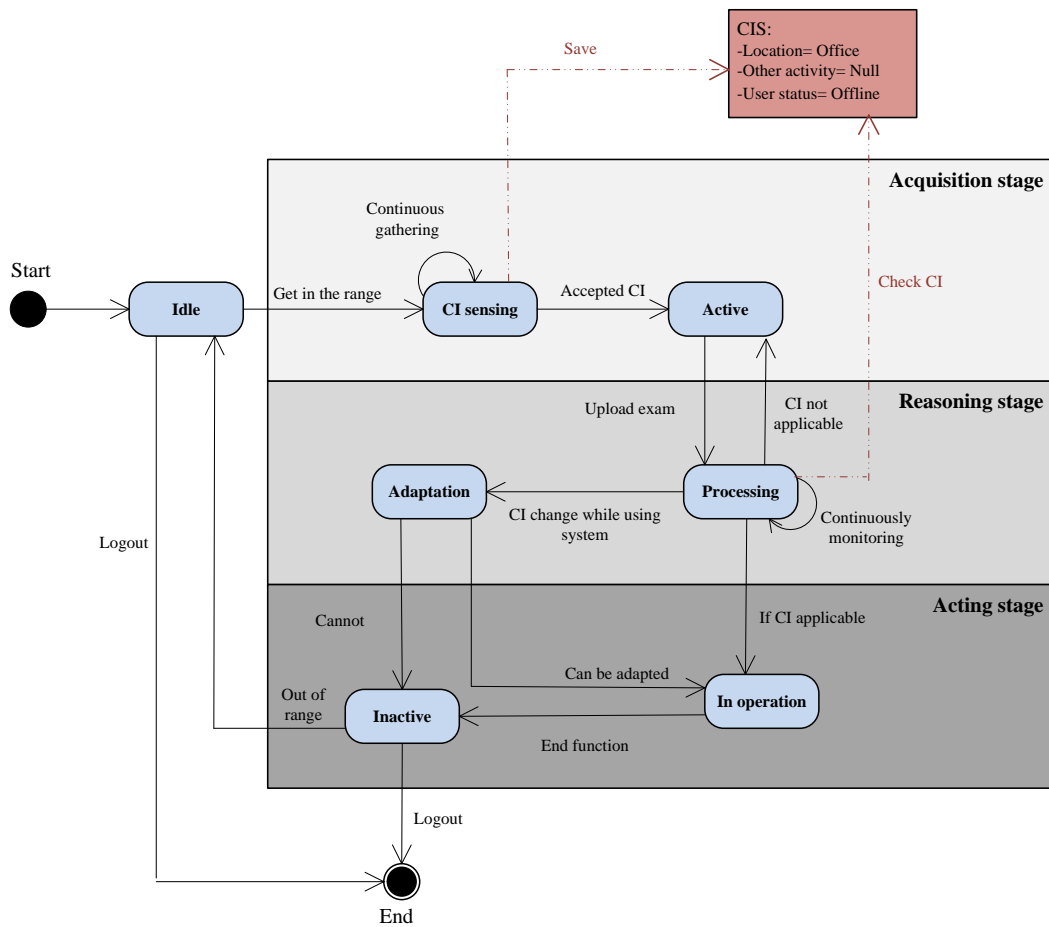


Figure 6.12: Authorisation for upload exam using State diagram

6.8.3 Confidentiality

All UML diagrams developed in the previous chapters will be used here to capture the confidentiality requirement for an M-learning system.

6.8.3.1 Using our Enhanced Use Case diagram

As presented in the above section, $\ll Upload\ exam \gg$ needs certain CI to be performed. The location parameter is one of the required constraints, and we assume that the lecturer has fulfilled the necessary CI; the service is granted accordingly. Therefore, in order to demonstrate the confidentiality mechanism, as shown in Figure 6.13, the M-learning system continuously checks the lecturer's location (i.e. whether or not s/he leaves the office); if the location changes, the exam information will be hidden (otherwise, the service will be provided in full).

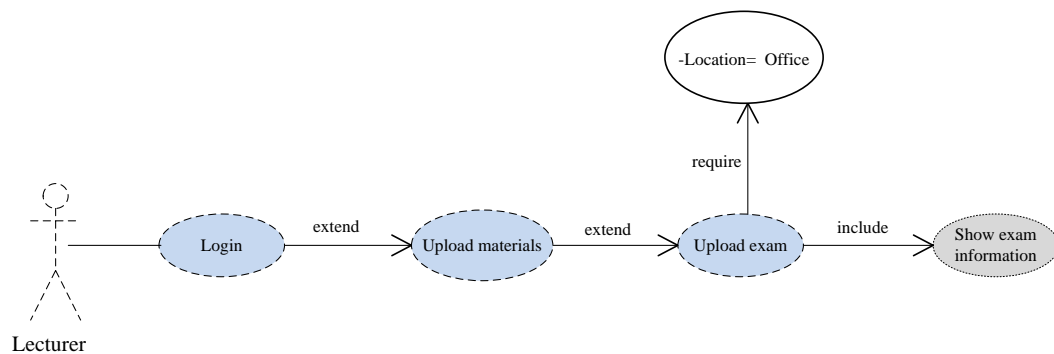


Figure 6.13: Confidentiality for upload exam using Use Case diagram

The diagram elements are describe in Table 6.6.

Name	Description
Actor	Lecturer
Main Function	Upload exam
Brief description	The lecturer can be verified by entering user-name/password, and meeting other context information such as location. Once the system approved these, the user can choose one of the available services.
Security Requirement	Confidentiality
Context Information	Location= Office
Precondition	The Lecturer must be in a certain and known location as well as using an applicant device.
Post-condition	Lecturer must remain in private location.
Flow of events	<ol style="list-style-type: none"> 1. Lecturer performs (upload exam) services. 2. CAS verifies the Lecturer location. 3. CAS realises that the Lecturer sits in the office. 4. CAS provides the whole such service.

Table 6.6: Confidentiality Use Case description

6.8.3.2 Using our Enhanced Activity diagram

The Activity diagram in Figure 6.14 shows the confidentiality process for managing the hiding of sensitive information during the execution of the << *Uploadexam* >> function.

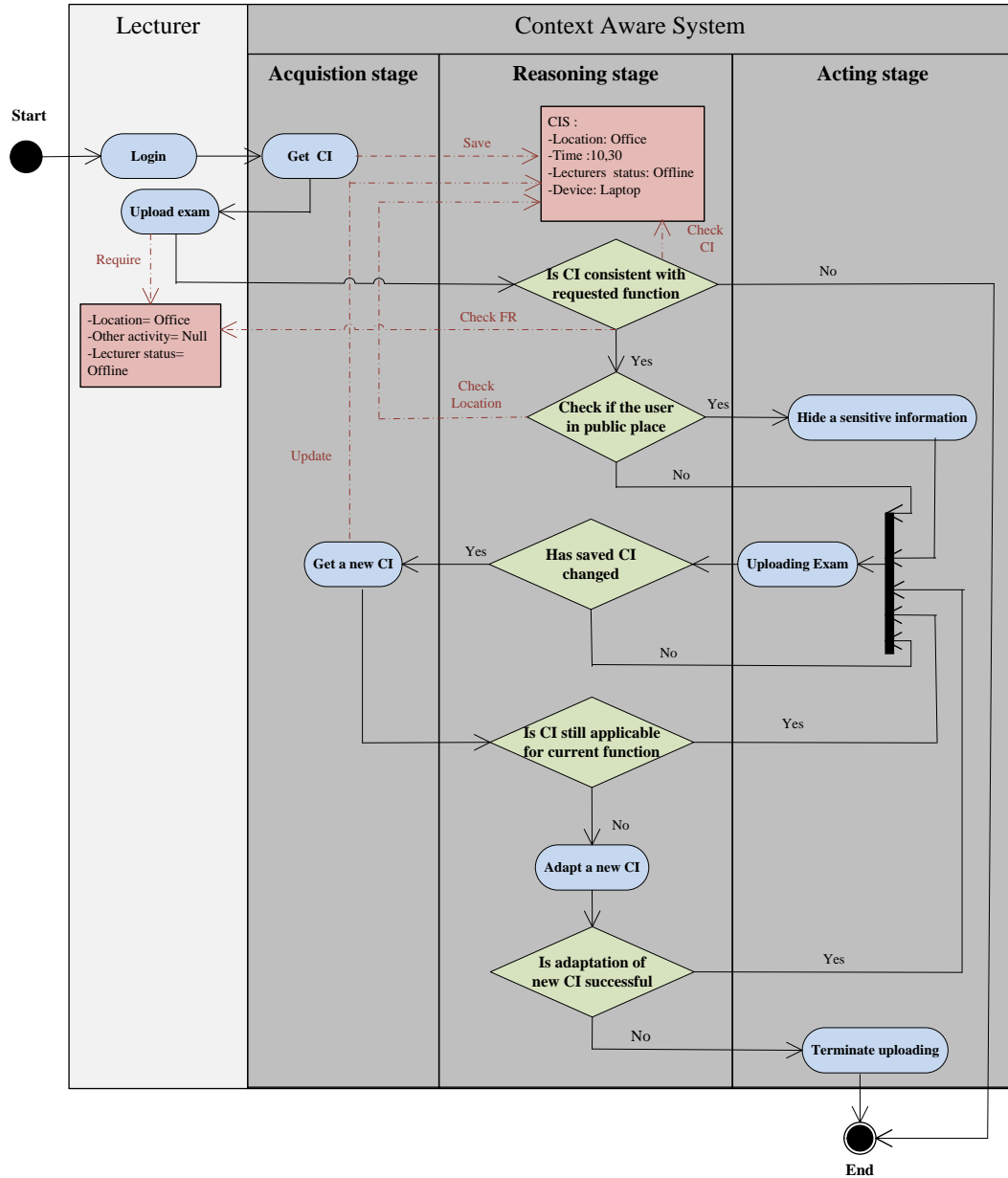


Figure 6.14: Confidentiality for Upload exam using Activity diagram

6.8.3.3 Using our Enhanced State diagram

The State diagram in Figure 6.15 shows the confidentiality process for managing the hiding of sensitive information during the execution of the *<< Upload exam >>* function.

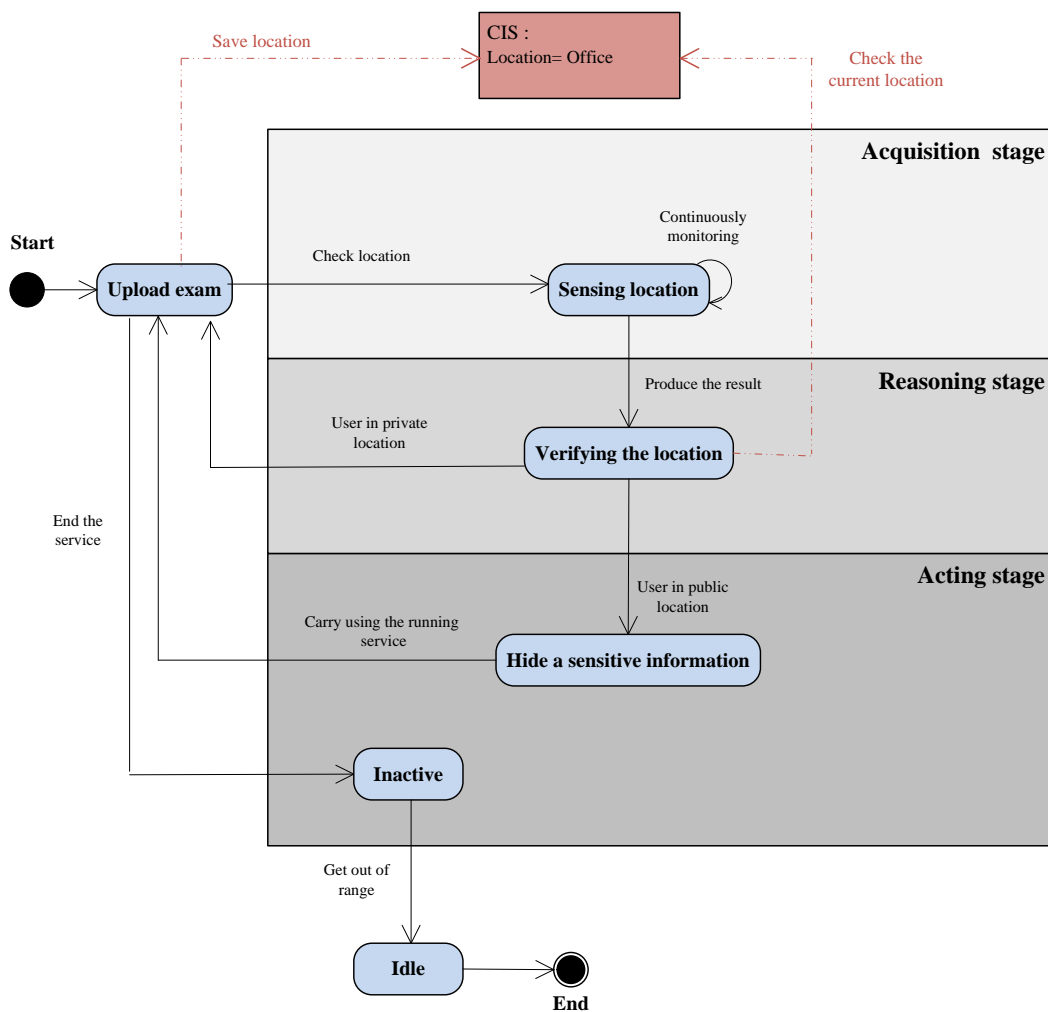


Figure 6.15: Confidentiality for upload exam using State diagram

6.8.4 Integrity

As explained in Chapter 3, the Use Case diagram is inherently inadequate for modeling Integrity, as Integrity behaviour requires tracing a package from its initial point to its end point; the Use Case diagram is unable to present this process, and therefore we do must skip this modeling via a Use Case diagram.

6.8.4.1 Using our Enhanced Activity diagram

The Activity diagram below 6.16 depicts the integrity process for the *<< Upload exam >>* function; we assume that the lecturer has already fulfilled the required CI, and then requests *<< Upload exam >>*. As soon as the service has been provided, the M-learning system generates a table that contains all transaction details. This table will be the key to deciding whether or not the data are corrupted.

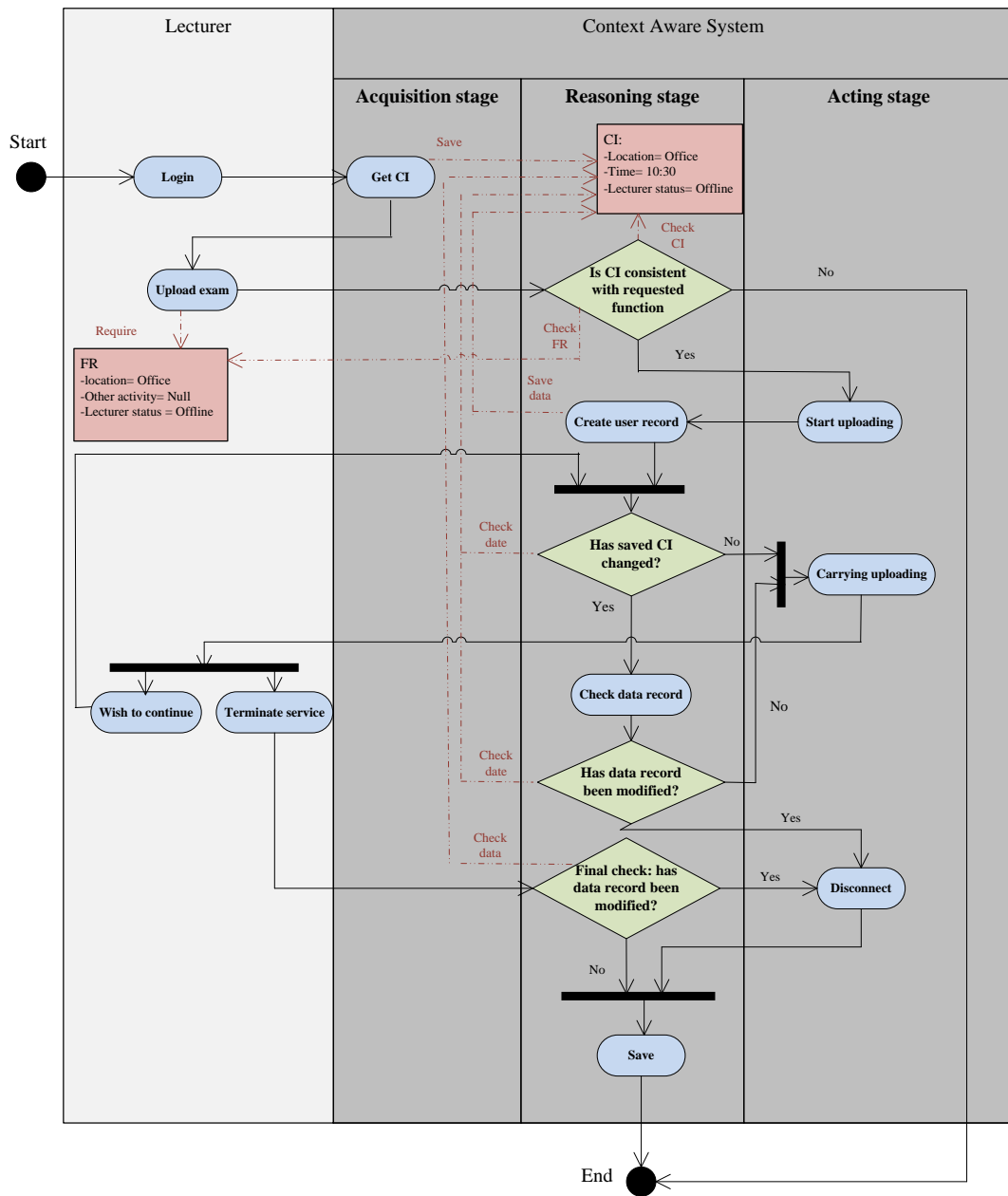


Figure 6.16: Integrity for upload exam using Activity diagram

6.8.4.2 Using our Enhanced State diagram

The State diagram in Figure 6.17 expresses all the states that can present the integrity process for the *<< Upload exam >>* function; it is assumed that the lecturer has already fulfilled the required CI, and then requests *<< Upload exam >>*. As soon as the service has been provided, the M-learning system generates a table that contains all transaction details. This table will be the key to deciding whether or not the data are corrupted.

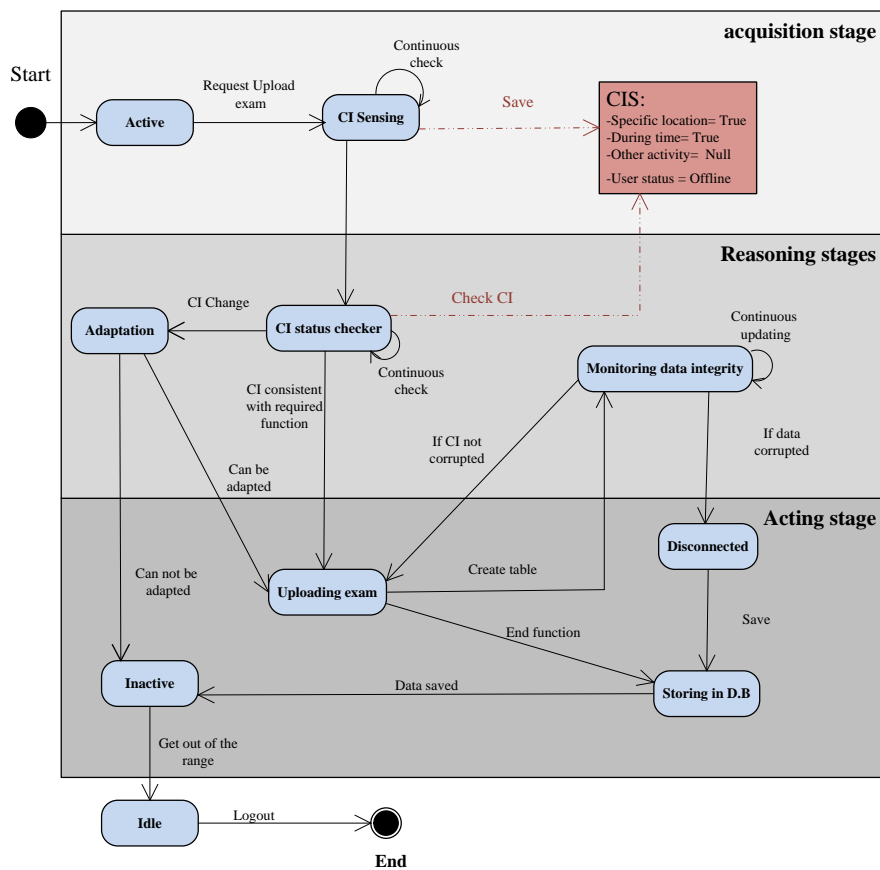


Figure 6.17: Integrity for upload exam using State diagram

6.9 Chapter Summary

The purpose of this M-learning system case study is to demonstrate the proposed extension that was done on the Use Case diagram as well as the refinements that were done on both the Activity and State diagrams; they now have a documented ability to assist in modeling the CAS and capturing its security requirements. We recall that the M-learning system is a type of mobile system, which is applied in the university environment to provide various educational functions to users, including students and lecturers, so its significance is clear today.

This Chapter is divided into several sections; the first section provided an overview and definition of the M-learning system as well as explaining its structures. The subsequent sections illustrated in detail how M-learning works by utilising the infostation system. Then, the modeling of the M-learning system was carried out by exploiting the extended Use Case diagram, and the enhanced Activity and State diagrams. Towards the end of this chapter, we described the security-requirement gathering process for the M-learning system by utilising the enhanced UML diagrams that form the core of this thesis.

Chapter 7

Conclusion and Future work

Objectives:

- To summarise the work carried out in this thesis
 - To list the main contributions of this work
 - To give a statement of evaluation
 - To revised the research questions
 - To sketch the future work
-

7.1 Research Summary

CAS are dynamic in nature and modelling such systems is not trivial. Security is considered one of major challenges in CAS specially because such systems often gather sensitive user information; this information may compromise the security of the system if disclosed to unauthorised users. Thus, the design of a CAS must consider system security as a major requirement. Although security is traditionally considered as a non-functional requirement and is delayed to a later stage of the system development lifecycle, this thesis insists that security must be considered as early as possible because of its high importance. This is also in line with the “secure by design” concept.

Therefore, in this thesis the UML diagrams Use Case diagram, Activity diagram and State diagram have been enhanced in order to enable them to model a CAS and then capture its security requirements at the earliest possible stage of the software development process. The summary of the work presented in this thesis is as follows:

The first Chapter sought to introduce the research motivation, the research questions and scope. Then, we identified the measure of success, and finally outlined the thesis structure.

The second Chapter presented the background of the topics discussed in this thesis. The main topics covered are modelling language ways with a detailed overview on UML and its diagram types such as Use Case diagram, Activity diagram and State diagram. The chapter also discussed security requirements and context-aware systems on its own, and then finally reviewed the work related to security requirements and context-aware systems using UML.

The third Chapter initially identified the shortcomings that Use Case diagram suffer in modelling CAS's and their security requirements. In the later part of this chapter, we have described the proposed enhancements to address these limitations;

and then presented some examples to demonstrate how these enhancements can be used to model CAS's and their security requirements.

In Chapter 4, initially an enhancement to Activity diagram was proposed; this enhancement proposed a framework, which enables to model CAS's. This enhancement to activity diagrams introduces two separate swim lanes: one representing the user functions (for e.g. login, request function) and the second represents the CAS functions (for e.g. get CI). The CAS Swim lane is further sub-divided into three swim lanes: Acquisition, Reasoning and Acting. These three sub-swim lanes represents the CAS complete life cycle. These Activity diagrams were used not only to support the previously enhanced Use Case diagrams but also to tackle the security requirements for CAS's.

In the fifth Chapter, we continued our line of enhancements over State diagram notations. As a result, it became possible to model CASs behaviour, these new enhancements make it possible to define states which are dynamic and is based on users CI. Just like Activity diagram, the State diagram is divided into three different levels to represent the complete life cycle of a CAS. This new addition to the State diagram, also enables it to gather the security requirements of CAS's.

In the sixth Chapter, we applied all the proposed work to an M-learning system case study. This particular M-learning system represents a University boundary, where online provision of resources is a requirement for both the students and the staff. Also, this requires separate security provisions for different type of mobile users with changing context in both spatial and temporal domain.

Thus, modelling of this system and gathering of security requirements shall fully cater for the CI. This M-learning system is represented with our proposed enhancements, with the extended Use Case diagram and showing how the functions of a CAS can be modeled, and then continuing on by capturing its security requirement. Then, we presented one of the main Use Case functions using the proposed Activity

diagram for the sake of demonstration, deferring a similar treatment of the other functions to the Appendix. Finally we modeled the example Use Case function by the enriched State diagram. In the end of this Chapter, we defined how the security requirements can be gathered for the example M-learning system using our full set of enhancements.

7.2 Statement of Evaluation

UML is universal graphical modeling language [113], [67], which contains several diagrams. UML is generally used to model systems and define requirements on system functions. UML has been utilised and extended to be appropriate for such areas. In both CAS and security requirements areas, the UML also has been enhanced for many purposes such as formalisation aspect and to show adaptation mechanism [23], [81], [61].

To the best of our knowledge, there is no a single study devoted to model security requirement for CAS using UML diagrams types (Use Case, Activity and State). These diagrams were found to be severely limited to present CASs behaviour; also this was found that it is incapable to show the effectiveness of changing CI on CASs functions.

There was a requirement to enhance these diagrams to present CAS behaviour. In the work presented the said UML diagrams were enhanced in order to overcome those limitations and then also to gather the key security requirements at the early stage of developing CASs.

The proposed enhancements have added at least the following features:

- Enabling the Use Case diagram technique to properly define and model CAS functions and the required CI to perform that functions, and allowing for the gathering of the security requirements for CASs at such an early stage of

software development.

- Enabling the Activity diagram technique to model CASs as well as to show the dataflow while executing any system function, and then clearly capturing CASs security requirements.
- Enabling the State diagram technique to present the various objects in a CAS by means of a general skeleton to account for the influence of IC in a CAS and to gather their key security requirement.

Furthermore, in regard to the chosen case study that has been presented in this thesis, which is probably correct. The M-learning was modelled based on our proposed enhancements on Use case diagram, Activity diagram and State diagram in order to represent some particular scenarios.

7.3 Research Questions Revisited

To evaluate the work presented in the thesis, the research questions formulated in Chapter 1 are revisited here.

The main research question was: How to model CASs and gather its security requirements using UML?

However, this question poses a couple of questions as follows:

Q1. Are the current form of Use Case notations applicable to modeling a CAS?

Q2. Can the Use Case notations be extended to model a CAS?

Q3. Is the extended Use Case diagram capable to capture all the main security requirements, namely (Authentication, Authorisation, Confidentiality and Integrity)?

All of the above questions were fully answered in Chapter 3 by means of our enhancements to the Use Case diagram.

In relation to the other target diagrams Activity and State, more questions arose:

Q4. Are the existing notations of both Activity and State diagram mature to present the extended Use Case diagram?

Q5. Can the current notations of Activity and State diagram be enhanced to do so?

These two questions were answered respectively in chapters 4 and 5.

And finally there was a concluding question in term of practicality of the proposed enhancements, which is:

Q6. Can the proposed extensions for gathering and modeling the security requirements of CAS be practically applicable to real-world case studies?

Chapter 6 presented a case study, where these enhancement have been practically applied to model a M-learning system.

In summary, the proposed enchantments derived from a deep and synergistic understanding of two main aspects: one is the nature of CAS and more specifically the influence that CI plays on the system functions; the other one is the strength and limitations of existing UML diagramming techniques.

7.4 Contribution to Knowledge

This thesis contributes to knowledge at least in the following ways:

- It provides an in-depth understanding of system modeling in general and in particular of pros and cons of UML diagramming techniques.
- It shows the main modeling stakeholders for CAS, with a focus on the role of CI.
- It defines a practical approach to modeling CAS and capture their key security requirement using appropriate enhancements to the most common UML diagram types.

- It provides the full modeling of real-world case study an infostation based M-learning system using these enhancements.

7.5 Future Work

Context-aware systems are gaining increasing interest, at least due to the availability of smart portable devices such as smart phones and mobile services such as electronic learning and electronic banking. On the other hand, UML is one of the most widely adopted modelling techniques for a variety of systems development projects. This thesis began with an under-pinning assumption that while UML is a valuable development tool, UML suffers from a number of limitations in coping with the distinctive features of CAS. Another startling observation of this thesis is that challenging task of capturing information systems security requirements in an early stage of systems development. Despite the above made observations; UML is a flexible tool that can accommodate enhancements to make it fit for the distinctive features of CASs and eliciting and documenting systems security requirements.

The enhancements recommended in this thesis have addressed a set of UML Models (Use Case Modelling, Activity Modelling and State Diagrams); the researcher will undertake a similar effort to identify UML enhancements necessary to Class Diagrams, Sequence Diagrams and Deployment Diagrams to make them fit for both CAS development and security requirements elicitation and modelling.

Another direction of future work is to develop a dedicated Object Constraint Language (OCL) to clearly describe the various specifications of CASs. Security requirements need to be both expressed clearly and functionally modelled into the design of the CAS; hence an OCL will provide such level of sophistication and clear definition of needs.

Another area of future work will require a survey of industrially utilised devel-

opment methods and approaches and an assessment of their ability to successfully develop CASs and elicit their security requirements. Industrially utilised development approaches the likes of Microsoft Development Framework would be candidates for such enquiry and attempts of enhancements.

Another avenue for future work will be to adopt the set of UML enhancements to other aspect of CAS Development in addition to security requirements elicitation and modelling for example control and end user support. CASs pose a set of challenges to systems developers in both the tools used and the approaches deployed. More and more challenges can be identified to CASs for example the challenges of control and ownership in cloud computing, the physical challenges in cross border travel facing mobile phone operators, etc.

Bibliography

- [1] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A mobile context-aware tour guide. *Wireless Networks*, 3(5):421–433, Oct. 1997.
- [2] M. H. Al-Sammorraie. *Policy-based Approach for Context-Aware Systems*. PhD thesis, De Montfort University, September 2011.
- [3] S. Albir. *UML in a Nutshell: A Desktop Quick Reference*. In a Nutshell (o'Reilly) Series. O'Reilly & Associates Incorporated, 1998.
- [4] H. Aldabbas, T. Alwada'n, H. Janicke, and A. Al-Bayatti. Data confidentiality in mobile ad hoc networks. *International Journal of Wireless Mobile Networks (IJWMN)*, abs/1203.1749, 2012.
- [5] K. Alghathbar. Validating the enforcement of access control policies and separation of duty principle in requirement engineering. *Information and Software Technology*, 49(2):142–157, Feb. 2007.
- [6] K. Alghathbar. Enhancement of use case diagram to capture authorization requirements. *Software Engineering Advances, International Conference on*, 0:394–400, 2009.

- [7] K. Alghathbar, C. Farkas, and D. Wijesekera. Securing UML information flow using flowuml. In *Journal of Research and Practice in Information Technology*, pages 229–238. INSTICC Press, 2006.
- [8] K. Alghathbar and D. Wijesekera. Modeling dynamic role-based access constraints using UML. In *proc. of the 1st International Conference on Software Engineering Research & Applications (ICSERA '03)*, San Francisco, CA. June 2003.
- [9] K. Alghathbar and D. Wijesekera. Consistent and complete access control policies in use cases. In P. Stevens, J. Whittle, and G. Booch, editors, *UML*, volume 2863 of *Lecture Notes in Computer Science*, pages 373–387. Springer, 2003.
- [10] J. M. Almendros-Jiménez and L. Iribarne. Describing use-case relationships with sequence diagrams. *The Computer Journal*, 50(1):116–128, Jan. 2007.
- [11] S. Almutairi, G. Bella, and A. Abu-Samaha. Specifying security requirements of context aware system using UML. In *Digital Information Management (ICDIM), 2012 Seventh International Conference on*, pages 259 –265, aug. 2012.
- [12] G. C. J. Anne Ngu. Context aware actors. In *Presented at the Ninth Biennial Ptolemy Miniconference, Berkeley, CA*, 16, February, 2011.
- [13] W. Arzac, G. Bella, X. Chantry, and L. Compagna. Multi-attacker protocol validation. *Journal of Automated Reasoning*, 46(3-4):353–388, 2011.
- [14] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, June 2007.

BIBLIOGRAPHY

- [15] M. Bandinelli, F. Paganelli, G. Vannuccini, and D. Giuli. A context-aware security framework for next generation mobile networks. In *Security and Privacy in Mobile Information and Communication Systems*. Springer Berlin Heidelberg, 2009.
- [16] J. E. Bardram, R. E. Kjear, and M. Pedersen. Context aware user authentication supporting. In *Proximity Based Login in Pervasive Computing, Proc. Ubicomp 2003*, pages 107–123, 2003.
- [17] G. Bella. What is correctness of security protocols?. *Journal of Universal Computer Science*, 14(12):2083–2106, 2008.
- [18] K. Bittner and I. Spence. *Use Case Modeling*. The Addison-Wesley Object Technology Series. Addison Wesley Professional, 2003.
- [19] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.
- [20] J. Booch, G. Rumbaugh and I. Jacobson. *El Lenguaje Unificado de Modelado*. The Addison-Wesley Object Technology Series. Addison-Wesley, 1999.
- [21] F. Braz, E. Fernandez, and M. VanHilst. Eliciting security requirements through misuse activities. In *Database and Expert Systems Application, 2008. DEXA '08. 19th International Workshop on*, pages 328–333, 2008.
- [22] L. Bussard, Y. Roudier, and R. Molva. Untraceable secret credentials: trust establishment with privacy. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 122–126, March 2004.

- [23] J. Choi. Context-driven requirements analysis. In *Proceedings of the 2007 international conference on Computational science and its applications - Volume Part III, ICCSA '07*, pages 739–748, Berlin, Heidelberg, 2007. Springer-Verlag.
- [24] B. Columbia. Activity diagram modeling standards and guidelines version 1.0. *Information and Technology Management Branch*, December 2, 2005.
- [25] B. R. Consensus. Building requirements consensus for business process software requirements. <http://www.building-requirements-consensus.com/>, 2008-2009. [Online; accessed 10-Feb-2011].
- [26] M. Covington, P. Fogla, Z. Zhan, and M. Ahamad. A context-aware security architecture for emerging applications. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 249–258, 2002.
- [27] V. T. da Silva, R. C. Noya, and C. J. P. de Lucena. Using the UML 2.0 activity diagram to model agent plans and actions. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, AAMAS '05*, pages 594–600, New York, NY, USA, 2005. ACM.
- [28] W. Dargie. *Context-Aware Computing and Self-Managing Systems*. Chapman & Hall/CRC Studies in Informatics Series. Taylor & Francis, 2010.
- [29] D. Desmond F. D'Souza and A. Wills. *Objects, components, and frameworks with UML: the catalysis approach*. The Addison-Wesley object technology series. Addison-Wesley, 1999.
- [30] A. K. Dey. Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7, Jan. 2001.

- [31] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, 1999.
- [32] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166, Dec. 2001.
- [33] J. Dong and J. Woodcock. *Formal Methods and Software Engineering: 5th International Conference on Formal Engineering Methods, ICFEM 2003, Singapore, November 5-7, 2003, Proceedings*. Number v. 5 in Lecture Notes in Computer Science. Springer, 2003.
- [34] P. D. Shekar Goud, Ishaq Md. A secured approach for authentication system using fingerprint and iris. *Global Journal of Advanced Engineering Technologies*, Vol1, Issue3, 2012.
- [35] C. L. Dym. *Structural Modeling and Analysis*. Cambridge University Press, 1997.
- [36] L. Engelen and M. van den Brand. Integrating textual and graphical modelling languages. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 253(7):105–120, Sept. 2010.
- [37] T. Farkhani and M. Razzazi. Examination and classification of security requirements of software systems. In *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, volume 2, pages 2778–2783, 2006.
- [38] D. Firesmith, B. Henderson-Sellers, and I. Graham. *Open Modeling Language (OML) Reference Manual*. Sigs Reference Library Series. Sigs, 1998.

- [39] D. G. Firesmith. Analyzing and specifying reusable security requirements. In *Proc. Solid Freeform Fabrication Sym*, pages 507–514, 2003.
- [40] D. G. Firesmith. A taxonomy of security-related requirements. *International Workshop on High Assurance Systems (RHAS'05)*, 2005.
- [41] L. B. Frank Armour and M. Sood. Use case modeling concepts for large business system development. In *OOPSLA 95, Workshop on Use Cases*, 1995.
- [42] I. Ganchev, S. Stojanov, M. O'Droma, and D. Meere. An infostation-based multi-agent system supporting intelligent mobile services across a university campus. *Journal of Computers*, 2(3):21–33, 2007.
- [43] H. gerd Hegering. Management challenges of context-aware services in ubiquitous environments. In *Proceedings of the 14th IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSCOM 2003)*, pages 246–259, 2003.
- [44] I. Global and I. Association. *Enterprise Information Systems: Concepts, Methodologies, Tools and Applications*. Premier reference source. IGI Global, 2010.
- [45] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Transactions on Software Engineering*, 34(1):133–153, 2008.
- [46] L. Han, S. Jyri, J. Ma, and K. Yu. Research on context-aware mobile computing. In *Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 24–30, 2008.
- [47] D. Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, June 1987.

- [48] E. Hayashi, S. Das, S. Amini, J. Hong, and I. Oakley. Casa: Context-aware scalable authentication. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, SOUPS '13, pages 3:1–3:10, New York, NY, USA, 2013. ACM.
- [49] X. He, Z. Ma, W. Shao, and G. Li. A metamodel for the notation of graphical modeling languages. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, volume 1, pages 219–224, 2007.
- [50] K. Henriksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems. In *International Symposium on Distributed Objects and Applications (DOA)*, pages 846–863. Springer, 2005.
- [51] J. I. Hong. An architecture for privacy sensitive ubiquitous computing. In *Proceedings of the 2nd international conference on mobile systems, applications, and services*, pages 177–189. ACM Press, 2004.
- [52] S. H. Houmb, S. Islam, E. Knauss, J. Jurjens, and K. Schneider. Eliciting security requirements and tracing them to design: an integration of common criteria, heuristics, and umlsec. *Requirements Engineering*, 15(1):63–93, 2010.
- [53] J. Hu and A. C. Weaver. A Dynamic, Context-Aware Security Infrastructure for Distributed Healthcare Applications. In *Proc. 1st Workshop on Pervasive Privacy Security, Privacy, and Trust (PSPT)*, 2004.
- [54] V. Illingworth. *A Dictionary of Computing*. Oxford University Press, Incorporated, 1996.
- [55] I. Jacobson. *Object-oriented software engineering: a use case driven approach*. ACM Press Series. ACM Press, 1992.

- [56] I. Jacobson. *The Unified Software Development Process*. Object technology series. Pearson Education, 1999.
- [57] I. Jacobson, M. Ericsson, and A. Jacobson. *The object advantage: business process reengineering with object technology*. ACM press books. Addison-Wesley, 1995.
- [58] H. Janicke, A. Cau, F. Siewe, and H. Zedan. Dynamic access control policies: Specification and verification. *The Computer Journal*, 56(4):440–463, 2013.
- [59] P. Jones. *Fundamentals Of Object-Oriented Design In UML*. Pearson Education, 2000.
- [60] J. Juerjens. Using UMLsec and goal trees for secure systems development. In *Proceedings of the 2002 ACM Symposium on Applied Computing, SAC '02*, pages 1026–1030, New York, NY, USA, 2002. ACM.
- [61] J. Juerjens. *Secure Systems Development with UML*. SpringerVerlag, Berlin, Heidelberg, 2003.
- [62] J. Juerjens and P. Shabalín. Automated verification of UMLsec models for security requirements. In *UML 2004 . The Unified Modeling Language. volume 2460 of LNCS*, pages 412–425. Springer, 2004.
- [63] J. Jurjens. Umlsec: Extending UML for secure systems development. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 412–425, London, UK, UK, 2002. Springer-Verlag.
- [64] M. Kang and K. Taguchi. Modelling mobile agent applications by extended UML activity diagram. In *ICEIS (4)*, pages 519–522, 2004.

BIBLIOGRAPHY

- [65] M. Kang, L. Wang, and K. Taguchi. Modelling mobile agent applications in UML2.0 activity diagrams. *In: Proc. of 3rd SELMAS Workshop at ICSE*, April 2004.
- [66] N. Koblitz and A. Menezes. Another look at security definitions. *IACR Cryptology ePrint Archive*, 2011:343, 2011.
- [67] J. Krogstie and S. Telecom. Evaluating UML using a generic quality framework. In *Chapter in UML and the Unified Process*, Idea Group Publishing, pages 1–22. Press, 2003.
- [68] S. Lee, S. Park, and S. goo Lee. A study on issues in context-aware systems based on a survey and service scenarios. In *Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 2009. SNPD '09. 10th ACIS International Conference on*, pages 8–13, 2009.
- [69] W. Li and A. Joshi. Security issues in mobile ad hoc networks- a survey. Dept. Computer Science Electrical Engineering, University of Maryland, Baltimore County., 2007.
- [70] J. C. D. Lima, C. C. Rocha, I. Augustin, and M. A. R. Dantas. A context-aware recommendation system to behavioral based authentication in mobile and pervasive environments. In *Proceedings of the 2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing, EUC '11*, pages 312–319, Washington, DC, USA, 2011. IEEE Computer Society.
- [71] S. Loke. *Context-Aware Pervasive Systems: Architectures for a New Breed of Applications*. Taylor & Francis, 2006.
- [72] I. Lutkebohle. UML Statechart Diagram. http://www.tutorialspoint.com/uml/uml_statechart_diagram.htm/, 2008. [Online; accessed 19-July-2012].

- [73] A. R. Masoumzadeh, M. Amini, and R. Jalili. Context-aware provisional access control. In *Proceedings of the Second International Conference on Information Systems Security, ICISS'06*, pages 132–146, Berlin, Heidelberg, 2006. Springer-Verlag.
- [74] M. C. Matthew, YMatthew, J. Moyer, and M. Ahamad. Generalized role-based access control for securing future applications. In *Proceedings of the National Information Systems Security Conference (NISSC)*, October 2000.
- [75] J. Mcdermott and C. Fox. Using abuse case models for security requirements analysis. In *Proceedings 15th IEEE Annual Computer Security Applications Conference*, 1999.
- [76] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.
- [77] A. W. Min, R. Wang, J. Tsai, M. A. Ergin, and T.-Y. C. Tai. Improving energy efficiency for mobile platforms by exploiting low-power sleep states. In *Proceedings of the 9th conference on Computing Frontiers, CF '12*, pages 133–142, New York, NY, USA, 2012. ACM.
- [78] M. Mori. A software lifecycle process for context-aware adaptive systems. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ESEC/FSE '11*, pages 412–415, New York, NY, USA, 2011. ACM.
- [79] B. Morin, T. Mouelhi, F. Fleurey, Y. Le Traon, O. Barais, and J.-M. Jézéquel. Security-driven model-based dynamic adaptation. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ASE '10*, pages 205–214, New York, NY, USA, 2010. ACM.

- [80] P. Muller. *Instant UML*. Springer-Verlag New York Incorporated, 1997.
- [81] H. Omasreiter and E. Metzker. A context-driven use case creation process for specifying automotive driver assistance systems. In *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*, pages 334–339, 2004.
- [82] F. Paganelli, G. Bianchi, and D. Giuli. Context model for context-aware system design towards the ambient intelligence vision: Experiences in the etourism domain. In *Universal Access in Ambient Intelligence Environments, 9th ERCIM Workshop on User Interfaces for All (ERCIM UI4ALL), Lecture Notes in Computer Science*, pages 173–191. Springer-Verlag, 2006.
- [83] G. Paleologo, L. Benini, A. Bogliolo, and G. De Micheli. Policy optimization for dynamic power management. In *Design Automation Conference, 1998. Proceedings*, pages 182–187, 1998.
- [84] J. Pascoe. Adding generic contextual capabilities to wearable computers. In *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*, pages 92–99, 1998.
- [85] J. Pavlich-mariscal, L. Michel, and S. Demurjian. Enhancing UML to model custom security aspects. In *Proceedings of the 11th International Workshop on Aspect-Oriented Modeling*,, 2007.
- [86] A. Ranganathan and R. H. Campbell. A middleware for context-aware agents in ubiquitous computing environments. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, pages 143–161, New York, NY, USA, 2003. Springer-Verlag New York, Inc.
- [87] M. Rayner, B. A. Hockey, N. Chatzichrisafis, and K. Farrell. OMG unified modeling language specification. In *Version 1.3, 1999 Object Management Group, Inc*, 2005.

- [88] Renaissance. Technology briefing report system modelling. Technical report, School of Computing and Communication, Lancaster university, 15 Sep 1996.
- [89] A. Rodraguez, E. Fernandez, E indez Medina, and P. Mario. M-bpsec: A method for security requirement elicitation from UML 2.0 business process specification. In *Advances in Conceptual Modeling Foundations and Applications*, volume 4802, pages 106–115. Springer Berlin Heidelberg, 2007.
- [90] A. Rodríguez, E. Fernández-Medina, J. Trujillo, and M. Piattini. Secure business process model specification through a UML 2.0 activity diagram profile. *Decision Support Systems*, 51(3):446–465, June 2011.
- [91] A. Rodriguez, E. Fernandez Medina, E.Medina, and M. Piattini. Security requirement with a UML 2.0 profile. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, pages 8 pp.–, 2006.
- [92] J. Roff. *UML: A Beginner's Guide*. Beginner's Guide. McGraw-Hill Education, 2003.
- [93] M. Roman, C. Hess, R. Cerqueira, R. H. Campbell, and K. Nahrstedt. Gaia a middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, 1:74–83, 2002.
- [94] D. Rosenberg and M. Stephens. *Use Case Driven Object Modeling with UML Theory and Practice*. Books for professionals by professionals. Apress, 2007.
- [95] C. Ruan. UML specification of e consent requirements in a health care system. In *Computer Science and its Applications. 2008. CSA 08. International Symposium on*, pages 275 –280, oct. 2008.

- [96] F. Ruiz, F. Harmelen, M. Aben, and J. Plassche. Evaluating a formal modelling language. In L. Steels, G. Schreiber, and W. Velde, editors, *A Future for Knowledge Acquisition*, volume 867 of *Lecture Notes in Computer Science*, pages 26–45. Springer Berlin Heidelberg, 1994.
- [97] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.
- [98] N. Ryan, J. Pascoe, and D. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications and Quantitative Methods in Archaeology (CAA 97)*, Oxford, 1997.
- [99] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Fourth ACM Symposium on Operating System Principles (October 1973)*. Revised version in *Communications of the ACM* 17, 7 (July 1974)., 63(9):1278–1308, 1974.
- [100] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, WMCSA '94, pages 85–90, Washington, DC, USA, 1994. IEEE Computer Society.
- [101] B. Schilit and M. Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, 1994.
- [102] A. Schmidt, K. A. Adoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, and W. V. D. Velde. Advanced interaction in context. In *In Proceedings of First International Symposium on Handheld and Ubiquitous Computing*, pages 89–101. Springer Verlag, 1999.

- [103] N. Shankar, D. Balfanz, and N. Shankar. Enabling secure ad-hoc communication using context-aware security services (extended abstract). In *Proceedings of UBICOMP2002 - Workshop on Security in Ubiquitous Computing*, 2002.
- [104] Q. Sheng and B. Benatallah. ContextUML: a UML-based modeling language for model-driven development of context-aware web services. In *Mobile Business. ICMB 2005. International Conference on*, pages 206–212, 2005.
- [105] B. Shneiderman. *Designing the user interface: strategies for effective human-computer-interaction*. Number v. 85. Addison Wesley Longman, 1998.
- [106] F. Siewe, H. Zedan, and A. Cau. The calculus of context-aware ambients. *Journal of Computer and System Sciences*, 77(4):597–620, July 2011.
- [107] G. Sindre and A. L. Opdahl. Eliciting security requirements by misuse cases. In *Technology of Object-Oriented Languages and Systems, 2000. tools-Pacific 2000. Proceedings. 37th International Conference on*, pages 120–131, 2000.
- [108] M. I. Smriti Jain. Software security requirements gathering instrument. *International Journal of Advanced Computer Science and Applications (IJACSA)*, Vol. 2, No. 7, 2011.
- [109] F. Stajano. *Security for Ubiquitous Computing*. John Wiley and Sons, Feb. 2002.
- [110] P. Stevens and R. Pooley. *Using UML software engineering with object and components*. Addison-Wesley, 2006.
- [111] I. Tondel, M. Jaatun, and P. Meland. Security requirements for the rest of us: A survey. *Software, IEEE*, 25(1):20–27, 2008.
- [112] A. Toninelli, R. Montanari, L. Kagal, and O. Lassila. A semantic context aware access control framework for secure collaborations in pervasive comput-

- ing environments. In *Collaborations In Pervasive Computing Environments 5th Intl. Semantic Web Conference*, pages 5–9. ACM Press, 2006.
- [113] A. Wegmann and G. Genilloud. The role of roles in use case diagrams. In *Proceedings of Third International Conference on The Unified Modeling Language. Advancing the Standard (UML 2000)*, pages 210–224. Springer-Verlag, 2000.
- [114] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.
- [115] T. Welte. Using state diagrams for modeling maintenance of deteriorating systems. *Power Systems, IEEE Transactions on*, 24(1):58–66, 2009.
- [116] E. Wilson. *An Introduction to Scientific Research*. Dover books explaining science. Dover Publications, 1990.
- [117] P. S. with Rob Pooley. *Using UML: software engineering with objects and components*. Object Technology Series. Addison-Wesley Longman, 1999. Updated edition for UML1.3: first published 1998 (as Pooley and Stevens).
- [118] K. Wrona and L. Gomez. Context-aware security and secure context-awareness in ubiquitous computing environments. *in XXI Autumn Meeting of Polish Information Processing Society*, 2005. [Online]. Available: <http://proceedings2005.imcsit.org/docs/75.pdf>.
- [119] T. Ye, H.-A. Jacobsen, and R. Katz. Mobile awareness in a wide area wireless network of info-stations. In *ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 109–120, 1998.
- [120] E. S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium*

BIBLIOGRAPHY

on Requirements Engineering, RE '97, pages 226–, Washington, DC, USA, 1997. IEEE Computer Society.

Appendix A

Presenting the rest of UML diagrams

This appendix targets to only present the rest of the UML modeling diagrams for both the M-learning system functions and the security requirements. This Appendix is divided into two main sections, the first one is to present the rest of modeling M-learning system functions, the second is to list the rest of capturing security requirement for M-learning system.

A.1 Modeling M-learning system functions

A.1.1 Use Case diagram

- View materials

This is to state the required CI in order to perform view materials function using Use Case diagram.

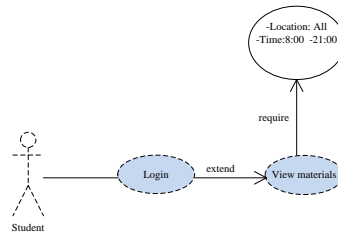


Figure A.1: View materials Use Case diagram

- Do exam

This is to state the required CI in order to perform do exam function using Use Case diagram.

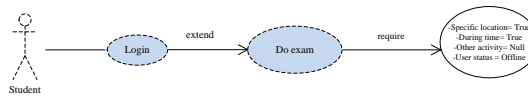


Figure A.2: Do exam Use Case diagram

- Make private chat for student

This is to state the required CI in order to make a private chat function using Use Case diagram.

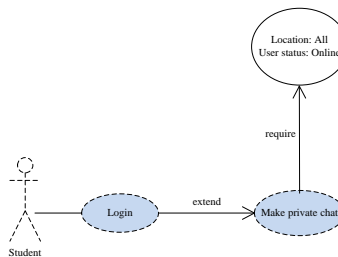


Figure A.3: Make a private chat Use Case diagram (Student)

- Make private chat for lecturer

This is to state the required CI in order to make a private chat function using Use Case diagram.

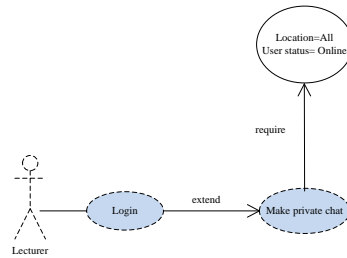


Figure A.4: Make a private chat Use Case diagram (Lecturer)

- Mark student exams

This is to state the required CI in order perform make student exams function using Use Case diagram.

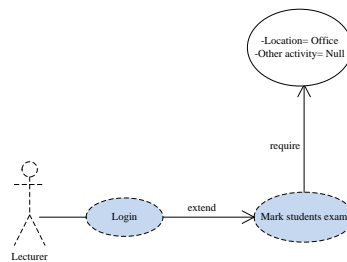


Figure A.5: Mark students exams Use Case diagram

- Upload lectures

This is to state the required CI in order to perform upload lectures function using Use Case diagram.

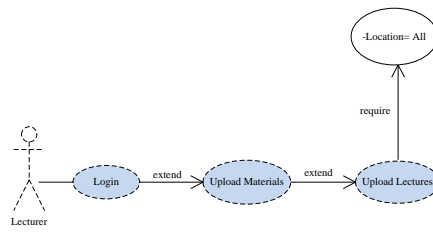


Figure A.6: Upload lectures Use Case diagram

A.1.2 Activity diagram

- View materials

This is to state the required CI in order to perform view materials function using Activity diagram.

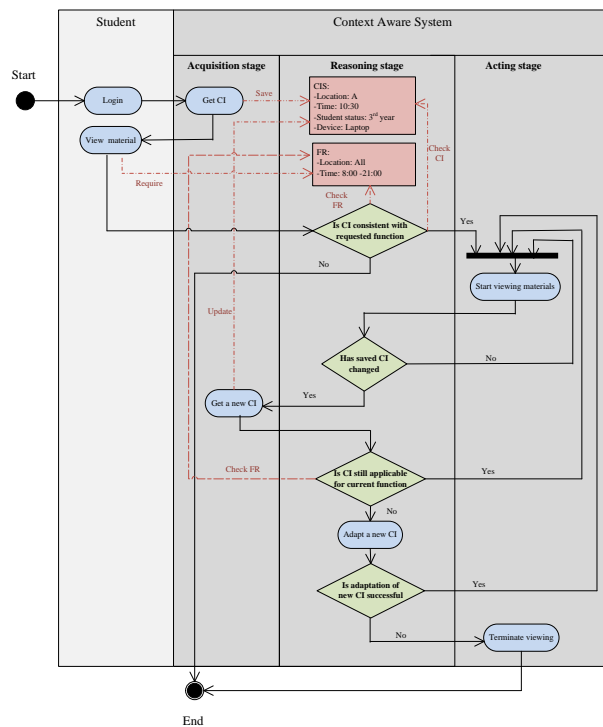


Figure A.7: View materials Activity diagram

- Do exam

This is to state the required CI in order to perform do exam function using Activity diagram.

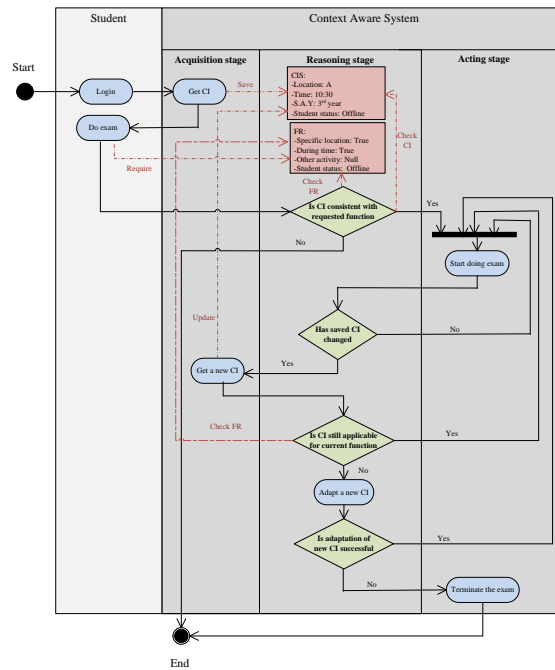


Figure A.8: Do exam Activity diagram

- Make private chat for student

This is to state the required CI in order to perform make a private chat function using Activity diagram.

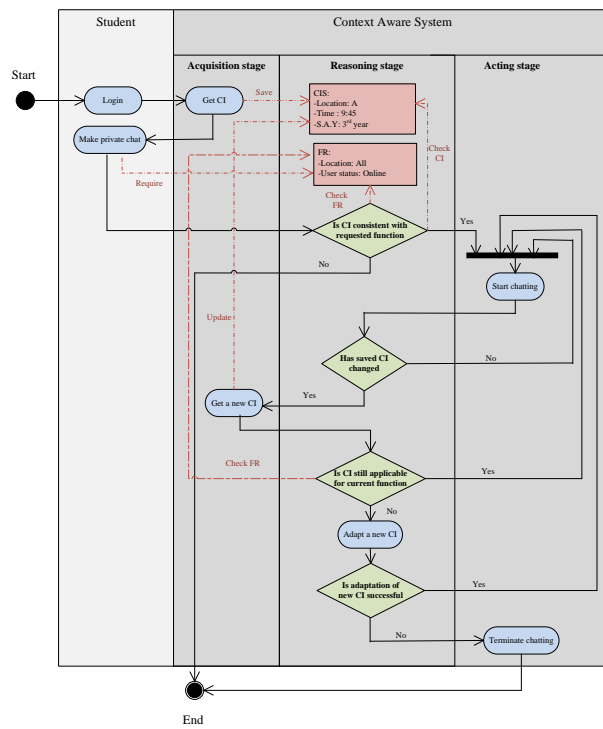


Figure A.9: Make a private chat Activity diagram (Student)

- Make private chat for lecturer

This is to state the required CI in order to perform make a private chat function using Activity diagram.

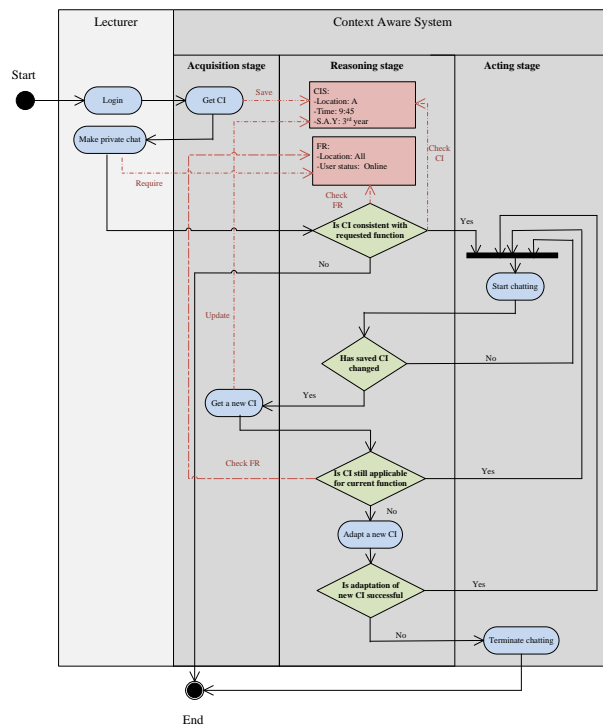


Figure A.10: Make a private chat Activity diagram (Lecturer)

- Mark student exam

This is to state the required CI in order to perform Mark student exam function using Activity diagram.

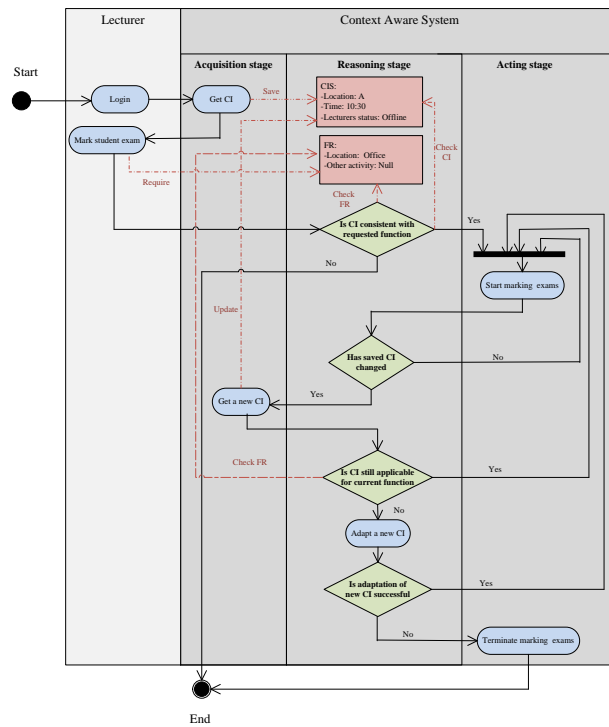


Figure A.11: Mark students exams Activity diagram

- Upload lectures

This is to state the required CI in order to perform Upload lectures function using Activity diagram.

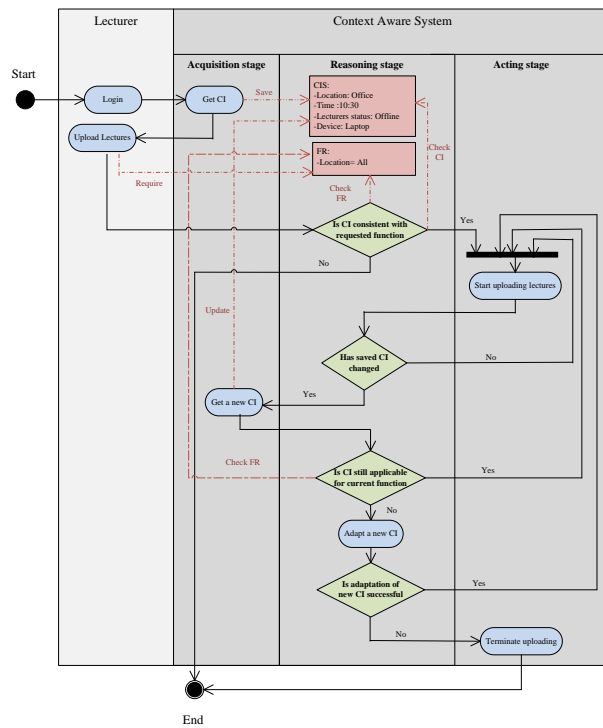


Figure A.12: Upload lectures Activity diagram

A.1.3 State diagram

- View materials

This is to state the required CI in order to perform View materials function using State diagram.

APPENDIX A. PRESENTING THE REST OF UML DIAGRAMS

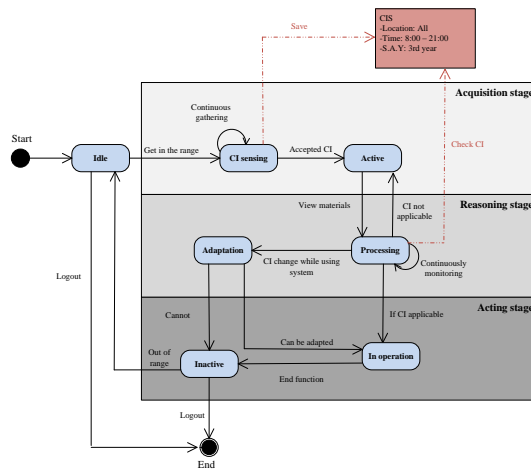


Figure A.13: View materials State diagram

- Do exam

This is to state the required CI in order to perform do exam function using State diagram.

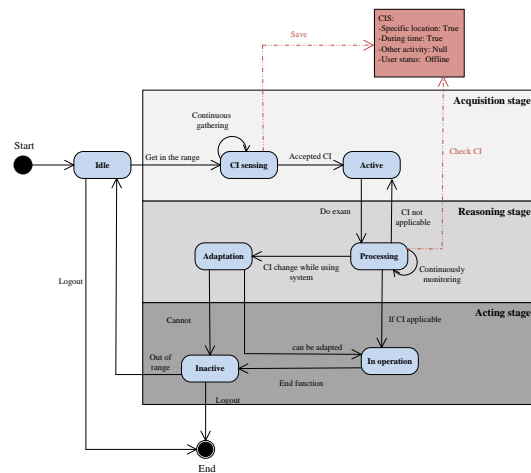


Figure A.14: Do exam State diagram

- Make private chat for student

APPENDIX A. PRESENTING THE REST OF UML DIAGRAMS

This is to state the required CI in order to perform Make private chat for student function using State diagram.

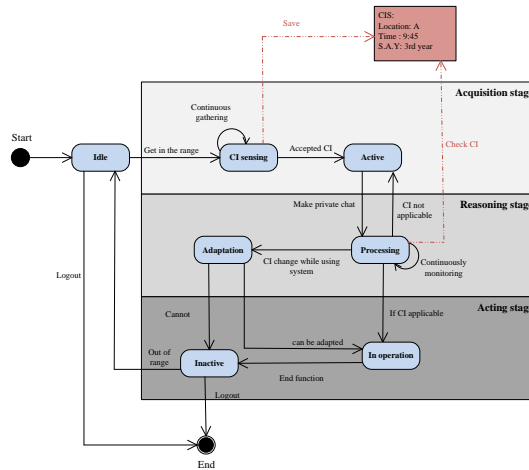


Figure A.15: Make a private chat State diagram (Student)

- Make private chat for lecturer

This is to state the required CI in order to perform Make private chat for lecturer function using State diagram.

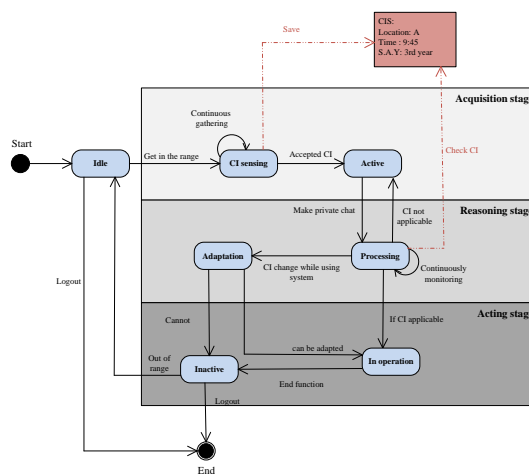


Figure A.16: Make a private chat State diagram (Lecturer)

- Mark student exam

This is to state the required CI in order to perform Mark student exam function using State diagram.

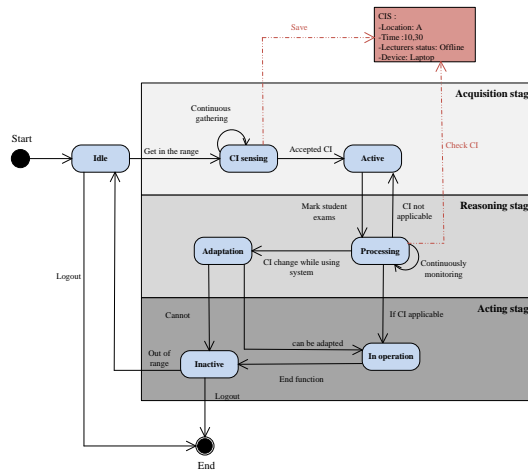


Figure A.17: Mark students exams State diagram

- Upload lectures

This is to state the required CI in order to perform Upload lectures function using State diagram.

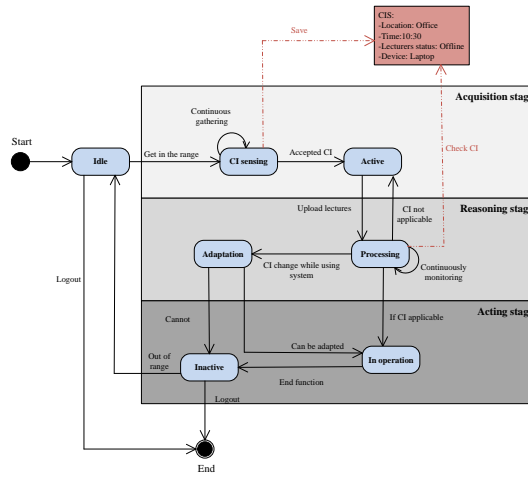


Figure A.18: Upload lectures State diagram

A.2 Capturing security requirement for M-learning system functions using UML

A.2.1 Authentication

All the diagrams that can explain the authentication mechanism using Use case diagram are already presented in Chapter 6.

A.2.2 Authorisation

A.2.2.1 Using our Enhanced Use Case diagram

This section is to explain the authorisation process using Use case diagrams.

- View materials

This is to show the authorisation process for viewing materials using Use Case diagram.

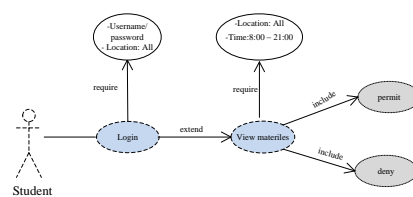


Figure A.19: View materials Authorisation Use Case

- Make private chat

This is to show the authorisation process for making a private chat using Use Case diagram.

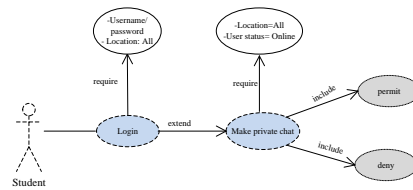


Figure A.20: Make private chat Authorisation Use Case

- Mark student exam

This is to show the authorisation process for marking student exam using Use Case diagram.

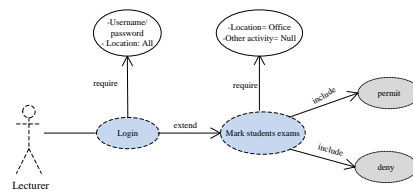


Figure A.21: Mark student exam Authorisation Use Case

- upload exam

This is to show the authorisation process for uploading exam using Use Case diagram.

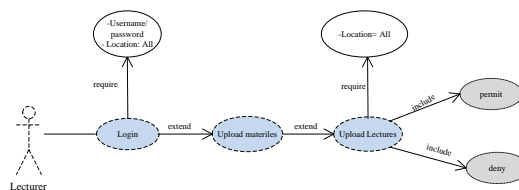


Figure A.22: Upload exam Authorisation Use Case

- Make private chat

This is to show the authorisation process for making a private chat using Use Case diagram.

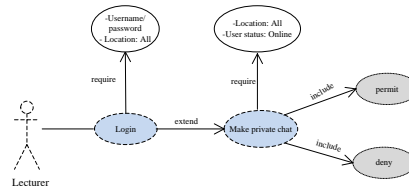


Figure A.23: Make private chat Authorisation Use Case

- Upload exam

This is to show the authorisation process for uploading exam using Use Case diagram.

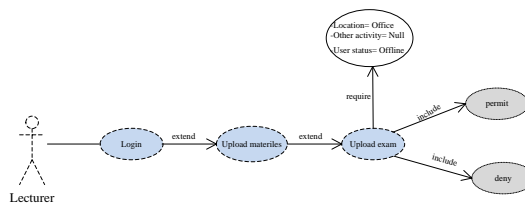


Figure A.24: Upload exam Authorisation Use Case

A.2.2.2 Using our Enhanced Activity diagram

This section is to explain the authorisation process using Activity diagrams.

- View materials

This is to show the authorisation process for viewing materials using Activity diagram.

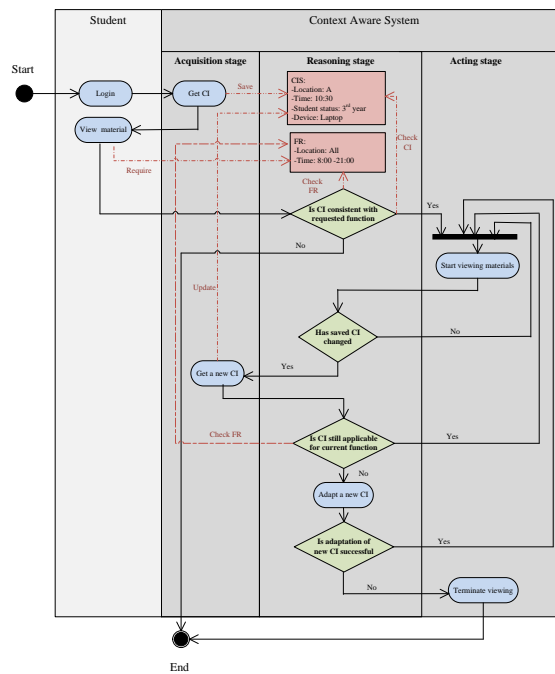


Figure A.25: View materials Authorisation Activity diagram

- Download materials

This is to show the authorisation process for downloading materials using Activity diagram.

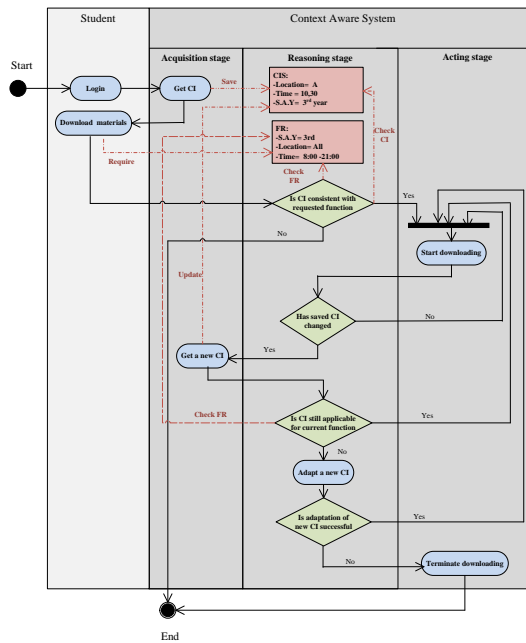


Figure A.26: Download materials Authorisation Activity diagram

- Do exam

This is to show the authorisation process for doing exam using Activity diagram.

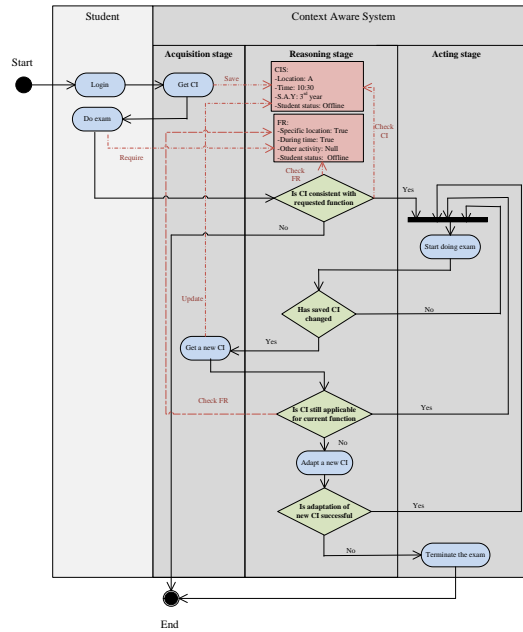


Figure A.27: Do exam Authorisation Activity diagram

- Make private chat

This is to show the authorisation process for making a private chat using Activity diagram.

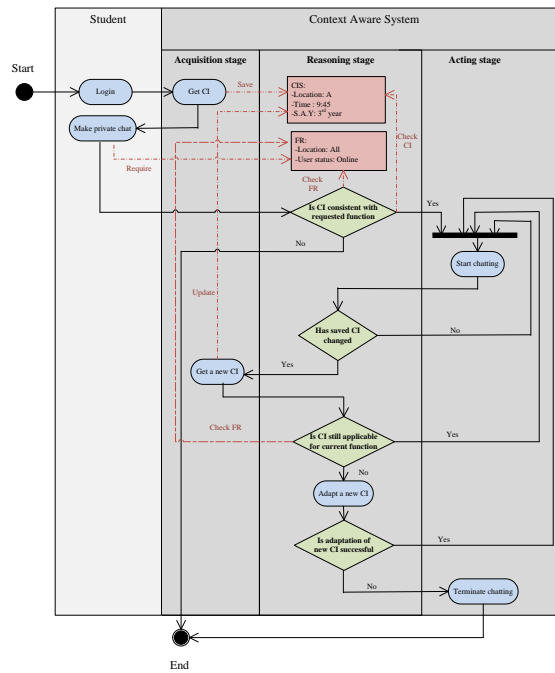


Figure A.28: Make a private chat Authorisation Activity diagram

- Upload lectures

This is to show the authorisation process for uploading lectures using Activity diagram.

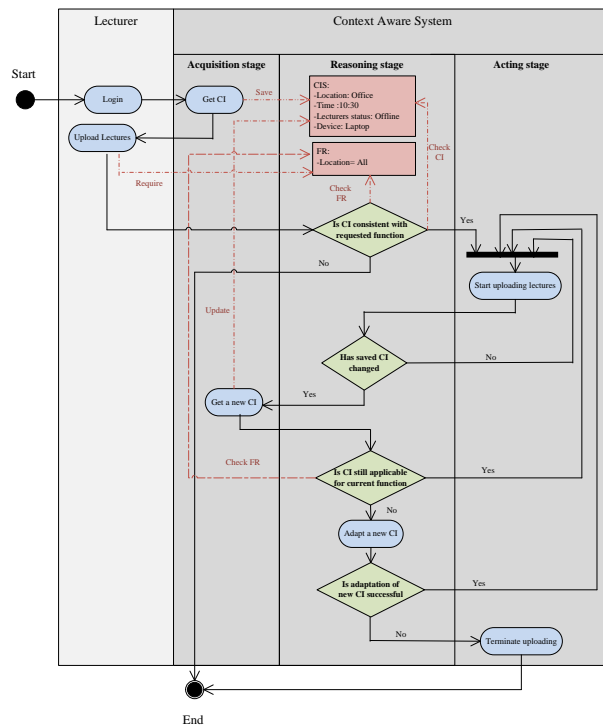


Figure A.29: Upload lectures Authorisation Activity diagram

- Mark student exams

This is to show the authorisation process for marking student exams using Activity diagram.

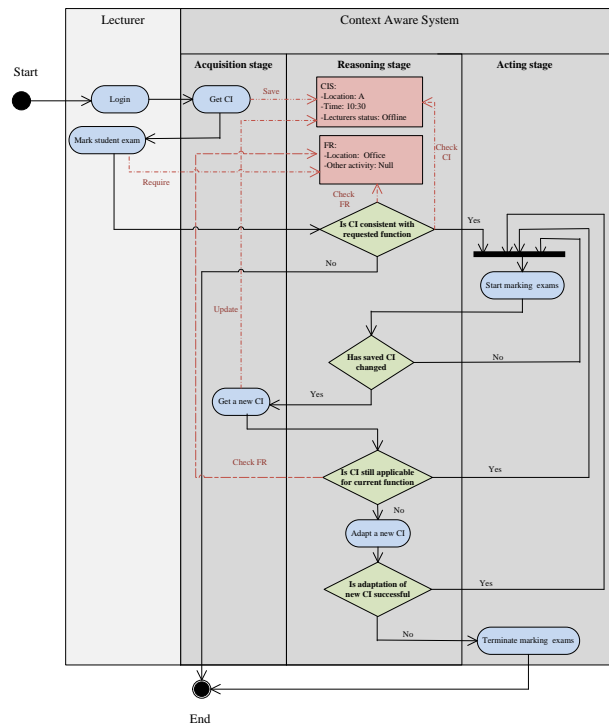


Figure A.30: Mark students exams Authorisation Activity diagram

- Make private chat

This is to show the authorisation process for make a private chat using Activity diagram.

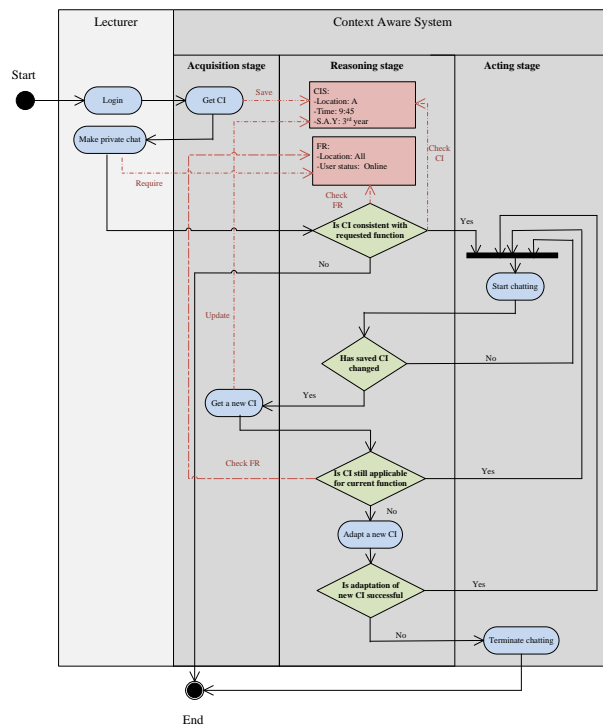


Figure A.31: Make a private chat Authorisation Activity diagram

A.2.2.3 Using our Enhanced State diagram

This section is to explain the authorisation process using State diagrams.

- view materials

This is to show the authorisation process for viewing materials using State diagram.

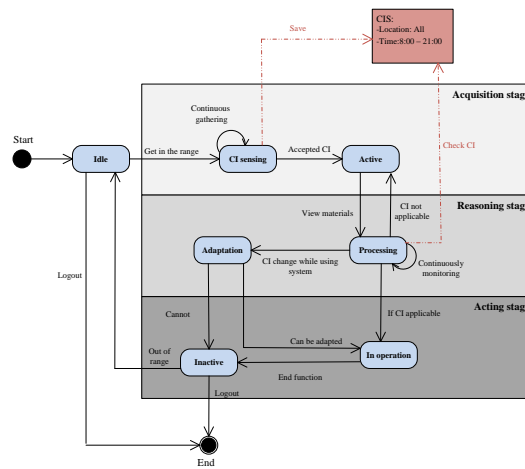


Figure A.32: View materials Authorisation State diagram

- Do exam

This is to show the authorisation process for doing exam using State diagram.

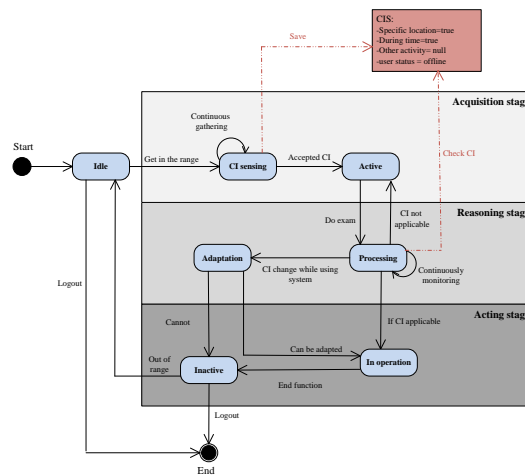


Figure A.33: Do exam Authorisation State diagram

Make private chat

This is to show the authorisation process for making private chat using State diagram.

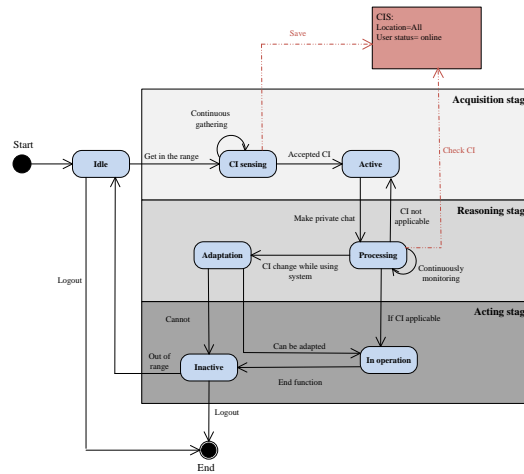


Figure A.34: Make private chat Authorisation State diagram

- Upload lectures

This is to show the authorisation process for uploading lectures using State diagram.

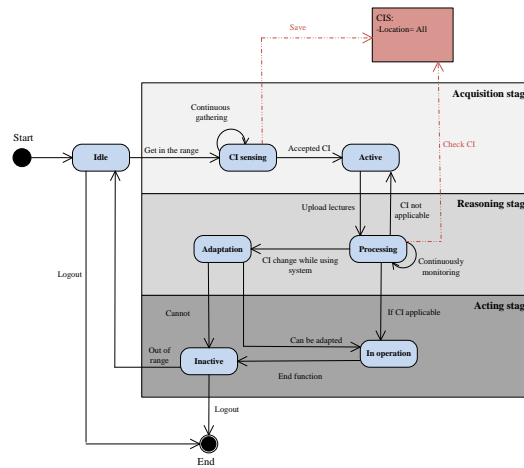


Figure A.35: Upload lectures Authorisation State diagram

- Upload exam

APPENDIX A. PRESENTING THE REST OF UML DIAGRAMS

This is to show the authorisation process for uploading exam using State diagram.

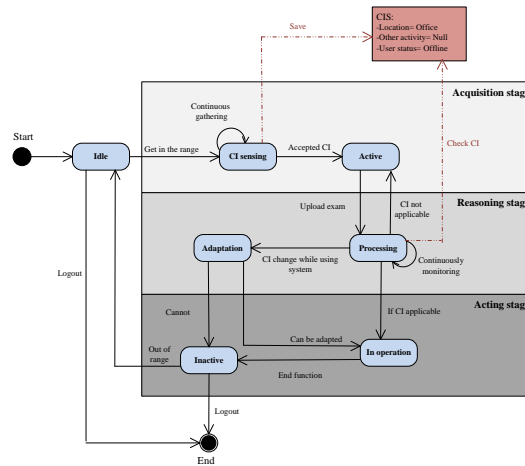


Figure A.36: Upload exam Authorisation State diagram

- Mark students exams

This is to show the authorisation process for marking student exams using State diagram.

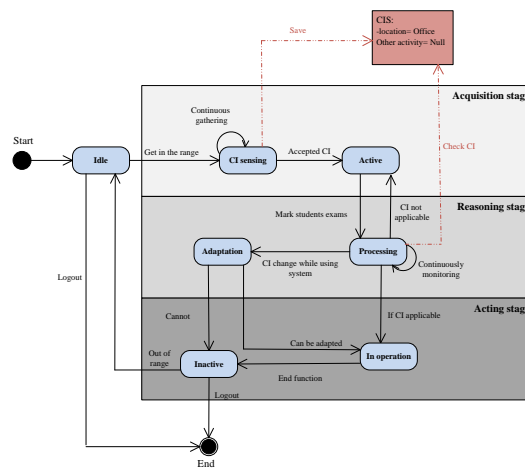


Figure A.37: Mark students exams Authorisation State diagram

A.2.3 Confidentiality

All the diagrams that can explain the confidentiality mechanism using Use case diagram are already presented in Chapter 6.

A.2.4 Integrity

A.2.4.1 Using our Enhanced Activity diagram

This section is to express the integrity process using Activity diagrams.

- View materials

This is to show the integrity process for viewing materials using Activity diagram.

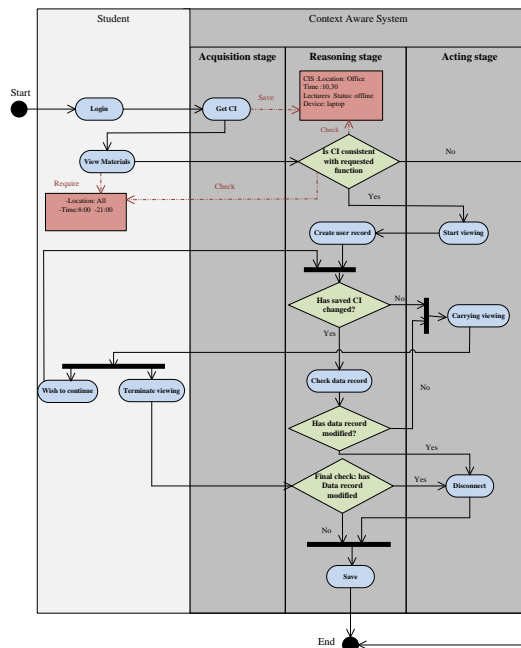


Figure A.38: View materials Integrity Activity diagram

- Make private chat

This is to show the integrity process for making a private chat using Activity diagram.

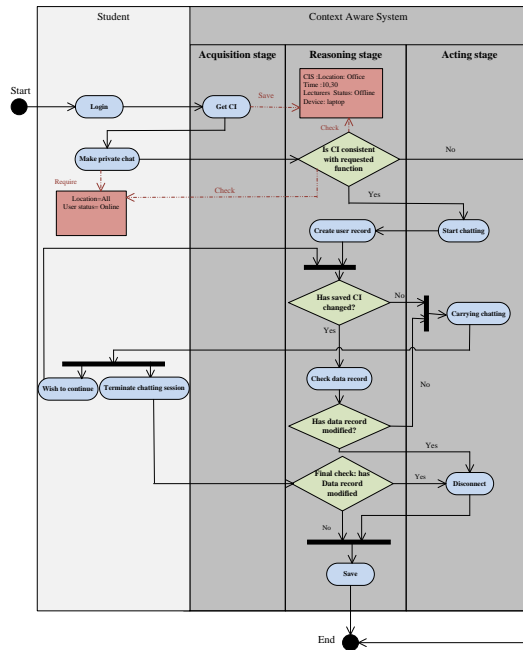


Figure A.39: Make private chat Integrity Activity diagram

- Do exam

This is to show the integrity process for doing exam using Activity diagram.

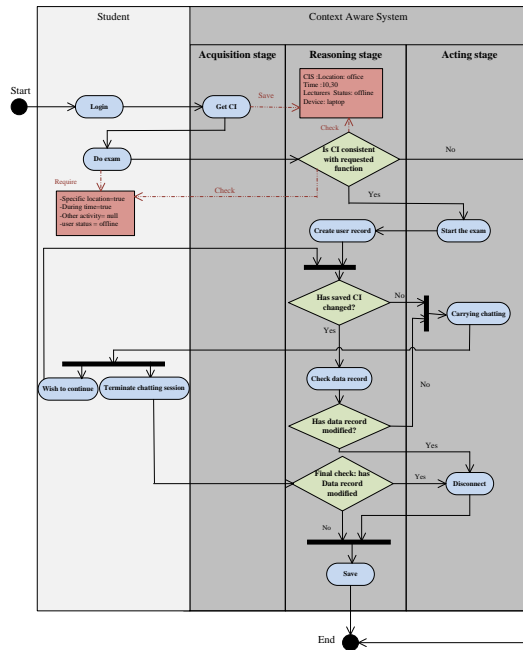


Figure A.40: Do exam Integrity Activity diagram

- Upload exam

This is to show the integrity process for uploading exam using State diagram.

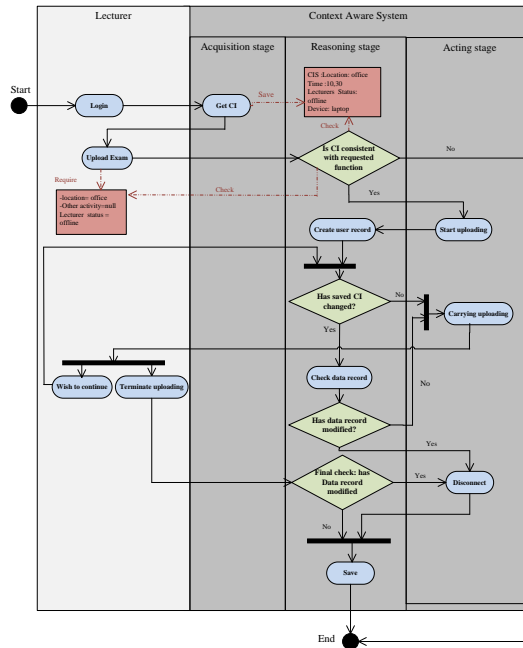


Figure A.41: Upload exam Integrity Activity diagram

- Mark students exam

This is to show the integrity process for marking students exam using Activity diagram.

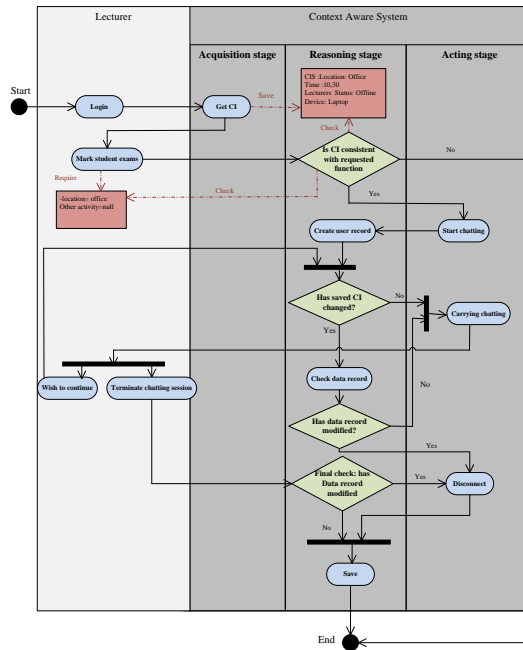


Figure A.42: Mark students exam Integrity Activity diagram

- Upload lectures

This is to show the integrity process for uploading lectures using Activity diagram.

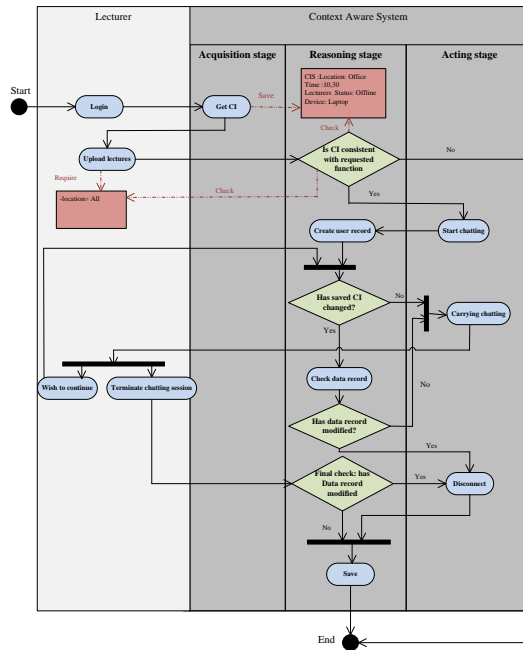


Figure A.43: Upload lectures Integrity Activity diagram

- Make private chat

This is to show the integrity process for making a private chat using Activity diagram.

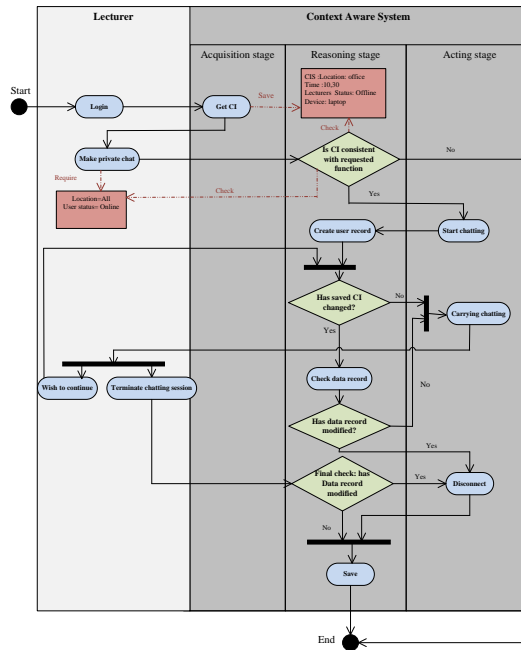


Figure A.44: Make private chat Integrity Activity diagram

A.2.5 Using our enhanced State diagram

This section is to express the integrity process using State diagrams.

- View materials

This is to show the integrity process for viewing materials using State diagram.

APPENDIX A. PRESENTING THE REST OF UML DIAGRAMS

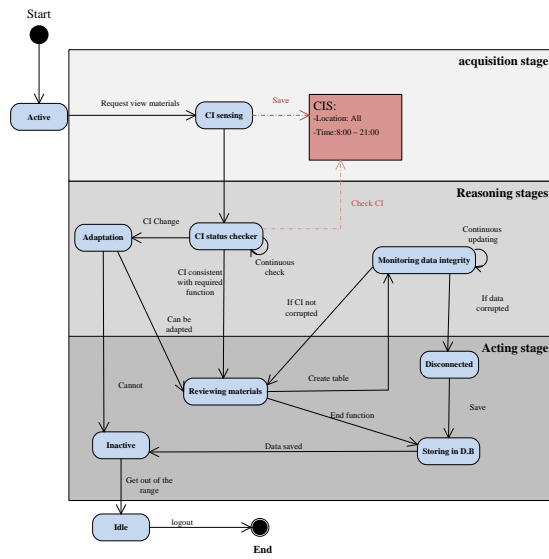


Figure A.45: View materials Integrity State diagram

- Do exam

This is to show the integrity process for doing exam using State diagram.

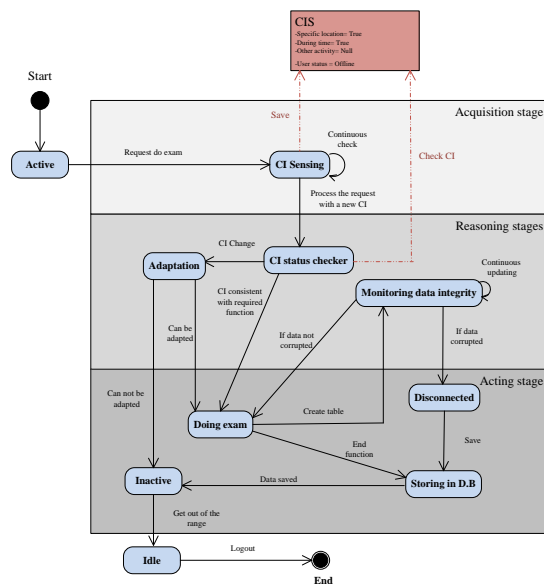


Figure A.46: Do exam Integrity State diagram

Make private chat

APPENDIX A. PRESENTING THE REST OF UML DIAGRAMS

This is to show the integrity process for making a private chat using State diagram.

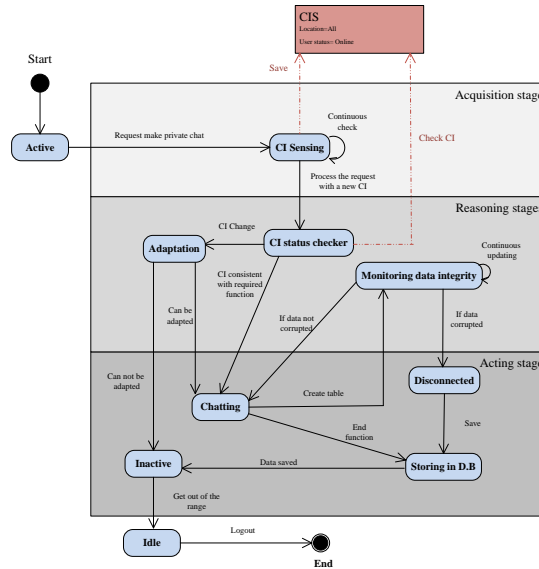


Figure A.47: Make private chat Integrity State diagram

- Upload exam

This is to show the integrity process for uploading exam using State diagram.

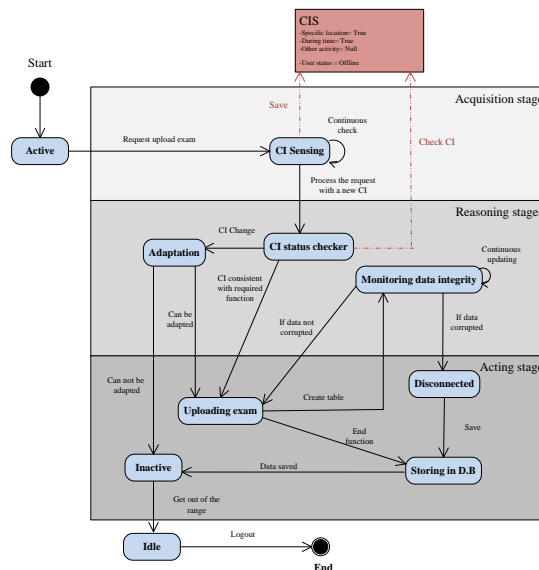


Figure A.48: Upload exam Integrity State diagram

- Upload lectures

This is to show the integrity process for uploading lectures using State diagram.

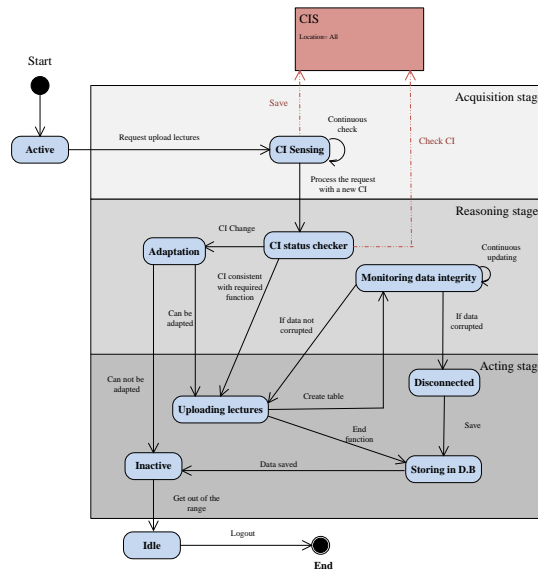


Figure A.49: Upload lectures Integrity State diagram

- Mark student exam

This is to show the integrity process for marking student exam using State diagram.

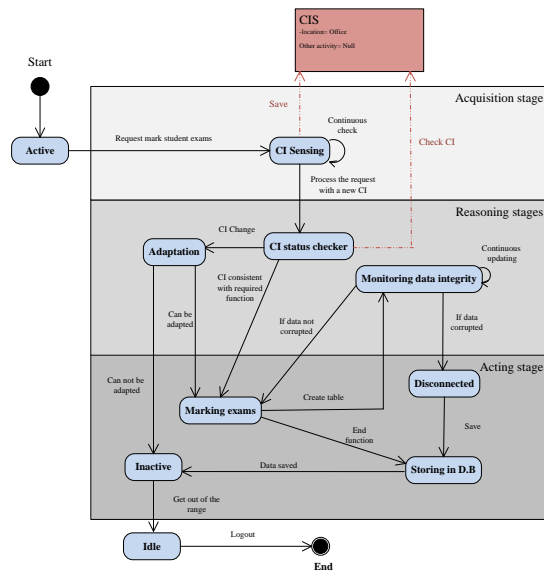


Figure A.50: Mark student exam Integrity State diagram